IOV42 - SAUCEDEMO

Test Strategy & Folder Structure Document

Overview

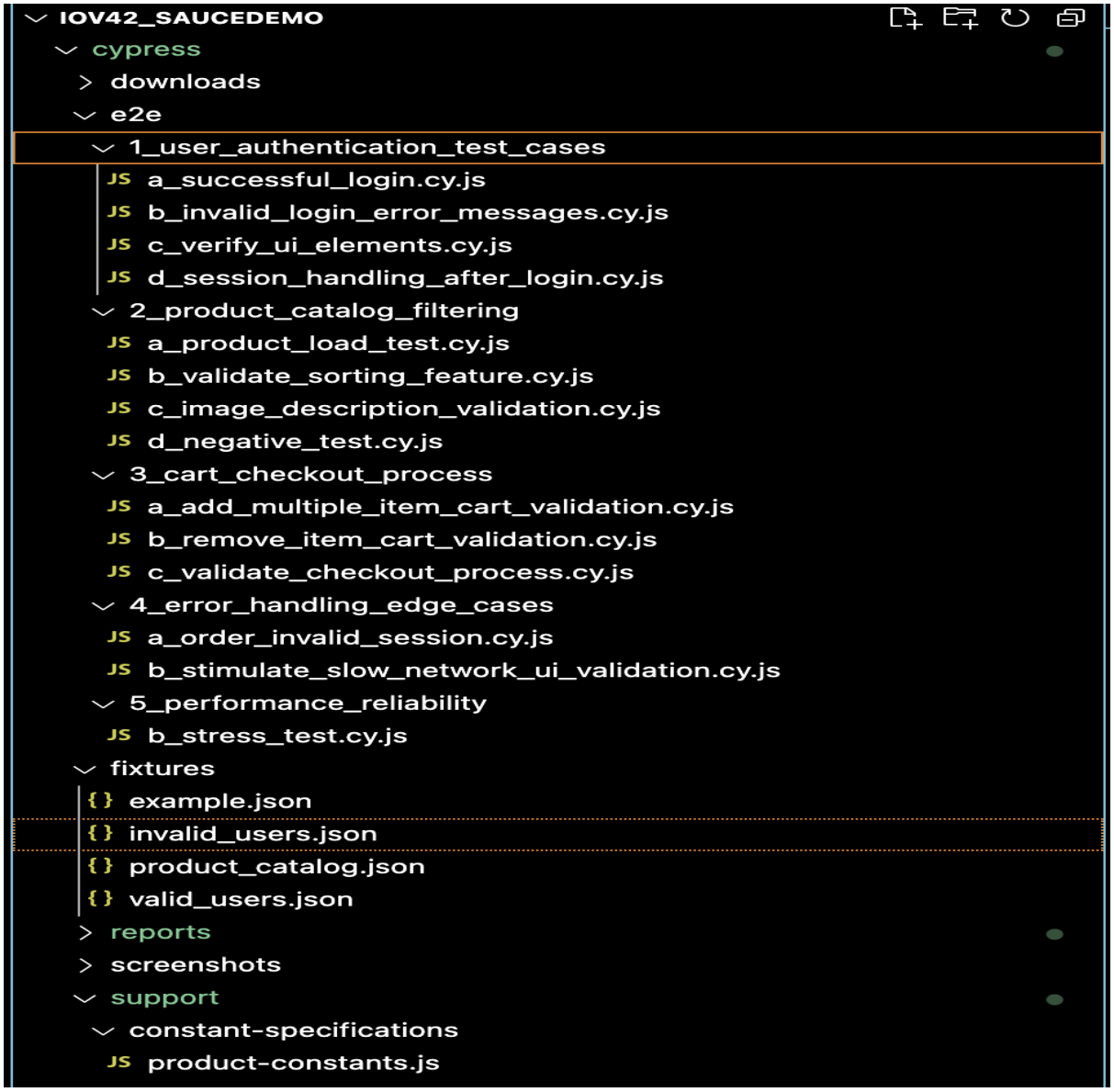
This document outlines the **test strategy** and **folder structure** for automation testing using Cypress. The goal is to ensure **modular, maintainable, scalable, and secure** test scripts while following best practices.

