Documentation: Money House Web Scraper

Code Source: GitHub Repo Link Click Here

- **IndexDatabase.py**
  - This is the Main Code which follow the database approach that the client have been mentioned
- **IndexDatabaseTesting.py**
  - This is the same approach but for Testing purpose for short range testing as the whole code will run for 726k Time but this code will run for 50 data only(Only For Testing Purpose
- **indexCSVPart1.py**
  - This is the code which follow the csv approach and download only CSV file that contain Page Url and other Information for further process.
- **indexCSVPart2.py**
  - This is the 2nd Part of CSV Code approach which further process each individual page and finally created the Final Data in CSV Format
- **indexDataTesting1.py**
  - This Code will test how many data is present base on the alphabets Search
- **indexDataTesting2.py**
  - This Code will test how many data is present base on the **Rechtsform** and **Kanton**

Data Source: Drive Link Click Here

- Click Here to get CSV and Excel for Final Data approx. 726k

User Usage Summary Report

The final code is designed to operate efficiently and handle large-scale web scraping tasks with built-in safeguards to prevent unnecessary repetition of steps. Here's how it works:

1. **Initial Run**:
   - When the user runs the code for the first time, it retrieves all URLs and creates a database with a Scraped column initially set to False.
   - The code then iterates over each URL, scraping the relevant data, and marking the Scraped column as True for each processed URL.
   - After all URLs are processed, the code generates a CSV file from the newly created database.
2. **Performance Considerations**:
   - In the initial step, approximately 14,000 requests are made, taking around 30 minutes to retrieve all the URL links.
   - To optimize performance, the code is designed to avoid repeating this time-consuming step if it has already been completed.
3. **Subsequent Runs**:

- If the user runs the code again, the script checks for the existence of the URL database.
- If the database is found in local storage, the code skips the first step (retrieving URLs) and proceeds directly to processing the individual pages. This ensures that time is not wasted on redundant operations.

4. **Resuming Operations**:
   - If the user encounters an interruption or the script stops during the retrieval of individual pages, the code is capable of resuming from where it left off. It continues processing the remaining URLs without starting from the beginning, ensuring a seamless and efficient workflow.

This approach ensures that the code is both time-efficient and resilient, allowing users to restart and resume operations without losing progress or wasting time.

Code Flow Summary

The script is divided into several key functions that work together to scrape URLs, store data, scrape individual pages for detailed information, and finally export the collected data to a CSV file. The main steps are:

1. **URL Scraping and Storage**: Scrape URLs based on legal form and alphabetical parameters, then store them in an SQLite database.
2. **Individual Page Scraping**: Extract detailed information from each scraped URL and update the database.
3. **Data Export**: Convert the final scraped data from the database into a CSV file.

Detailed Function Explanations

1. ScrapeUrl(headers, internal_params, legal_form, name_letter, i)

- **Purpose**: This function performs the initial scraping of company URLs based on the provided parameters.
- **Parameters**:
  - headers: HTTP headers to mimic a real browser.
  - internal_params: Query parameters for the HTTP request.
  - legal_form: The legal form of the companies to filter.
  - name_letter: The starting letter of the company names.
  - i: An index to track progress.
- **Process**:
  - Sends a GET request to the URL.
  - Parses the JSON response to extract company details.
  - Constructs a list of dictionaries with the company data.

- Returns this list for further processing.

2. store_data_in_db(Main_list, database, databaseTable)

- **Purpose**: Stores the scraped URL data into an SQLite database.
- **Parameters**:
  - Main_list: A list of dictionaries containing the scraped data.
  - database: The name of the SQLite database file.
  - databaseTable: The name of the table where the data will be stored.
- **Process**:
  - Connects to the SQLite database.
  - Inserts or updates each entry in the Main_list into the specified database table using the upsert method to avoid duplicates.

3. threadingScrapeUrl(database, databaseTable)

- **Purpose**: Manages the multithreaded scraping of URLs and stores the results in the database.
- **Parameters**:
  - database: The name of the SQLite database file.
  - databaseTable: The name of the table where the data will be stored.
- **Process**:
  - Iterates over predefined legal forms and alphabetical letters to generate requests.
  - Uses a thread pool to handle multiple requests concurrently.
  - Collects the results and stores them in the database.

4. read_db_to_dict_list(table)

- **Purpose**: Reads and converts the database entries into a list of dictionaries, filtering only rows where the Scraped column is False.
- **Parameters**:
  - table: The database table to read from.
- **Process**:
  - Fetches rows from the table where Scraped is False.
  - Converts the rows to a list of dictionaries for further processing.

5. retriveIndividualPages(obj1, index)

- **Purpose**: Scrapes detailed information from each company's page.

- **Parameters**:
    - obj1: A dictionary representing a single company entry.
    - index: The index of the current item for tracking progress.
- **Process**:
    - Sends a GET request to the company's detailed page.
    - Parses the HTML content using BeautifulSoup to extract various details like company age, revenue, management, etc.
    - Returns a dictionary with the extracted data or a list of values if the scraping fails.

6. threadingIndividualPages(ScrapedData, Scrapedtable, UpdatedTable)

- **Purpose**: Manages the multithreaded scraping of individual company pages and updates the database.
- **Parameters**:
    - ScrapedData: The data to be scraped, where Scraped is False.
    - Scrapedtable: The table where the Scraped status is stored.
    - UpdatedTable: The table where the detailed data will be stored.
- **Process**:
    - Iterates over the scraped data, launching a thread for each page to scrape detailed information.
    - Updates the Scraped status in the database and inserts the detailed data into the UpdatedTable.

7. sql_table_to_csv(UpdatedTable, csv_file_path)

- **Purpose**: Exports the data from the SQLite database table to a CSV file.
- **Parameters**:
    - UpdatedTable: The database table containing the final detailed data.
    - csv_file_path: The path where the CSV file will be saved.
- **Process**:
    - Converts the table data to a Pandas DataFrame.
    - Saves the DataFrame to a CSV file.

8. main()

- **Purpose**: The main entry point of the script that coordinates the entire scraping and data export process.
- **Process**:

- Checks if the database exists; if not, initiates the URL scraping process.
- Reads the database for unscraped URLs.
- Initiates the detailed scraping of individual pages.
- Converts the final data to a CSV file.

Code Flow

1. **Initial Check and URL Scraping**:
   - The script begins by checking if the SQLite database exists.
   - If not, it scrapes URLs and stores them in the database.
2. **Individual Page Scraping**:
   - The script reads unscraped URLs from the database and initiates the scraping of detailed information for each page.
   - It updates the database to mark URLs as scraped.
3. **Data Export**:
   - Once all pages have been scraped, the script exports the final detailed data to a CSV file.

Summary

This script is designed for efficient web scraping with robust error handling and multi-threading capabilities. It scrapes company data from a specified website, stores it in a local SQLite database, and exports the final data to a CSV file. The script is modular, allowing easy adjustments and expansions for different scraping tasks.