Playwright Setup (VS Code + TypeScript)
Framework & Concepts Guide By Shammi Jha

A practical, end-to-end reference for building a scalable Playwright automation framework in TypeScript.

Quickstart (5–10 minutes)

Goal	Commands / Steps
Install prerequisites	Install Node.js LTS, Git, VS Code
Create project	mkdir pw-ts && cd pw-ts npm init -y
Install Playwright	npm i -D @playwright/test npx playwright install
Add TS + config	npm i -D typescript ts-node @types/node Create tsconfig.json + playwright.config.ts
Run tests	npx playwright test npx playwright show-report

1. Prerequisites

1.1 Tools you need

- Node.js (LTS recommended) and npm (comes with Node).
- VS Code (recommended editor for Playwright TypeScript).
- Git (recommended for source control).
- Chromium/Firefox/WebKit will be installed by Playwright.

1.2 Recommended VS Code extensions

- Playwright Test for VS Code (test explorer, codegen, trace viewer integration).
- ESLint (linting) and Prettier (formatting).
- GitLens (optional).

1.3 Folder location best practices

- Keep the project in a short path (avoid deeply nested folders) to prevent Windows path issues.
- Avoid spaces/special characters in the folder name for CI friendliness.

2. Create a Playwright + TypeScript project

2.1 Initialize the project

1. Create a folder and initialize npm.
2. Install Playwright test runner.
3. Install browsers.

```
mkdir pw-ts-framework
cd pw-ts-framework
npm init -y
npm i -D @playwright/test
npx playwright install
```

2.2 Playwright project template (optional)

- If you prefer the official starter structure, you can scaffold a project via:

```
npm init playwright@latest
```

Note: Choose TypeScript when prompted. This sets up an initial config, example tests, and a CI workflow template.

2.3 TypeScript setup (if you did not use the template)

- Install TypeScript and Node typings:

```
npm i -D typescript ts-node @types/node
```

2.4 Create tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "commonjs",
    "moduleResolution": "node",
    "strict": true,
    "esModuleInterop": true,
    "resolveJsonModule": true,
    "types": ["node"],
    "outDir": "dist",
    "baseUrl": ".",
    "paths": {
      "@pages/*": ["src/pages/*"],
      "@utils/*": ["src/utils/*"],
      "@fixtures/*": ["src/fixtures/*"],
      "@data/*": ["src/data/*"]
    }
  },
  "include": ["src", "tests", "playwright.config.ts"]
}
```

3. VS Code configuration (recommended)

3.1 Workspace settings

- Create .vscode/settings.json to keep consistent formatting, linting, and test discovery.

```
{
  "editor.formatOnSave": true,
  "editor.defaultFormatter": "esbenp.prettier-vscode",
  "typescript.preferences.importModuleSpecifier": "non-relative",
  "playwright.reuseBrowser": true
}
```

3.2 Debugging with VS Code

- Create .vscode/launch.json so you can debug Playwright tests with breakpoints.

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Debug Playwright Tests",
      "type": "node",
      "request": "launch",
      "program": "${workspaceFolder}/node_modules/@playwright/test/cli.js",
      "args": ["test", "--headed", "--project=chromium"],
      "console": "integratedTerminal",
      "internalConsoleOptions": "neverOpen"
    }
  ]
}
```

4. Framework structure (scalable project layout)

4.1 Recommended folder structure

```
pw-ts-framework/
  playwright.config.ts
  package.json
  tsconfig.json
  .env
  .gitignore
  .vscode/
  src/
    pages/           # Page Objects
    api/             # API clients (optional)
    fixtures/        # Custom fixtures
    utils/           # Helpers, wrappers, constants
    data/            # Test data (json/ts), schemas
    types/           # Shared TypeScript types
  tests/
```

```
    ui/              # UI tests
    api/             # API tests
    e2e/             # End-to-end flows
  playwright-report/
  test-results/
```

4.2 What goes where (rules)

- tests/: only test specs and orchestration. Keep specs thin.
- src/pages/: page objects. Keep locators and actions here.
- src/fixtures/: shared setup and typed fixtures (auth, API client, shared pages).
- src/utils/: reusable helpers (dates, random generators, wait wrappers).
- src/data/: test data objects, JSON payloads, schemas, constants.

