**Technical Guidelines for Designing Test Automation Framework**

**Set Up/Accessibility**

* It should be simple and easy for installation/configuration
* Should have minimal or no dependency on external system configurations (Platform, Browsers, files etc.)
* All the dependencies should be the part of pom.xml (Java), package. Json (Node.js) or requirements.txt (Python). There should not be any requirement of installing any dependency externally.
* Selenium Web driver dependencies for browsers should be downloaded on the fly at run time through Web driver Manager in case of Java and Python based framework. Modern Java script frameworks are already bundled with browser dependencies, and we need to make use of it effectively.

**Logs**

* It should have detailed information of each native web event(Click,Select,Check etc.)
* Test Suite Information before and after execution must be logged with details of failure, kind of short summary for test suite.
* Test Events must be logged for below actions
  + On start
  + On completion
  + On Failure
* Test Execution Summary with total no. of test suites with small description must be logged. Nowadays, it's easily available in all loggers.
* For parallel runs, separate log files must be created for each thread/run.
* For every new run, create or override logs in text file.

**Html Reports**

* Should be easily configurable
* Supporting multiple html reports with charts, details of tests, screenshots on failure, execution time for suite and test.
* Extent Reports, Allure Reporting etc. must be incorporated
* All events from logs must be captured inside html report.

**Retry Mechanism for Flaky tests/objects**

* There must be a retrying mechanism for false failures due to application performance or any intermittent issues. This includes re-running failed tests after running complete suite or re-retrying steps failed at element.
* Retry count should be configurable

**Provision for skipping and running selected tests**

* Framework must allow user to run selected test from entire test suite or skip it as per choice.

**Test Execution Output**

* As a part of test execution output, below items must be collected for each test or only for Failed test for easy debugging. This will allow us to quickly report defects or fix the script.
* Har file with network logs
* Video recording
* Screenshots of failed tests
* Network tracing file showing performance of Api's during test flow.

**Tests**

* Use of page objects. For maximum re-use, define page objects as global objects and use them anywhere in test or across any page objects.
* All the test validations must be called inside test only instead of asserting in page object methods.
* Test dependency must be organized with the help of custom flags or custom fixtures.
* Use test data stored in files dynamically, don't hardcode any value in test.

**Page Objects**

* Each page should have its own page object.
* All the locators on the page must be defined as variables with value as locator inside page object
* Locators must be designed using **CSS** only for best and fast results, go for XPath or another locator only if CSS can't be built.
* Avoid defining elements with duplicate or multiple locators.
* Try to identify locator uniquely rather than using element text or contains methods
* Elements having common properties on different pages must be grouped inside common page objects.
* Make use of common methods for designing page objects methods simply by-passing locator.

**Common Web Element Methods**

* The native events on web elements must be represented by a wrapper method on top of selenium's method and it should be called inside page object methods.
* This includes
  + Click
  + Dropdown, Multiselect dropdown, Dropdown search, List box
  + Button/Radio button
  + Textbox,TextArea box, Slider Textbox
  + fetching multiple or single element
  + fetching text from multiple or single element
  + fetching element attributes from multiple or single element.
* While designing the above methods, make sure to do all the element handling like
  + Wait for element to be visible/clickable
  + Scroll into view so that element is properly interactable
  + Don’t make any exception from webdriver as silent, please throw it and handle in page object method if required and log the events
* We can have a common method for element handling that will be called from web native methods.
* Page Operations like switching to page frames, handling spinners, waiting for page to load with all elements can be separately defined.
* Auto-Waiting -All the wrapper function should make use of this function to avoid repeatedly calling the same methods.

**Test Data**

* Test data must be handled separately for test and environment
* Good to use json or yaml, as excel is limited only for windows creating platform dependency.
* Every test suite can have a separate test data file
* Credentials must be encrypted.

**Helpers**

* Common Methods for designing page object methods.
* For Managing web drivers
* For reading data from database or files
* For managing third party integrations like slack, email
* For executing Api
* Filtering html/xml results files etc.
* Custom Assertions (soft and hard)
* loggers

**Test Runners**

* It should have all the tests and environment configuration.
* Only mention test suite list and environment details for test execution, other things must be pre-configured like browser details.
* Support for cross browser run, parallel run must be configurable.
* Should have Easy for Debugging and fast in execution
* Test runners must be configurable as per test types like Sanity, Regression, smoke etc. The grouping of tests can be done at test level. TestNG, Jest, Pytest already has support for this.
* Test suites can be mapped with unique names and same can be called on command line for running.

**Integration**

* CICD integration; Jenkins runs should be smooth and have the same stability what we see on local runs.
* Jenkins Jobs/pipeline must be parameterized.
* For Sharing test execution reports there must be integration with slack, email and other utilities.
* Good to have multiple integrations like Azure DevOps, Sauce labs etc.

**UI Automation Tools/libraries available in market that one should be aware of:**

* Selenium Web driver (Java and Python among most popular bindings)
* Protractor (Deprecating in 2022)
* Cypress
* Puppeteer
* Playwright

**API Automation libraries available in market and used widely**

* Rest Assured with Java and test-ng
* Request module in python with pytest
* Super Jest with JavaScript and jasmine
* Request module in node.js with jasmine