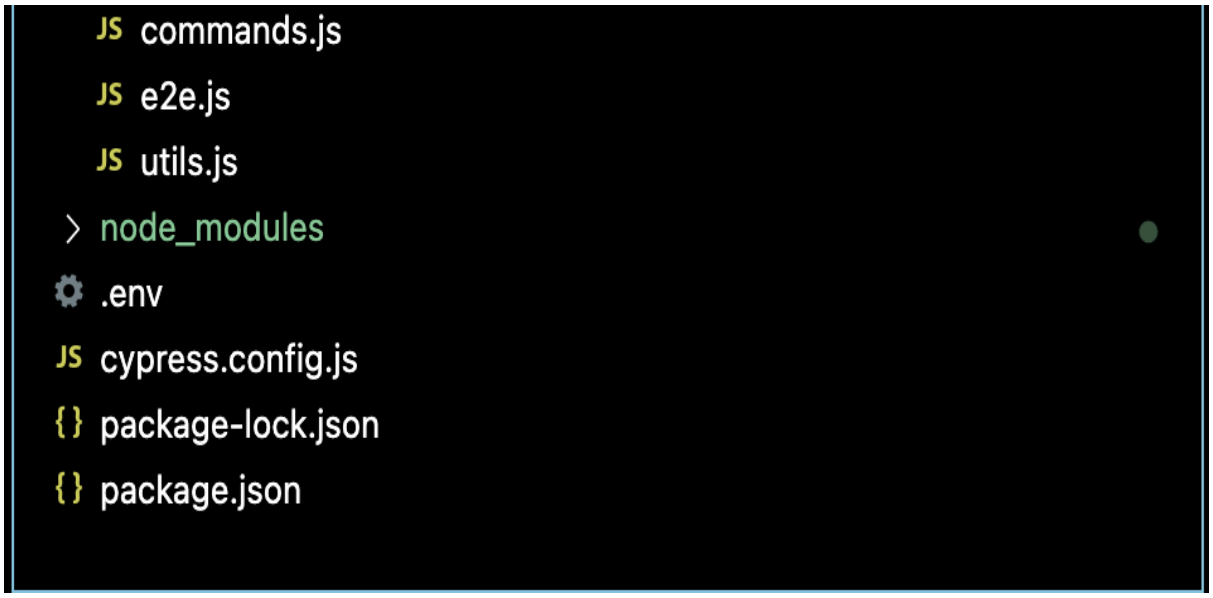
Key Objectives:

- **Maintainability** – Storing selectors, test data, and frequently changing values in dedicated files for easy updates.
- **Reusability** – Implementing reusable functions in Cypress commands and utility files to avoid redundant code.
- **Security** – Protecting sensitive data like credentials by using environment variables.
- **Reliability** – Ensuring independent test cases that do not rely on previous test results.
- **Consistency** – Following structured naming conventions and a well-organized folder structure.

2. Folder Structure

The project follows a structured **folder hierarchy** to keep test scripts, utilities, configurations, and reports well-organized.


IOV42_SAUCEDEMO
cypress
downloads
e2e
1_user_authentication_test_cases
a_successful_login.cy.js
b_invalid_login_error_messages.cy.js
c_verify_ui_elements.cy.js
d_session_handling_after_login.cy.js
2_product_catalog_filtering
a_product_load_test.cy.js
b_validate_sorting_feature.cy.js
c_image_description_validation.cy.js
d_negative_test.cy.js
3_cart_checkout_process
a_add_multiple_item_cart_validation.cy.js
b_remove_item_cart_validation.cy.js
c_validate_checkout_process.cy.js
4_error_handling_edge_cases
a_order_invalid_session.cy.js
b_stimulate_slow_network_ui_validation.cy.js
5_performance_reliability
b_stress_test.cy.js
fixtures
example.json
invalid_users.json
product_catalog.json
valid_users.json
reports
screenshots
support
constant-specifications
product-constants.js


commands.js
e2e.js
utils.js
node_modules
.env
cypress.config.js
package-lock.json
package.json

3. Selectors & Component Organization

Grouping Selectors for Easy Maintenance

- **Selectors are stored as constants** in a separate file (product_constants.js) to avoid hardcoding them inside test scripts.
- **Selectors are grouped page-wise** to improve readability and ease of modifications.
- **Global components like main navigation menu , cart and shared UI elements** are stored in global_components.js.

Benefits:

✅Easy to update selectors if UI changes.
✅Improved readability and organization.
✅Shared components reduce redundancy in test scripts.

---

4. Handling Frequently Changing Elements

To **simplify maintenance**, frequently changing elements such as **selectors, error messages, and navigation URLs** are stored in separate files.

Examples of frequently changing elements stored as constants:

- **Selectors (product_constants.js)** – Unique identifiers for UI elements.
- **Standard Error Messages**.
- **Navigation URLs (navigationUrls.js)** – Centralized control of application routes.

Benefits:

✅Changes can be made in a single place without modifying multiple test files.
✅Improves maintainability and reduces code duplication.

