Extracting Information from Websites

Objective

The goal of this project is to develop a solution to extract specific information from a list of 100 websites. The information includes:

1. Social Media Links
2. Tech Stack
3. Meta Title
4. Meta Description
5. Payment Gateways (e.g., PayPal, Stripe, Razorpay)

Requirements

- **Programming Language**: Any suitable language (Python is used here)
- **Web Scraping**: Techniques to extract data from websites
- **Data Storage**: Store extracted data in a MySQL database
- **Documentation**: Clear instructions and explanations of the solution
- **Code Quality**: Well-organized, readable, and following best practices

Deliverables

1. Source code of the solution.
2. SQL script to create necessary tables in the MySQL database.
3. README file with setup instructions.
4. A brief report explaining the approach and challenges encountered.

WEB SCRAPING

Web scraping is the process of automatically extracting information from websites. It involves fetching the content of web pages and then parsing that content to extract the desired data. This technique is widely used for data collection, analysis, and integration from various web sources.

Key Components of Web Scraping

1. HTTP Requests
2. HTML Parsing
3. Data Extraction
4. Data Storage

MYSQL

MySQL is a popular relational database management system (RDBMS) that is widely used for storing and managing data. When conducting a web scraping project, MySQL can be used to store the extracted data for easy retrieval and analysis. Using MySQL for web scraping projects offers several advantages, making it a popular choice for storing and managing scraped data. Here are some key benefits:

1. Structured Data Storage
2. Scalability
3. Flexibility
4. Ease of Use
5. Backup and Recovery

Setting Up MySQL

1. **Install MySQL**:

   - Download and install MySQL from the official website.
   - Follow the installation instructions specific to your operating system.

2. **Start MySQL Server**:
   - Ensure the MySQL server is running. This can usually be done via the command line or through a MySQL management tool like MySQL Workbench.

Approach

Step 1: Setup and Libraries

- **Libraries Used**:
  - **`requests`** for making HTTP requests.
  - **`BeautifulSoup`** from **`bs4`** for parsing HTML content.
  - **`mysql-connector-python`** for MySQL database interaction.

Step 2: Define the Database Schema

- Create a database named **`scraping`**
- **Use the Database**: The scraping database is selected for subsequent operation
- **Create a Table**: A table named **`site_info`** is created within the **`scraping`** database. This table has the following columns:
  - **`id`**: An integer that auto-increments with each new entry, serving as the primary key.
  - **`url`**: A string (up to 255 characters) that stores the URL of the website (cannot be NULL).
  - **`meta_title`**: A string (up to 255 characters) that stores the meta title of the website. **`meta_description`**: A text field that stores the meta description of the website.
  - **`tech_stack`**: A text field that stores information about the technology stack used by the website.
  - **`payment_gateways`**: A text field that stores information about the payment gateways supported by the website.
  - **`social_media_links`**: A text field that stores links to the website's social media profiles

```
# creating the database
CREATE DATABASE scraping;

#using that database
USE scraping;

#creating the table
CREATE TABLE site_info (
    id INT AUTO_INCREMENT PRIMARY KEY,
    url VARCHAR(255) NOT NULL,
    meta_title VARCHAR(255),
    meta_description TEXT,
    tech_stack TEXT,
    payment_gateways TEXT,
    social_media_links TEXT
);
```

Step 3: Web Scraping Logic

- **Fetch Website Content**:
- Use `requests.get(url)` to fetch the HTML content of the website.
- **Parse HTML Content**:
- Use `BeautifulSoup` to parse the HTML content and extract the required information.
- **Extract Information**:
- **Social Media Links**: Search for anchor tags (`<a>`) containing URLs to social media platforms (e.g., Facebook, Twitter).
- **Tech Stack**: Identify common tech stack indicators (e.g., comments in the HTML, specific JavaScript libraries).
- **Meta Title and Description**: Extract from `<title>` and `<meta name="description">` tags.
- **Payment Gateways**: Look for keywords or specific scripts related to payment processors.
- **Store Data**:
- Insert the extracted information into the MySQL database using `mysql-connectorpython`

Step 4: Code Structure

The code is organized into functions:

- `meta_info(url)`:
- Fetches the HTML content of the given URL.
- Parses the HTML to extract the title and meta description.
- Returns the title and meta description.
- `social_media_links_extraction(soup)`:
- Takes a `BeautifulSoup` object (`soup`) as input.
- Finds all anchor tags (`<a>`) with `href` attributes.
- Filters the links to include only those pointing to Facebook, Twitter, LinkedIn, or Instagram.
- Returns a comma-separated string of these social media links.

- **`tech_stack_extraction(soup)`:**

- Takes a **`BeautifulSoup`** object (`soup`) as input.
- Finds all script tags (`<script>`) with `src` attributes.
- Identifies the use of jQuery and Bootstrap by checking the `src` attribute.
- Returns a comma-separated string of detected technologies.

