Playwright Framework Setup Guide

# **1. Installation and Setup**

1. Install Dependencies:  
- Ensure Node.js and npm are installed on your machine.  
- Run the following command to install the necessary packages:  
```bash  
npm install  
```

2. Configuration:  
- The `playwright.config.ts` file handles environment-specific configurations.  
- It reads from environment files (`.env.qa`, `.env.staging`) based on the `ENVIRONMENT` variable.  
- To run tests with specific settings, set the `ENVIRONMENT` variable before execution:  
```bash  
ENVIRONMENT=qa npx playwright test  
```

3. Directory Structure:  
- \*\*`e2e/tests/ui`:\*\* Contains UI tests.  
- \*\*`e2e/tests/api`:\*\* Contains API tests.  
- \*\*`e2e/libs/custom-decorators`:\*\* Contains custom decorators.  
- \*\*`scripts`:\*\* Contains scripts for generating HTML reports.

# **2. Custom Decorators**

The custom `@step` decorator is designed to log test steps in a readable format within your Playwright tests. It wraps test methods to provide detailed step-level logging using Playwright's built-in `test.step` functionality.

Usage:  
To use the custom `@step` decorator, simply import it and apply it to any method within your test:  
```typescript  
import step from '../libs/custom-decorators/step.decorator';  
class ExamplePage {  
 @step('Log in to the application')  
 async login(username: string, password: string) {  
 await this.page.fill('#username', username);  
 await this.page.fill('#password', password);  
 await this.page.click('#login');  
 }  
}  
```

This will output the step in the test report, making it easier to trace actions within your tests.

# **3. HTML Reporting**

The framework includes scripts to generate HTML reports that visualize test data. The `html-generator.js` script processes data from an Excel file (`Test\_Data.xlsx`) and generates an HTML report.

- \*\*Chart.js Integration:\*\* The report uses Chart.js to visualize data, including test functionality distribution, sprint data, and more.

- \*\*Customization:\*\* You can customize the charts and data processing within `html-generator.js` to suit your reporting needs.

Generating the Report:  
To generate the HTML report, simply run the script:  
```bash  
node scripts/html-generator.js  
```  
This will create an HTML file (`test\_report.html`) in your project directory, which you can open in a browser to view the results.

# **4. Running Tests**

Command-Line Options:  
- \*\*Running All Tests:\*\*  
```bash  
npx playwright test  
```

- \*\*Running Specific Tests:\*\*  
Use `--grep` or `--grep-invert` to run specific tests or exclude tests based on titles:  
```bash  
npx playwright test --grep @smoke  
```

- \*\*Headless Mode:\*\*  
Although the config specifies `headless: false` for Chromium, you can override this in the command line:  
```bash  
npx playwright test --headed  
```

Parallel Execution:  
- The framework is configured for parallel execution (`fullyParallel: true`), which speeds up test runs. You can control the number of workers with the `--workers` option:  
```bash  
npx playwright test --workers=4  
```

# **5. CI/CD Integration**

- \*\*GitHub Actions:\*\* A GitHub Actions workflow file (`.github/workflows/playwright.yml`) is included to integrate your tests into a CI/CD pipeline.

- This workflow will trigger on pushes to the main branch, running your Playwright tests automatically.

# **6. Extending the Framework**

- \*\*Custom Fixtures:\*\* The framework includes a `customFixtures.ts` file, allowing you to define and register custom fixtures for your tests. This is useful for setting up shared state or resources.

- \*\*Page Object Model (POM):\*\* The `e2e/pages` directory contains Page Object classes (e.g., `loginPage.ts`, `cartPage.ts`), promoting the reuse of code and better test maintenance.

# **7. Best Practices**

- \*\*Test Data Management:\*\* Store test data separately in `e2e/testdata`, and utilize data-driven testing by loading data from files or external sources.

- \*\*Error Handling and Debugging:\*\* Utilize Playwright’s `trace` feature (enabled in your config) for debugging. It provides detailed information on failed tests, including screenshots, logs, and network requests:  
```bash  
npx playwright show-trace <trace.zip>  
```

- \*\*Test Reporting:\*\* While the framework uses an HTML reporter, you can integrate with other reporters (e.g., Allure) if additional features like categorization and history tracking are needed.

# **8. Future Enhancements**

- \*\*Visual Testing:\*\* Implement visual regression tests in the `visual-tests` directory for UI consistency across versions.

- \*\*API Testing:\*\* Expand the `api` tests to cover more extensive scenarios, leveraging Playwright’s robust API testing capabilities.

# **9. Linting and Code Quality**

Your Playwright framework is configured with ESLint and Prettier to maintain code quality and consistency.

1. \*\*ESLint:\*\*

- \*\*Linting Command:\*\* Run `npm run lint` to check for code issues based on the ESLint configuration.

- \*\*Auto-Fix:\*\* Run `npm run lint:fix` to automatically fix issues that can be resolved programmatically.

2. \*\*Playwright ESLint Plugin:\*\*

- The framework includes a Playwright ESLint plugin that enforces specific rules:

- \*\*Await for Assertions:\*\* Ensures that all `expect` conditions are awaited, which is crucial for reliable test execution.

- \*\*Mandatory Assertions:\*\* Checks that each test includes at least one assertion to validate test outcomes effectively.

3. \*\*Prettier:\*\* Prettier is integrated for consistent code formatting across the project, making the codebase clean and readable.

4. \*\*Git Pre-Commit Hook:\*\*

- \*\*Mandatory Checks:\*\* The linting and formatting checks are enforced through a Git pre-commit hook using Husky. This means that any code that doesn’t pass the linting or formatting rules will be flagged before it can be committed, ensuring that the codebase remains clean and consistent.

- \*\*Setup:\*\* The pre-commit hook is configured in the `.husky/pre-commit` file and automatically runs the linting and formatting checks before each commit.