**Introduction**

This Playwright project aims to create a scalable and maintainable test automation framework for verifying task details within Asana web application. The primary goal was to dynamically drive test scenarios using a JSON-based data structure, minimizing code duplication while ensuring adaptability for future test cases. By centralizing the data and actions the framework reduces maintenance and provides a streamlined approach to adding new tests.

**Implementation Details**

The solution leverages Playwright's end-to-end testing capabilities, combining a modular structure with data-driven techniques. Test scenarios are generated dynamically from a JSON file that contains locators, task details, and selectors. This approach ensures scalability and simplifies the process of updating or adding test cases. Actions such as typing, clicking, and assertions are abstracted into reusable helper functions to eliminate repetitive code. The modularity of the framework keeps test logic, data, and utility functions separate, ensuring maintainability and ease of future enhancements. A logging mechanism was integrated to provide detailed success messages for each task, offering better traceability during execution.

**Challenges and Solutions**

The development process encountered a few challenges. The website was displaying an error message that prevented the code from logging in after exceeding a specified number of login attempts during testing. Managing dynamic locators for multiple tasks without cluttering the test logic was addressed by centralizing all locators in a JSON file and replacing placeholders at runtime. Avoiding code duplication was another critical requirement, which was resolved by creating a helper function to handle repetitive actions like interacting with elements and performing assertions. Integrating the framework with GitHub also posed initial difficulties due to SSH authentication issues, but these were resolved by configuring SSH keys and verifying repository connections. Finally, debugging task-specific failures required enhanced logging, which was implemented to provide clear insights into task-level outcomes.

**Results**

The implemented test cases successfully verified task details, such as the presence of tasks in specific columns and the existence of associated tags. Tasks like "Draft project brief" passed validation without issues, demonstrating the framework’s ability to dynamically execute tests based on JSON data. The framework was able to adapt to different scenarios seamlessly, and no failures were detected during test runs. This confirms the effectiveness of the data-driven approach in automating test scenarios while maintaining flexibility for future enhancements.

**Recommendations**

While the framework performs well, there are opportunities for improvement. Expanding test coverage to include edge cases and error-handling scenarios would make the tests more robust. Integrating continuous integration pipelines, such as GitHub Actions, would enable automated test execution upon every code push, ensuring immediate feedback. The logging mechanism could also be enhanced with structured outputs, such as JSON-formatted logs, for easier analysis and integration with monitoring tools. As the number of test cases grows, enabling parallel execution would significantly reduce runtime. Finally, enhancing the JSON structure to accommodate conditional logic for more complex scenarios would add another layer of flexibility to the framework.