Customized Automation Framework

Project Structure:

Present a folder structure that separates test scripts, page objects, configurations, and utilities.

Emphasize the modular and organized approach.

Data Input Sheet Folder:

Purpose: Contains input data sheets or files used as test data inputs.

Example: Excel spreadsheets or CSV files with test data, parameter values, or configurations.

Drivers Folder:

Purpose: Houses the browser drivers required for Selenium WebDriver to interact with different web browsers.

Example: Chrome Driver, Gecko Driver, WebDriver executables.

Excel TXT File Folder:

Purpose: Stores final output in Excel or log file.

Page Object Model (POM) Folder:

Locator Folder:

Purpose: Contains locators used to identify web elements in the application.

Example: XPath, CSS selectors, IDs, or names of HTML elements.

Pages Folder:

Purpose: Organizes page-specific classes that encapsulate interactions with specific pages of the application.

Example: Login Page, CMD Page, Account Page classes.

Test Folder:

Purpose: Contains test scripts that utilize the Page Objects and interact with web elements.

Example: Test classes with methods for different test scenarios.

Screenshot Folder:

Purpose: Stores screenshots taken during test execution for capturing and diagnosing issues.

Example: Screenshots of application states, errors, or failures encountered during testing.

Test Cases Backup Folder:

Purpose: Serves as a backup for storing test cases and their details, often in a structured format.

Example: Excel sheets containing test case steps, expected outcomes, and actual results.

This framework structure aims to promote a modular and organized approach to test automation. It separates different types of resources, such as test scripts, test data, screenshots, and drivers, into distinct folders, enhancing maintainability, readability, and scalability of your test suite. Using the Page Object Model (POM) design pattern further improves code reusability and maintainability by isolating page interactions from test scripts. Additionally, the inclusion of a backup folder for test cases helps in maintaining a historical record of test executions and results.

Benefits of Automating T24 Modules with Selenium Python and TDD Framework:

Efficiency: Automated tests run faster and can be executed more frequently, providing rapid feedback on changes made to the application.

Accuracy: Automated tests eliminate the risk of human errors and ensure consistent test execution, leading to more reliable results.

Coverage: Automated tests can cover a wide range of scenarios and functionalities, ensuring thorough testing of T24 modules.

Regression Testing: Automated tests quickly identify regression issues caused by new code changes, helping to maintain existing functionality.

Parallel Execution: Automation allows running tests in parallel, reducing test execution time and accelerating the testing process.

Documentation: Automated test scripts serve as living documentation of system behavior and expected outcomes.

Scalability: The customized framework can be extended to accommodate new modules and features as the application evolves.

Maintenance: A well-structured framework makes it easier to maintain and update tests when application changes occur.

Consistency: A customized framework enforces consistent coding practices and standardization across automated tests.

Cost Savings: Automated testing reduces manual effort, leading to cost savings in the long term.