---

5. Reusable Functions in Cypress Commands

Implementing Cypress Commands (commands.js)

- Cypress commands are used to **avoid repetitive code** in test cases.
- Common actions such as **login, logout, and adding items to the cart** are implemented as reusable functions.

Key Features of commands.js:

✅Simplifies test scripts by abstracting common actions.
✅Increases test execution speed and reduces redundancy.
✅Ensures consistency in performing key actions.

---

6. Returning Values from Functions

Cypress commands **do not return values** by default. However, in cases where we need to return dynamically generated data (e.g., auto-generating user details for checkout), a separate utility function is used.

Implementation in utils.js:

- Utility functions **return values** that can be reused across different test cases.
- Example: A function that generates random user details (first name, last name, and ZIP code).

Benefits:

✅Allows dynamic data generation for testing.
✅Enhances flexibility in test execution.
✅Avoids hardcoded values, making tests more scalable.

---

7. Test Data Management Using Fixtures

Using JSON Files for Test Data

- Test data is stored in **fixtures** (cypress/fixtures/) as JSON files.
- This allows **separation of test data from test logic**, making it easy to update data without modifying test scripts.

Types of Test Data Stored in Fixtures:

- **Valid/invalidusers.json** – Stores valid and invalid user credentials.
- **productCatalog.json** – Contains product names, descriptions, and prices for cart validation.

Benefits:

✅Centralized test data for easy modification.
✅Reduces test maintenance effort.
✅Increases flexibility for different test scenarios.

---

8. Security Best Practices

Storing Credentials Securely

- **Usernames and passwords are stored in the .env file** instead of hardcoding them in test scripts.
- The .env file is **added to .gitignore** to prevent accidental exposure in version control.

Why use .env?

✅Enhances security by keeping credentials private.
✅Prevents unauthorized access to sensitive data.
✅Enables easy configuration for different environments (like test,production).

---

9. Cypress Configuration (cypress.config.js)

The **Cypress configuration file** (cypress.config.js) is customized to optimize test execution and reporting.

Key Configurations:

1. **E2E Settings:**
    - Base URL to avoid repeating full URLs in tests.
    - Default timeout to ensure tests wait for elements to load properly.
    - Chrome Web Security disabled for handling cross-origin requests.
    - Global retries:
        - **1 retry in GUI mode** (for debugging).
        - **2 retries in headless mode** (for stability).
2. **Reporting Configuration:**
    - Reports are stored in a **dedicated folder**.
    - Consolidated reports are named dynamically with timestamps.
    - Supports **HTML and JSON formats** for better visibility.

Benefits:

✅Improved test execution speed and reliability.
✅Detailed test reports for debugging and analysis.
✅Enhanced retry mechanism to handle flaky tests.

---

10. Ensuring Accurate Test Execution

Clearing State Before Each Test

To ensure **consistent and accurate results**, Cypress clears browser data before each test run.

Key Steps:

- **Cookies and local storage are cleared** before each test execution to avoid data persistence issues.
- **Every test starts with clear cookies, session then login, performs the action, and logs out**, keeping test cases **independent and self-contained**.

Why is this important?

✅Prevents data from previous tests affecting the current test.
✅Ensures each test runs in a clean environment.
✅Reduces test flakiness and improves accuracy.

---

11. Independent Test Cases for Reliability

Avoiding Dependencies Between Tests

- Each test case **runs independently** to avoid reliance on previous test results.
- Tests always **start with clear C&C , login, execute the test, and end with logout**.
- This ensures **consistent and repeatable** test execution across multiple runs.

Benefits:

✅Tests can be executed in parallel without dependencies.
✅Reduces false positives due to session conflicts.
✅Ensures stability across different test environments.

---

12. Dependency Management

Managing Packages in `package.json`

- All required dependencies (Cypress, test reporters, plugins) are listed in package.json.
- Running npm install ensures all dependencies are installed correctly before running tests.
- Regular updates are performed to keep the framework stable and compatible with new versions of Cypress.

Benefits:

✅Ensures a controlled test environment with the correct package versions.
✅Reduces the risk of compatibility issues between different dependencies.
✅Simplifies project setup for new team members.

---

13. Naming Conventions

- **Folders & files:** Use **kebab-case** (e.g., checkout_tests/, product_constants.js).
- **Constants & functions:** Follow **camelCase** (e.g., standardMessages, generateRandomUser).
- This ensures **consistency and readability** across the project.

Conclusion

By following this **structured approach**, I ensured **scalability, maintainability, and security** in Cypress automation. The **modular folder structure, reusable functions, secured credentials, and independent test cases** contribute to a **robust and reliable** test suite.