Playwright Framework

Prerequisites

Before you begin, make sure the following tools are installed:

- **Node.js** (v16 or later) – Download here
- **npm** (comes with Node.js)
- **Visual Studio Code (VS Code)** – Download here

Step 1: Create a New Project & Install Dependencies

1.1 Create Project Folder and Open it in VSCode editor

1.2 Install Playwright with TypeScript

Run the below command to initialize a Playwright project:

```
npm init playwright@latest
```

Choose the following options when prompted:

- Language: **TypeScript**
- Test directory: **tests**
- GitHub Actions workflow: **Yes or No** (optional)
- Install browsers: **Yes**

1.3 Install Additional Packages

```
npm install csv-parse
npm install xlsx
npm install -D allure-playwright
npm install @faker-js/faker
```

Step 2: Create Project Folder Structure

Organize your project like this:

```
PlaywrightFramework/
│
├── tests/                  --> All test files
├── pages/                  --> Page Object Model classes
├── utils/                  --> Utility functions
├── data/                   --> Test data files (JSON, CSV)
├── reports/                --> Generated test reports
├── test.config.ts          --> Test configuration values
├── playwright.config.ts    --> Playwright main configuration
└── package.json
```

Step 3: Update playwright.config.ts

Here's a sample configuration:

```
import { defineConfig, devices } from '@playwright/test';

export default defineConfig({
  timeout: 30 * 1000,   //30000 ms(30 secs)
  testDir: './tests',
  fullyParallel: false,
  //retries: process.env.CI ? 2 : 0,
  retries:1,
  //workers: process.env.CI ? 1 : undefined,
  workers: 1,

  reporter: [
    ['html'],
    ['allure-playwright'],
    ['dot'],
    ['list']
  ],

  use: {
    trace: 'on-first-retry',
    screenshot: 'only-on-failure',
    video: 'retain-on-failure',
    //headless: false,
    viewport: { width: 1280, height: 720 }, // Set default viewport size for consistency
    ignoreHTTPSErrors: true, // Ignore SSL errors if necessary
    permissions: ['geolocation'], // Set necessary permissions for geolocation-based tests
  },

  //grep: /@master/,

  projects: [
   {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] },
    },
    /*{
      name: 'firefox',
      use: { ...devices['Desktop Firefox'] },
    },

    {
      name: 'webkit',
      use: { ...devices['Desktop Safari'] },
    } */
  ],

});
```

Step 4: Create test.config.ts for Global Test Data

```
export class TestConfig{

  appUrl="http://localhost/opencart/upload/"

  //valid login credentials- create your own login account
  email="pavanol@abc.com"
  password="test@123"

  //product details
  productName="MacBook"
  productQuantity="2"
  totalPrice="$1,204.00"
}
```

Step 5: Create Page Object Model (POM) Classes

Create a separate .ts file under the pages/ folder for each page:

- HomePage.ts
- RegistrationPage.ts
- LoginPage.ts
- LogoutPage.ts
- MyAccountPage.ts
- ProductPage.ts
- SearchResultsPage.ts
- ShoppingCartPage.ts
- CheckoutPage.ts

Each class should contain methods for interacting with UI elements of that page.

Step 6: Add Test Data Files

Place test data files inside the data/ folder:

- logindata.json
- logindata.csv

Step 7: Create Utility Files

Inside utils/ folder:

- dataProvider.ts – Read JSON and CSV data
- randomDataGenerator.ts – Generate dummy data using faker

Step 8: Write Test Cases

Store your test cases under the tests/ folder. Example tests:

- AccountRegistration.spec.ts
- Login.spec.ts
- LoginDDT.spec.ts
- Logout.spec.ts
- SearchProduct.spec.ts
- AddToCart.spec.ts
- EndToEndTest.spec.ts

Step 9: Use Tags for Test Grouping

Tag your tests for easy execution:

```
test('@master', async ({ page }) => { ... });
test('@sanity', async ({ page }) => { ... });
test('@regression', async ({ page }) => { ... });
test('@datadriven', async ({ page }) => { ... });
```

Step 10: Generate Reports (HTML & Allure)

Already included in playwright.config.ts:

```
reporter: [
  ['html',],
  ['allure-playwright']
]
```

Run the command to generate and view allure reports.

```
allure generate ./allure-results -o ./allure-report --clean
allure open ./allure-report
```

Step 11: Parallel or Serial Execution

For parallel test runs, add this to your config:

```
export default defineConfig({
  workers: 4,
  fullyParallel: true,
});
```

Step 12: Add Scripts to package.json

Add custom commands under "scripts" section:

```
"scripts": {
  "test:master": "playwright test --grep @master",
  "test:sanity": "playwright test --grep @sanity",
  "test:regression": "playwright test --grep @regression",
  "test:datadriven": "playwright test --grep @datadriven",
  "test:master:headed": "playwright test --grep @master --headed",
  "test:sanity:debug": "playwright test --grep @sanity --debug"
}
```

To Run:

```
npm run test:master
```

Step 13: Commit the project files to Git and Push them to Github.

Step 14: Playwright Integration with Jenkins.

- Run Local Playwright project in Jenkins
- Run GitHub Playwright project in Jenkins
- Run GitHub Playwright project with Jenkins Pipeline