DA5402 Assignment 1

Name: Sukriti Shukla
Roll No: ED20B067

February 2, 2025

Module 1: Setting up the Chrome Driver and Scraping Home Page

File: module1.py

The `setup_driver` function in the script configures and launches a headless Chrome browser for automated web browsing. The `scrape_page` function uses Selenium to open a website, wait for it to load, and then save the entire HTML content of the page to a file. This is useful for collecting data from websites automatically.

Module 2: Scraping Home Page and Getting Links

File: module2.py

The `scrape_top_stories_link` function finds and retrieves the URL for the "Top Stories" section from a specified homepage, using Selenium for web navigation. The driver navigates to the `home_url`, the URL of the homepage from which you want to scrape the "Top Stories" link.

Wait up to 20 seconds or until at least one hyperlink (`<a>` tag) is present on the page, ensuring that the page has loaded sufficiently to proceed with finding the link.

Locate Top Stories Link:

The function prepares to locate the link by converting the `top_stories_text` to lowercase to facilitate a case-insensitive search. Constructs an XPath expression to find an `<a>` element whose text includes the normalized `top_stories_text`. `WebDriverWait` is used to wait up to 10 seconds for the presence of an element that matches the XPath query. Once the link element is found, the function retrieves the `href`, which contains the URL of the "Top Stories" section.

The function returns the URL of the "Top Stories" link, which can be used for further processing or navigation.

Module 3: Dynamic Web Content Interaction

File: module3.py

Scroll_page - This automates the scrolling of a webpage to load dynamic content, which is lazy-loaded as the user scrolls. The function repeatedly scrolls to the bottom of the page using JavaScript (`window.scrollTo`). After each scroll, it pauses (`time.sleep`)

to allow the page content to load and then checks if the page height has increased (`document.body.scrollHeight`). The loop continues until the page height no longer increases after a scroll, indicating that all content has likely been loaded.

Download_image - This manages the downloading of images from specified URLs, catering to the need to store visual content locally: The function uses `requests.get` to fetch the image from the URL. It handles the response by checking the HTTP status code. If the image is successfully retrieved, it writes the binary content of the image to a file in the specified directory.

Scrape_top_stories_page - This function orchestrates the scraping of "Top Stories" from a news website, leveraging the browser driver to interact with the web page dynamically:

It waits for the initial set of articles to be present on the page and checks how many articles are loaded initially. It calls `scroll_page` to ensure that any additional articles that load upon scrolling are also captured. After scrolling, it retrieves an updated list of articles and extracts crucial data from each article, such as the headline and the URL of the thumbnail image. It uses `extract_thumbnail_url` to obtain the best resolution image URL and then downloads the image using `download_image`.

Module 4: Database Interaction

File: module4.py

get_mongo_database - This establishes a connection to a MongoDB instance. It attempts to connect to a MongoDB server running locally on the default port (27017) and access a database named "news_scrapper."

insert_image - This handles the storage of image data in MongoDB. If the image data is successfully retrieved (non-null), it inserts a document into the images collection containing the image data (stored as a BSON binary), and the image URL.

Insert_data - It inserts a new document into the articles collection with the article's headline, thumbnail URL, and the scrape timestamp. It then attempts to download the image from the thumbnail_url. If the image is successfully downloaded, it calls `insert_image` to store the image data along with the article_id, creating a relational link between the article and its thumbnail image.

Module 5: Data Management and De-duplication

File: module5.py

generate_article_hash - This generates a unique hash for each article using the MD5 hashing algorithm.

The headline is converted to lowercase and any leading or trailing whitespace is removed. The normalized headline is concatenated with the thumbnail URL (if available). This combination adds a layer of specificity to the hash, as two articles might have the same headline but different thumbnails. The combined string is then encoded into bytes and passed to the MD5 hashing function to generate a unique hash.

Insert_data - This function manages the insertion of articles into the MongoDB database:

It first generates a hash for the article using the aforementioned `generate_article_hash` function. Before inserting the new article, it checks the articles collection for an existing

document with the same hash. If such a document exists (`articles_collection.find_one({"article_hash"})` returns a non-null result), If no duplicate is found, the article data, along with the newly generated hash, is inserted into the articles collection.

Module 6: Pipeline Orchestration and Logging

File: module6.py

Setup Logging - Logs are directed both to a file and the console, ensuring visibility of the pipeline's operations.

Orchestrate Pipeline - Module 1: It scrapes the homepage of a given URL and saves the HTML content, logging the outcome. Module 2: Extracts the "Top Stories" link from the homepage. If the link isn't found, the pipeline logs an error and halts further execution. Module 3: Uses the Top Stories link to scrape articles, capturing headlines and thumbnails. Failures in data extraction lead to an early termination, while success is logged. Module 4 and 5: interact with a database to store the scraped data and remove duplicates.