5. package.json scripts (recommended)

- Scripts make CI and local usage consistent. Below is a solid baseline.

```
{
  "scripts": {
    "test": "playwright test",
    "test:headed": "playwright test --headed",
    "test:ui": "playwright test --ui",
    "test:debug": "playwright test --debug",
    "test:chromium": "playwright test --project=chromium",
    "test:firefox": "playwright test --project=firefox",
    "test:webkit": "playwright test --project=webkit",
    "test:smoke": "playwright test --grep @smoke",
    "test:regression": "playwright test --grep @regression",
    "report": "playwright show-report",
    "codegen": "playwright codegen"
  }
}
```

6. playwright.config.ts (framework side - complete concepts)

6.1 Production-ready baseline config

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

export default defineConfig({
  testDir: './tests',
  timeout: 30 * 1000,
  expect: { timeout: 10 * 1000 },

  fullyParallel: true,
  forbidOnly: !!process.env.CI,
  retries: process.env.CI ? 2 : 0,
```

```
  workers: process.env.CI ? 2 : undefined,

  reporter: [
    ['html', { open: 'never' }],
    ['list']
  ],

  use: {
    baseURL: process.env.BASE_URL || 'https://example.com',
    headless: true,
    actionTimeout: 15 * 1000,
    trace: 'on-first-retry',
    screenshot: 'only-on-failure',
    video: 'retain-on-failure',
    viewport: { width: 1366, height: 768 },
    ignoreHTTPSErrors: true
  },

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

  outputDir: 'test-results'
});
```

6.2 Key config concepts

- testDir: where Playwright looks for specs.
- timeout: max time per test; expect.timeout: max time for assertions.
- fullyParallel/workers: controls parallel execution.
- retries: rerun failing tests (useful in CI).
- reporter: HTML report + any additional reporters.
- use: shared context options like baseURL, headless, trace, screenshot, video, viewport.
- projects: run the same suite across browsers/devices.

6.3 Tags and grep strategy

- Tag tests using @smoke, @regression, @sanity, @api, @ui, etc.
- Run targeted suites with --grep or --grep-invert.

```
import { test } from '@playwright/test';

test('login works @smoke', async ({ page }) => {
  // ...
});
```

```
npx playwright test --grep "@smoke"
npx playwright test --grep "@regression" --grep-invert "@flaky"
```

7. Core Playwright concepts (technical)

7.1 Test runner: test, describe, hooks

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

test.describe('Login', () => {
  test.beforeEach(async ({ page }) => {
    await page.goto('/login');
  });

  test('valid login', async ({ page }) => {
    await page.getByTestId('username').fill('user');
    await page.getByTestId('password').fill('pass');
    await page.getByRole('button', { name: 'Sign in' }).click();
    await expect(page).toHaveURL(/dashboard/);
  });
});
```

- test.describe: groups tests; can also configure per-suite settings.
- beforeEach/afterEach: setup/cleanup per test.
- beforeAll/afterAll: suite-level setup/teardown (avoid heavy UI state sharing).

7.2 Locators (recommended patterns)

- Prefer role-based and test-id locators for stability.
- Avoid brittle CSS/XPath if better options exist.

```
const loginButton = page.getByRole('button', { name: 'Sign in' });
const email = page.getByTestId('email');
const menu = page.getByRole('navigation');
```

Note: Best practice: add data-testid attributes in your app for stable selectors.

7.3 Assertions with expect

```
await expect(page.getByText('Welcome')).toBeVisible();
await expect(page).toHaveTitle(/Dashboard/);
await expect(page.locator('#count')).toHaveText('10');
```

7.4 Auto-waiting vs explicit waits

- Playwright auto-waits for actionable conditions (visibility, enabled, stable).
- Avoid hard waits like waitForTimeout unless debugging.
- Use expect assertions + locator actions to leverage auto-wait.

7.5 Frames, popups, downloads, dialogs

```
// Frames
const frame = page.frameLocator('#payment-iframe');
await frame.getByRole('button', { name: 'Pay' }).click();

// Popup / new tab
```

```
const [popup] = await Promise.all([
  page.waitForEvent('popup'),
  page.getByRole('link', { name: 'Open terms' }).click()
]);
await expect(popup).toHaveURL(/terms/);

// Download
const [download] = await Promise.all([
  page.waitForEvent('download'),
  page.getByRole('button', { name: 'Export' }).click()
]);
await download.saveAs('downloads/report.csv');

// Dialog
page.on('dialog', async d => await d.accept());
```

