**Playwright Test Script Authoring**

This document provides guidelines for authoring Playwright test scripts in a consistent and maintainable manner. By following these guidelines, your test scripts will be more readable, easier to maintain, and better structured.

**File Naming Convention**

* Test files should be named using the following convention: **featureNameTest.ts**. The **featureName** should be descriptive and use camelCase (words without spaces, with each word's first letter capitalized except the first word). The `**test.ts`** suffix indicates that it is a test file for a TypeScript project.

Example:

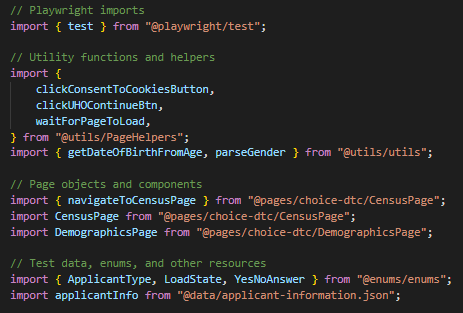
* For a login feature: **loginTest.test.ts**
* For a user registration feature: **userRegistrationTest.test.ts**
* For a shopping cart feature: **shoppingCartTest.test.ts**

**Organizing and Grouping Imports**

Organize and group imports according to their origin and purpose. In general, follow these guidelines:

* First, import packages and modules from the Playwright library.
* Second, import utility functions and helpers.
* Third, import page objects and custom components.
* Finally, import test data, enums, and other resources.

Example:

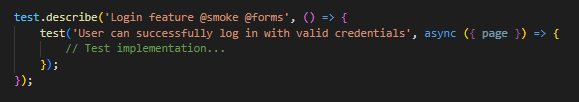


**Test Descriptions and Tags**

* **Description:** Use clear and concise test descriptions that explain the purpose of the test. The description should be easy to understand and communicate the intent of the test.
* **Tags:** Add relevant tags to your test using the **@tag** notation to help with filtering and organization. Tags can be used to group related tests, mark tests as smoke tests, or label tests by functionality.

Some common tags include **@smoke, @e2e, @regression**, or feature-specific tags like **@STM, @aca, @dental**.

Example:

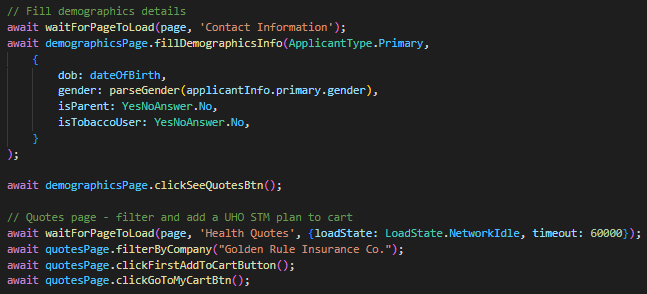


For reference, use the **@tag** provided in the **Test Tags Categories and Descriptions** documentation.

**Guidelines for Writing Test Steps within the `test()` Function**

* Start by initializing page object instances and defining any test data or variables needed for the test.
* Perform any necessary setup steps, such as navigation or cookie consent.
* Interact with the web application using the page objects, including filling out forms, clicking buttons, or making selections.
* Add comments when necessary to provide a brief overview of the action being performed or to clarify complex logic.
* Use the **`await`** keyword to ensure that each action is completed before moving on to the next step.
* Implement checks for page loads or specific elements to be visible before interacting with them.

Example:



Following these guidelines will help create well-organized, maintainable, and consistent Playwright test scripts.