Some points that I worked on:

1. Add Docstrings to Functions

Summary: Enhance code documentation by adding meaningful docstrings to each function.

Description: For better code readability and maintainability, include detailed docstrings for each function. These docstrings should describe the purpose of the function, its input parameters, expected output, and any exceptions it may raise.

2. Improve Code Robustness

Summary: Implement error handling and validation to reduce the likelihood of code-breaking.

Description: Enhance the code's robustness by adding error-handling mechanisms such as try-except blocks. Validate inputs and handle potential exceptions gracefully to prevent crashes or unexpected behavior.

3. Add Non-Callable Functions

Summary: Introduce non-callable utility functions like **json_detect_error** and **json_parse_result** to improve code modularity.

Description: Create utility functions that are not meant to be called directly but serve as helpers within the codebase. These functions can handle specific tasks like detecting errors in JSON data (json_detect_error) or parsing JSON results (json_parse_result), making the code more modular and maintainable.

4. Implement captcha_solver_function(for placeholder only)

Summary: Integrate a captcha_solver_function to automate captcha solving.

Description: Develop a dedicated function, captcha_solver_function, for now this function only returns placeholder values

5. Create main.py

Summary: Establish a main.py script to retrieve website URLs and JSON subpart URLs from environment variables.

Description: Create a main.py script responsible for fetching website URLs and JSON subpart URLs from environment variables. This centralizes configuration management and allows for easy configuration changes without altering the code.

6. Add Unit Test Cases

Summary: Implement unit tests to validate code functionality.

Description: Develop unit test cases to verify the correctness of functions and ensure they work as expected. Unit testing helps detect regressions and provides a safety net during code changes.

7. Update README.md

Summary: Update the project's README.md file with comprehensive documentation.

Description: Improve the README.md file to include clear instructions on how to use the code, explanations of functions and their purpose, prerequisites, and any relevant information for users and developers.

Some tasks that should be implemented to enhance the library:

1. Implement Captcha Solving Function

Summary: Develop a captcha-solving function to automate the process of solving captchas encountered during web scraping.

Description: Create a Python function that utilizes captcha-solving techniques (e.g., OCR, third-party API integration) to solve captchas encountered during web scraping automatically. This function should accept a captcha image or challenge as input and return the solved captcha response.

Acceptance Criteria:
-> The function should solve various types of captchas commonly found on websites.
->It should have appropriate error handling for captcha-solving failures.
-> Unit tests should be written to ensure the function's accuracy.

Note: We can use inbuilt Python libraries to implement a captcha-solving function

2. Implement Script for Scraping Multiple JSON Subpart URLs

Summary: Develop a script that accepts multiple JSON subpart URLs for a particular website and scrapes data from each URL.

Description: Create a Python script that takes a list of JSON subpart URLs as input and scrapes data from each URL using the Playwright library. The script should handle URL validation, data extraction, and storage into a suitable format (e.g., JSON, CSV).

Acceptance Criteria:
-> The script should be able to process and scrape data from multiple JSON subpart URLs.
-> It should include error handling for invalid or inaccessible URLs.
-> The scraped data should be saved in an organized format.

3. Implement Script for Scraping Multiple Websites

Summary: Develop a script that accepts multiple website URLs and performs web scraping on each website.

Description: Create a Python script that accepts a list of website URLs as input and performs web scraping on each website using the Playwright library or another suitable web scraping tool. The script should handle URL validation, data extraction, and storage in a suitable format (e.g., a database or CSV).

Acceptance Criteria:

-> The script should be capable of scraping data from multiple websites.
-> It should include error handling for invalid or inaccessible URLs.
-> The scraped data should be saved in an organized format.

4. Implement Multithreading for Scraping

Summary: Enhance the web scraping script to use multithreading for concurrent data extraction.

Description: Modify the web scraping script developed in Ticket 2 or Ticket 3 to use multithreading for concurrent data extraction. Each thread should be responsible for scraping data from one URL or website. Proper synchronization mechanisms should be implemented to ensure thread safety.

Acceptance Criteria:
-> The script should use multithreading to concurrently scrape data from multiple URLs or websites.
-> It should handle thread synchronization to prevent data corruption or conflicts.
-> The multithreading implementation should improve the overall scraping efficiency.