write-up for the Playwright-driven test suite project:

**Playwright UI and API Automation Suite**

**Introduction**

This task involved creating a Playwright-driven test suite for automating login functionality and verifying tasks in the Demo App. The main objective was to implement a data-driven approach to minimize code duplication and ensure scalability while adhering to industry best practices, such as the KISS (Keep It Simple and Stupid) and DRY (Don’t Repeat Yourself) principles.

**Implementation Details**

**Key Features of the Framework**

* **Technology Stack**:
  + **Playwright JS**: For browser automation and test execution.
  + **Page Object Model (POM)**: To organize locators and actions for reusability.
  + **Data-Driven Testing**: Test scenarios are dynamically generated from a JSON object to ensure flexibility.
  + **Keyword-Driven Approach**: Allows modular, maintainable, and reusable test steps.
  + **Reporting**: Allure Report for detailed test result visualization.
* **Code Principles**:
  + DRY, SOLID, and clean code practices.
  + Reusability and scalability to easily add new test cases.

**Steps to Execute**

1. **Setup Environment**:
   * Install Node.js and configure Java for Allure reporting.
   * Use Visual Studio Code (or other editors) to manage and run the project.
2. **Project Configuration**:
   * Credentials are managed in an .env.secret file.
   * Test data is stored in a JSON file for a data-driven approach.
3. **Test Execution**:
   * Headless mode: npm run test.
   * Headed mode (for debugging): npm run browser.
4. **Reporting**:
   * Automatically generate an Allure Report after execution.

**Implementation Highlights**

1. **Login Functionality**:
   * Inputs credentials (admin, password123) from environment variables.
   * Verifies successful navigation post-login.
2. **Task Verification**:
   * Dynamic navigation and task validation based on JSON-driven test data.
   * Verifies task titles, columns, and associated tags.
3. **Error Handling**:
   * Ensures graceful handling of failures with logs and screenshots for debugging.

**Challenges and Solutions**

1. **Dynamic Test Scenarios**:
   * Solution: Implemented JSON-driven test cases to easily adapt to future updates without hardcoding.
2. **Tag Verification**:
   * Challenge: Ensuring accurate tag matching.
   * Solution: Used assertions with flexible data comparison techniques.
3. **Allure Report Configuration**:
   * Challenge: Setting up Allure for detailed reporting.
   * Solution: Configured custom commands to integrate reporting seamlessly.

**Results**

* **Test Execution Summary**:
  + All test cases executed successfully in both headless and headed modes.
  + Screenshots and logs were generated for verification and debugging.
* **Pass Rate**:
  + 100% for implemented test cases.

**Recommendations**

1. **Enhanced Reporting**:
   * Include video recordings for tests in addition to screenshots.
2. **Parallel Execution**:
   * Optimize execution time by running tests across multiple browsers/devices in parallel.
3. **API Integration**:
   * Extend the suite to include API validation for end-to-end coverage.

**Pre-requirements to Execute the Automation Suite**

1. Install the latest version of Node.js.
2. Install Java and configure the JAVA\_HOME environment variable (for Allure).
3. Clone the project from GitHub and configure the .env.secret file with login credentials.

**GitHub Repository Link**

[GitHub Repository](https://chatgpt.com/c/67511e8d-83ec-800e-ae1e-24c22a1574c2)