Prompt Handbook for Playwright Test & Page Class Generation

This handbook provides a curated set of LLM-optimized prompt templates designed to generate robust, best-practice-compliant Playwright tests and Page Object classes. Use these templates with tools like ChatGPT or other code-generation models to accelerate test development.

---

1. Page Object Model (POM) Class Generation

Prompt:

Generate a Playwright Page Object Model (POM) class in TypeScript for the [PageName] page of a web application. Follow best practices for maintainability and reusability. The page contains the following elements and actions:

- A login form with `username`, `password`, and `login` button
- Validation message shown on login failure
- Navigation bar with links to Dashboard, Settings, and Logout

Ensure the class:

- Uses proper selectors (e.g., `data-testid` or accessible roles)
- Has async methods like `login(username: string, password: string)`
- Initializes elements in the constructor using Playwright's `Locator`
- Avoids hardcoding wait logic; use `expect()` or built-in waits

Return only the complete TypeScript class code.

---

2. E2E Test File Creation

Prompt:

Create an end-to-end test file using Playwright Test (TypeScript) for verifying login functionality. The test should:

- Use the Page Object class for the Login page
- Validate successful login redirection to the Dashboard page
- Include negative tests for invalid credentials
- Follow BDD-style naming (`test('should login successfully', async () => {})`)
- Use `test.describe` and `test.beforeEach` for setup
- Include necessary imports and a sample data block

Only include the final test file. Follow best practices for readability and structure.

---

3. Component-Specific Page Class (Search Component)

Prompt:

Write a Playwright Page Object class for a Search component that:

- Accepts user input
- Shows suggestions in a dropdown list
- Allows keyboard navigation

- Supports selection of suggestions

Ensure:

- All locators use `getByRole` or `getByTestId`
- Includes methods like `search(term: string)` and `selectSuggestion(index: number)`
- Incorporates accessibility best practices

Return a full TypeScript page class using Playwright.

---

4. Data-Driven Test Generation

Prompt:

Generate a Playwright test file in TypeScript that performs **data-driven testing** for a registration form. Test cases should vary:

- Email formats (valid, invalid)
- Password policies (length, characters)
- Required fields

Requirements:

- Use `test.each` or similar construct for looping test data
- Assert validation messages or successful submission
- Import data from a JSON or array within the test

Include full test code with comments.

---

5. Table Component Test

Prompt:

Generate a Playwright test in TypeScript that validates a React-based table component. The test should:

- Verify column headers and row data
- Filter data using a search box
- Sort by columns (ascending/descending)
- Paginate through results

Requirements:

- Use assertions to validate visible data
- Write modular helper functions if needed
- Use Page Object pattern if applicable

Only return the final test code.

---

6. Cross-Browser Testing Support

Prompt:

Write a Playwright test in TypeScript to verify login functionality across Chrome, Firefox, and WebKit

using the `--project` config in `playwright.config.ts`.

- Use a LoginPage object
- Print the browser name in logs
- Include a matrix-based test setup using `test.use({ browserName: '...' })`

Include config snippet and test file.

---

7. Visual Regression Test

Prompt:

Generate a Playwright visual regression test using `expect(page).toHaveScreenshot()` to validate the layout of the homepage.

- Use `test.skip` for Firefox if rendering differs
- Define baseline screenshots per browser
- Use `fullPage: true` for complete viewport capture

Return a test file with annotations and screenshot options.

Feel free to customize these prompts for specific project workflows, UI components, or testing goals.

For more advanced templates or generation strategies, consider integrating these prompts into your

IDE or CI pipelines using OpenAI API, or other AI platforms.