Playwright Plus

Playwright Plus Summary

The provided package code utilizes Playwright, a web automation library, to intercept JSON data from web pages. It includes utility functions like json_detect_error to identify errors within the JSON response and json_parse_result to format the JSON data into a standardized response structure. The main functions, such as intercept_json_playwright and intercept_json_playwright_multiple, use Playwright to intercept JSON responses, handle errors, and parse the data. These functions offer flexibility by allowing users to provide custom error detection and parsing logic. Additionally, there's a function request_json_playwright that simplifies the process by internally utilizing intercept_json_playwright. These functions facilitate seamless interaction with web APIs, providing a structured approach to handling JSON data retrieval and potential errors during the process.

Function Encounter

- **json_detect_error**: Detects errors in the provided JSON object.
- **json_parse_result**: Parses and formats JSON data into a standardized response format.
- **intercept_json_playwright**: Intercepts JSON data from a specified URL using Playwright, handles errors, and parses the result. Supports handling captchas and multiple responses.
- **intercept_json_playwright_old**: Legacy version of intercept_json_playwright.

Function Encounter

- **open_new_page**: Create and configure a new web page within a browser using Playwright. This function simplifies the process of creating a new web page by providing default configurations and options that can be customized.
- **with_page**: A decorator function that adds a new page obtained from the _instantiate_browser_context_page function as a keyword argument to the decorated function.
- **_get_page_arg**: Retrieve a 'Page' object from function arguments or keyword arguments. This utility function is used to extract a 'Page' object from the arguments or keyword arguments of a decorated function. It is typically used within decorators in web automation or testing scripts.

Function Encounter

- **wait_after_execution**: Decorator that introduces a pause after the execution of a decorated function. This decorator is used to add a waiting period after the execution of a function. It can be helpful in web automation and testing scenarios where you need to control the timing of actions on a web page.
- **check_for_loaded_marker**: Decorator that checks for the presence and visibility of an HTML element (marker) on a web page. This decorator is used to ensure that a specific HTML element (marker) is present and visible on a web page before and after executing a decorated function. It is typically used in web automation or testing scripts.

Testing with Playwright

Test Case 1: Successful JSON Retrieval

Objective: Verify successful JSON data retrieval using Playwright.

Steps: Mock Page object, intercept JSON data, and validate 'success' key.

Expected Outcome: Assert 'success' key in intercepted JSON response to be True.

Test Case 2: Error Handling

Objective: Verify error handling when JSON retrieval fails using Playwright.

Steps: Mock Page object, intercept JSON data from an invalid URL, and validate 'error' key.

Expected Outcome: Assert 'error' key in intercepted JSON response to be 'PlaywrightInterceptError'.

Tickets Encounter For Improvement

- By implementing the Strategy Design Pattern, you can decouple the logic for error detection and JSON parsing, making the codebase more modular and easier to maintain. It also allows for easy substitution of different algorithms without modifying the existing code.

- Implement custom exception classes for every type of the PlaywrightPlus error module to provide more detailed and meaningful error handling for PlaywrightPlusException, PlaywrightInterceptError and PlaywrightInterceptJsonError.

Tickets Encounter For Improvement

- Develop a Python function designed to automatically solve captchas encountered during web scraping. The function will employ captcha-solving techniques, such as Optical Character Recognition (OCR) or integration with third-party APIs. It should be capable of accepting a captcha image or challenge as input and returning the solved captcha response.

- Develop a Python script that takes a list of website URLs as input and conducts web scraping on each website using the Playwright library or an appropriate web scraping tool. The script will encompass URL validation, data extraction, and storage in a suitable format such as a database or CSV.

Tickets Encounter For Improvement

- Develop a Python script capable of processing a collection of JSON subpart URLs provided as input. Utilize the Playwright library for extracting data from each URL. The script must incorporate functionalities such as URL validation, data extraction, and storage in an appropriate format like JSON or CSV.

Resolved Issue 1

Title: Adding Docstrings to Functions

Summary: Enhance code documentation by incorporating meaningful docstrings within each function.

Description: To elevate code readability and maintainability, it's crucial to include detailed docstrings for every function. These docstrings serve as inline documentation, describing the function's purpose, input parameters, expected output, and any exceptions it might raise. Clear and comprehensive docstrings significantly improve code understanding for developers, making maintenance and collaboration more efficient.

Resolved Issue 2

Title: Implement Custom Exceptions for PlaywrightPlus

Summary: Develop custom exception classes within the PlaywrightPlus module to enhance error handling with detailed and meaningful error messages.

Description: The task aims to implement custom exception classes in the PlaywrightPlus module to improve error handling. The custom exceptions include PlaywrightPlusException,PlaywrightInterceptError and PlaywrightInterceptJsonError.

Resolved Issue 3

Title: Refactor Error Handling and JSON Detection Logic

Summery: Refactor the error handling and JSON detection logic within the intercept_json_playwright function to ensure accurate inclusion of errors detected by json_detect_error in the target_json result data.

Description: Modify the code implementation to guarantee that json_detect_error is invoked before the assessment of target_json. By doing so, the error handling process will be streamlined, and errors identified by json_detect_error will be consistently included in the result data.

Resolved Issue 4

Title: Refactor the Code

Summary: Removed the unnecessary creation of a result object from intercepted data. Addressed the issue where calling buffer.get("error") in list format caused errors. Also, added the missing callable during the invocation of the json_parse_result function and removed unused functions for improved efficiency.

Description: During the interception of JSON data in list format, the code previously encountered errors when attempting buffer.get("error"). This problem has been resolved. Additionally, a missing callable was added when calling the json_parse_result function to ensure proper execution. Unused functions were also removed to enhance overall code efficiency and readability.

Thank You