- **`payment_gateways_extraction(soup)`:**

- Takes a BeautifulSoup object (`soup`) as input.
- Searches the text content for mentions of PayPal, Stripe, and Razorpay.
- Returns a comma-separated string of detected payment gateways.

```python
def meta_info(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')
    title = soup.find('title').text if soup.find('title') else ''
    description = soup.find('meta', attrs={'name': 'description'})['content'] if soup.find('meta', attrs={'name': 'description'}) 
    return title, description

def social_media_links_extraction(soup):
    social_links = []
    for link in soup.find_all('a', href=True):
        href = link['href']
        if 'facebook.com' in href or 'twitter.com' in href or 'linkedin.com' in href or 'instagram.com' in href:
            social_links.append(href)
    return ','.join(social_links)

def tech_stack_extraction(soup):
    tech_stack = []
    for script in soup.find_all('script', src=True):
        src = script['src']
        if 'jquery' in src:
            tech_stack.append('jQuery')
        elif 'bootstrap' in src:
            tech_stack.append('Bootstrap')
    return ','.join(tech_stack)

def payment_gateways_extraction(soup):
    gateways = []
    if 'paypal.com' in soup.text:
        gateways.append('PayPal')
    if 'stripe.com' in soup.text:
        gateways.append('Stripe')
    if 'razorpay.com' in soup.text:
        gateways.append('Razorpay')
    return ','.join(gateways)
```

- **`scrape_website(url)`:** • Fetches the HTML content of the given URL and parses it into a BeautifulSoup object.
- Calls the `meta_info`, `social_media_links_extraction`, `tech_stack_extraction`, and `payment_gateways_extraction` functions to extract relevant information.
- Returns a dictionary containing the URL, meta title, meta description, tech stack, payment gateways, and social media links.

- **`store_data_in_db(data)`:**

- Establishes a connection to a MySQL database.
- Prepares an SQL `INSERT` statement to insert the scraped data into the `site_info` table.

- Executes the SQL statement and commits the transaction.
- Closes the database connection.

```
def scrape_website(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')
    meta_title, meta_description = meta_info(url)
    social_media_links = social_media_links_extraction(soup)
    tech_stack = tech_stack_extraction(soup)
    payment_gateways = payment_gateways_extraction(soup)
    return {
        'url': url,
        'meta_title': meta_title,
        'meta_description': meta_description,
        'tech_stack': tech_stack,
        'payment_gateways': payment_gateways,
        'social_media_links': social_media_links
    }
```

```
def store_data_in_db(data):
    conn = mysql.connector.connect(
        host='localhost',
        user='root',
        password='mysql',
        database='scraping'
    )
    cursor = conn.cursor()
    sql = """INSERT INTO site_info (url, meta_title, meta_description, tech_stack, payment_gateways, social_media_links)
             VALUES (%s, %s, %s, %s, %s, %s)"""
    val = (data['url'], data['meta_title'], data['meta_description'], data['tech_stack'], data['payment_gateways'], data['social_
    cursor.execute(sql, val)
    conn.commit()
    cursor.close()
    conn.close()
```

- **`main(websites)`:**
- Iterates through a list of websites.
- Calls `scrape_website` for each website to scrape data.
- Calls `store_data_in_db` to store the scraped data in the database.

```
def main(websites):
    for website in websites:
        data = scrape_website(website)
        store_data_in_db(data)
```

```
sites = [
        '' #websites link
    ]
main(sites)
```

Step 5: Documentation

A README file is provided with:

- **Project Overview:** Briefly describe the purpose and scope of the project**.**
- **Setup Instructions**: How to install dependencies, set up the database, and run the script.
- **MySQL Integration**: How to integrate with mysql through python.

Challenges

1. **Dynamic Content**: Some websites load content dynamically using JavaScript, which requests and `BeautifulSoup` cannot handle. Tools like Selenium or Scrapy could be alternatives but are more complex and resource-intensive.
2. **Different Website Structures**: Websites have varied structures, making it challenging to create a one-size-fits-all scraping solution. Heuristics and flexible code were required to handle different HTML structures.
3. **Rate Limiting**: To avoid being blocked by websites, the script includes delays between requests and error handling for failed requests.
4. **Data Extraction Complexity**: Extracting structured data from unstructured or semistructured HTML can be challenging, especially when dealing with nested elements, inconsistent formatting, or multiple data sources on the same page.
5. **Legal and Ethical Considerations**: Web scraping raises ethical concerns related to data ownership, copyright infringement, and terms of service violations. Legal regulations such as GDPR in Europe and DMCA in the United States add complexity.

Conclusion

The solution will successfully extract the required information from a list of 100 websites and will store it in a MySQL database. The code is well-documented and follows best practices, ensuring it is maintainable and extensible. Future improvements could include handling dynamic content more effectively and optimizing the extraction heuristics for better accuracy.

References

- `BeautifulSoup` **Documentation**: [Official Documentation]
(https://www.crummy.com/software/BeautifulSoup/bs4/doc/)

- `requests` **Library:** [GitHub Repository] (https://github.com/psf/requests)

- `MySQL` **Documentation:** [Official Documentation] (https://dev.mysql.com/doc/)