GitHub User Data Scraper Documentation

Code Source: GitHub Repo Link Click Here

- **IndexDatabase.py**
    - This is the Main Code which receive the GitHub Threads URL and create the Database and CSV File for the entire data

Data Source: Drive Link Click Here

- Click Here to get Excel And DataBase File for Final Data approx. 2148

Overview

This Python script is designed to scrape user profile data from GitHub. The script retrieves information such as name, username, bio, location, social media links, and follower/following counts. It processes multiple pages of user data using threading to speed up the process, stores the results in a database, and finally exports the data to a CSV or Excel file.

Code Flow Explanation

1. Imports and Dependencies

- **requests**: Handles HTTP requests to GitHub.
- **json**: Provides tools for working with JSON data.
- **BeautifulSoup**: Parses HTML content from GitHub.
- **pandas**: Used to create and manipulate data in CSV and Excel formats.
- **ThreadPoolExecutor**: Manages concurrent execution of functions.
- **alive_bar**: Displays a progress bar during data scraping.
- **threading**: Supports thread-based parallelism.
- **dataset**: Simplifies database interactions.
- **traceback**: Provides detailed exception handling.

2. Function: retrievePageUrls(url, URL_List)

- **Purpose**: Extracts individual user profile URLs from a given GitHub page.
- **Process**:
    1. Sends an HTTP GET request to the provided URL.
    2. Parses the HTML content to locate user profile links within an ordered list (<ol>).
    3. Appends each found URL to the URL_List.

3. Function: getCount(url, Main_Url)

- **Purpose**: Determines the total number of pages containing user data.
- **Process**:
    1. Sends an HTTP GET request to the provided URL.
    2. Finds the total number of items (e.g., stars, followers) related to the Main_Url.
    3. Calculates the number of pages required by dividing the total count by 48 (items per page) and rounds up.

4. Function: retrieveIndividualPages(url, index)

- **Purpose**: Scrapes detailed information from individual GitHub user profile pages.
- **Process**:
    1. Sends an HTTP GET request to the provided user profile URL with custom headers.
    2. Parses the HTML content to extract details such as:
        - Name, Username, Bio, Location, Company
        - Social Media Links (Facebook, Twitter, LinkedIn, Instagram)
        - Personal Website, Followers, Following
    3. Returns the data as a dictionary if successful; otherwise, it returns the URL indicating a failed attempt.
- **Error Handling**: Captures and logs exceptions using traceback.

5. Function: ThreadingPages(PageUrlsData, base_link)

- **Purpose**: Manages concurrent scraping of multiple user profile pages.
- **Process**:
    1. Initializes an empty list for processes, results, and missing URLs.
    2. Uses ThreadPoolExecutor to handle concurrent execution of the retrieveIndividualPages function.
    3. Submits tasks for each URL in PageUrlsData.
    4. Collects results as tasks complete, distinguishing between successful scrapes (dictionaries) and failures (URLs).
    5. Returns a list containing the results and missing URLs.

6. Function: Final_updation_inDB(UpdatedTable, list)

- **Purpose**: Inserts scraped data into a database table.
- **Process**:
    1. Inserts the list of dictionaries (scraped data) into the specified UpdatedTable.
    2. Prints a confirmation message upon successful insertion.

7. Function: makeCSV(Data_List, path)

- **Purpose**: Saves the scraped data to a CSV file.
- **Process**:
    1. Converts the list of dictionaries (Data_List) into a Pandas DataFrame.
    2. Exports the DataFrame to a CSV file at the specified path.

8. Function: sql_table_to_csv(UpdatedTable, csv_file_path)

- **Purpose**: Exports data from a database table to an Excel file.
- **Process**:
    1. Retrieves all rows from the UpdatedTable as a list of dictionaries.
    2. Converts the list into a Pandas DataFrame.
    3. Saves the DataFrame to an Excel file at the specified csv_file_path.
    4. Prints a confirmation message upon successful export.

9. Function: main()

- **Purpose**: Orchestrates the entire scraping process.
- **Process**:
    1. Prompts the user to input a GitHub URL.
    2. Validates the URL to ensure it contains "github.com".
    3. Constructs the main URL and determines the number of pages to scrape using getCount.
    4. Calls retrievePageUrls for each page to gather all profile URLs.
    5. Uses ThreadingPages to scrape data concurrently from all gathered URLs.
    6. Connects to a SQLite database and updates the FinalData table with the scraped data.
    7. Exports the final data to an Excel file and missing data to a CSV file.

10. Execution

- The script starts execution with the main() function, which is run if the script is executed directly (not imported as a module).

Usage Instructions

1. **Run the Script**:
    - Ensure all dependencies are installed: pip install requests beautifulsoup4 pandas concurrent.futures alive-progress dataset traceback.
    - Run the script in your terminal or command prompt: python script_name.py.

2. **Input a GitHub URL**:
    - When prompted, input a GitHub URL for the repository or user whose data you want to scrape.
3. **Wait for Completion**:
    - The script will display a progress bar as it scrapes data.
    - Upon completion, the data will be saved to a database and exported to an Excel or CSV file.

Conclusion

This script efficiently scrapes GitHub user data using threading to maximize speed and outputs the results in a user-friendly format. It can be modified to scrape other types of data or adapted to different websites with similar structures.