Playwright Test Framework Generator

User Manual

Table of Contents

Prerequisites

Before using the generator or the generated framework, you need:

- **Node.js** (16+; LTS recommended)
- **npm** (comes with Node)
- **Git** (recommended)
- **A modern browser** (Chrome/Edge/Firefox)

Quick Commands (PowerShell / pwsh)

```
node --version
npm --version
git --version
```

How to Use the Generator

Follow this basic flow to generate your Playwright test framework:

1. **1** **Open the app** in your browser (local dev: `npm run dev`) or use the deployed UI
2. **2** **Select a programming language** (TypeScript or JavaScript)
3. **3** **Pick the browsers** you want (Chromium, Firefox, Safari)
4. **4** **Enable testing capabilities** you need (UI, API, visual, accessibility, performance)
5. **5** **Toggle integrations** (GitHub Actions, Allure, Cucumber, Faker, JUnit) as required
6. **6** **Configure fixtures**, environment variables, Docker, and code-quality preferences
7. **7** **Use the PreviewPanel** to inspect the project structure and samples
8. **8** **Click Generate** to download a ZIP with the ready-to-run framework

Features

What each feature does, why use it, and how to use it:

SelectProgrammingLanguage

What it is: Choose TypeScript (type-safety) or JavaScript (simpler).

Why use it: TypeScript helps catch bugs earlier and provides editor assistance; JavaScript is easier if you don't want types.

How to use: Click the language radio button. The generated files and examples will match the selection.

> 💡 If you plan team maintenance or large projects, prefer TypeScript.

ConfigureBrowserSettings

What it is: Choose which browsers your tests will run against (Chromium, Firefox, WebKit/Safari).

Why use it: Real cross-browser confidence — make sure your app works in all target browsers.

How to use: Tick the boxes for the browsers you need. The generator will add them to `playwright.config.*`.

> ✅ Generated `playwright.config.*` will list projects for each selected browser.

ConfigureTestingCapabilities

What it is: Enable UI testing, API testing, visual, accessibility, and performance testing examples.

Why use it: Only generate the parts you need — keeps the project minimal and relevant.

How to use: Turn on the checkboxes for the capabilities you want. The ZIP will include sample tests and utilities for each chosen capability.

> 📝 If you enable API testing, a tests/api folder with an example request test will be created.

ConfigureIntegrations

What it is: Toggle integrations such as GitHub Actions CI, Allure reporter, Cucumber, Faker, and JUnit output.

Why use it: Integrations let you run and report tests in CI or use BDD formats and test data helpers.

How to use: Enable an integration and (when present) fill its simple settings. The generator will include the necessary dependencies and config (e.g., GitHub workflow files).

> 💡 If you enable Allure, you'll need to install the Allure CLI locally or in CI to view reports.

ConfigureTestFixtures

What it is: Choose fixture patterns (shared authenticated sessions, test data setup/teardown, page objects).

Why use it: Fixtures make tests consistent and reduce repetition (e.g., login once per test, consistent DB setup).

How to use: Pick patterns or enable page-object scaffolding. The generator will emit fixture files under `tests/fixtures`.

> ✅ Check `tests/fixtures/*` in the generated project.

ConfigureEnvironmentSettings

What it is: Set environment-specific values (base URLs, API endpoints, credentials) and create multiple environment examples.

Why use it: Run the same tests against dev, staging, prod by switching env files.

How to use: Add base URL and API URL for each environment. ZIP includes `.env.example` and per-env files if specified.

> 📋 **Usage:** Copy `.env.example` to `.env` and edit values before running tests.

ConfigureCodeQuality

What it is: Options for ESLint, Prettier, and TypeScript compiler settings.

Why use it: Keeps test code clean and consistent across teams.

How to use: Enable defaults or custom presets; the generator will include config files if requested.

> 💡 Run linter/formatter locally before committing for consistent style.

ConfigureCIPipeline

What it is: Generates GitHub Actions workflows to run tests and store artifacts.

Why use it: Run tests automatically on PRs and pushes.

How to use: Enable CI in the UI; the generator will create `.github/workflows/playwright.yml` (and optional PR/nightly workflows).

> ✅ The generated `.github/workflows` folder contains the YAML workflows.

ConfigureDockerization

What it is: Add Dockerfile and docker-compose helpers to run tests inside a container.

Why use it: Reproducible environments and easier CI container runs.

How to use: Enable Docker; the generator will add Dockerfile, `.dockerignore`, and compose files.

How to run:

```
docker build -t playwright-tests .
docker run --rm playwright-tests
```

> 💡 If on Apple Silicon, prefer multi-arch images or run with `--platform linux/amd64`.

PreviewPanel

What it is: A live preview of the generated project structure and sample files before downloading.

Why use it: Inspect what will be created so you can adjust options before generating.

How to use: Click the Preview tab/section in the UI to browse files that will be in the ZIP.

GenerateButton

What it is: Produces the ZIP with all selected configuration, example tests, fixtures, CI files, and docs.

How to use: After configuring options, click Generate → download ZIP.

After download: unzip, install deps, install Playwright browsers, then run tests (commands below).

ExampleShowcase / Example Test Files

What it is: Realistic example tests (UI Page Object examples, API client, utilities).

Why use it: Shows you how to write tests with the chosen framework choices.

How to use: Open the tests folder in the generated project to read and adapt examples.

What the Generator Produces

Summary of files created by the generator:

- `package.json` with dependencies (Playwright, plus integration deps if enabled)
- `playwright.config.js/ts` configured with selected browsers and reporters
- `tests/...` example test suites (UI, API, visual, a11y, performance if enabled)
- `tests/pages/*` page objects and components
- `tests/fixtures/*` fixtures for auth/data setup
- `.env.example` and per-environment example files
- `.github/workflows/*` if CI enabled
- `Dockerfile`, `docker/` files if Docker enabled
- `README.md` with basic run/troubleshooting notes

(These behaviors come from generateFramework.ts — see generatePackageJson, generatePlaywrightConfig, and generateTestExamples functions.)

How to Run the Generated Framework

Basic steps to get your generated framework up and running:

1. **Unzip** the generated archive and open the project folder
2. **Install dependencies:**

```
npm install
```

3. **Install Playwright browsers:**

```
npx playwright install
```

4. **If running in CI or headless environment, also run deps installer:**

```
npx playwright install-deps
```

5. **Copy and edit environment file:**

```
cp .env.example .env  # or copy manually on Windows: copy .env.example .env
# Edit .env with correct BASE_URL / API_BASE_URL etc.
```

6. **Run tests:**

```
npm test
# or run UI debug UI:
npm run test:ui
```

Docker: Build + Run

```
docker build -t playwright-tests .
docker run --rm playwright-tests
```

Allure Reports (if enabled)

1. **Install Allure CLI** (globally or in CI):

```
npm install -g allure-commandline
```

2. **After tests produce results, generate the report:**

```
npm run test:report  # if generator includes this script
allure serve <results-directory>
```

Simple Verification & Troubleshooting

Common Issues & Solutions

- **If tests fail due to missing env values:** ensure `.env` has correct BASE_URL, API_BASE_URL, test user credentials
- **If Playwright complains about missing browsers:** run `npx playwright install`
- **Lint/type issues:** run `npm run type-check` or `npm run lint` (if included)
- **CI workflow doesn't run:** ensure you committed `.github/workflows/*` and pushed to GitHub
- **Docker build fails on Apple Silicon:** use multi-arch base image or `--platform linux/amd64`

Generated by Playwright Test Framework Generator