7.6 Network: intercept, mock, and wait

```
// Wait for a response
const resp = await page.waitForResponse(r => r.url().includes('/api/login') &&
r.status() === 200);

// Route / mock
await page.route('**/api/users', async route => {
  const json = [{ id: 1, name: 'Mock User' }];
  await route.fulfill({ json });
});
```

8. Framework patterns (POM + Fixtures)

8.1 Page Object Model (POM)

- POM keeps selectors and UI actions in one place.
- Keep page methods business-focused (loginAs(user)) instead of low-level clicks everywhere.

```
// src/pages/LoginPage.ts
import { Page, Locator, expect } from '@playwright/test';

export class LoginPage {
  readonly page: Page;
  readonly username: Locator;
  readonly password: Locator;
  readonly submit: Locator;

  constructor(page: Page) {
    this.page = page;
    this.username = page.getByTestId('username');
    this.password = page.getByTestId('password');
    this.submit = page.getByRole('button', { name: 'Sign in' });
  }
```

```
  async goto() {
    await this.page.goto('/login');
  }

  async login(user: string, pass: string) {
    await this.username.fill(user);
    await this.password.fill(pass);
    await this.submit.click();
  }

  async assertLoginSuccess() {
    await expect(this.page).toHaveURL(/dashboard/);
  }
}
```

8.2 Thin tests (ideal spec style)

```
// tests/ui/login.spec.ts
import { test } from '@playwright/test';
import { LoginPage } from '@pages/LoginPage';

test('valid login @smoke', async ({ page }) => {
  const login = new LoginPage(page);
  await login.goto();
  await login.login(process.env.USERNAME!, process.env.PASSWORD!);
  await login.assertLoginSuccess();
});
```

8.3 Custom fixtures (typed + reusable)

- Fixtures extend the built-in test object with your own objects.
- Use fixtures for shared pages, API clients, logged-in context, and data builders.

```
// src/fixtures/baseFixture.ts
import { test as base } from '@playwright/test';
import { LoginPage } from '@pages/LoginPage';

type MyFixtures = {
  loginPage: LoginPage;
};

export const test = base.extend<MyFixtures>({
  loginPage: async ({ page }, use) => {
    await use(new LoginPage(page));
  }
});

export { expect } from '@playwright/test';
```

8.4 Storage state (fast auth for large suites)

- Use storageState to save authenticated cookies/localStorage once, then reuse.
- This reduces flaky UI-login steps and speeds up execution.

```
// global-setup.ts
import { chromium, FullConfig } from '@playwright/test';

async function globalSetup(config: FullConfig) {
  const browser = await chromium.launch();
  const page = await browser.newPage();
  await page.goto(process.env.BASE_URL + '/login');

  await page.getByTestId('username').fill(process.env.USERNAME!);
  await page.getByTestId('password').fill(process.env.PASSWORD!);
  await page.getByRole('button', { name: 'Sign in' }).click();

  await page.context().storageState({ path: 'src/data/auth.json' });
  await browser.close();
}
export default globalSetup;
```

Note: Then set use.storageState in config or per-suite.

```
// playwright.config.ts (snippet)
globalSetup: require.resolve('./global-setup'),
use: { storageState: 'src/data/auth.json' }
```

9. Environment management (.env) and test data

9.1 dotenv setup

- Store secrets outside code using .env (never commit secrets).
- Load env vars via dotenv/config.

```
npm i -D dotenv

# .env (example)
BASE_URL=https://your-app.com
USERNAME=demo_user
PASSWORD=demo_pass

// playwright.config.ts (top)
import 'dotenv/config';
```

Note: Add .env to .gitignore so credentials are not committed.

9.2 Test data patterns

- Small data: TypeScript objects in src/data.
- Large data: JSON files + resolveJsonModule.
- Dynamic data: generate per test (timestamp-based unique values).

```
// src/utils/random.ts
export const uniqueEmail = () =>
  `user_${Date.now()}_${Math.floor(Math.random() * 1000)}@example.com`;
```

10. Reporting and artifacts

10.1 HTML report

- Playwright includes a rich HTML report by default.

```
npx playwright show-report
```

10.2 Trace Viewer

- Trace is the best debugging tool: steps + network + DOM snapshots.
- Open trace:

```
npx playwright show-trace test-results/**/trace.zip
```

10.3 Allure (optional)

- If your org uses Allure, add allure-playwright reporter.

```
npm i -D allure-playwright
```

```
// playwright.config.ts (snippet)
reporter: [['html'], ['allure-playwright']]
```

11. CI/CD examples

11.1 GitHub Actions

```
# .github/workflows/playwright.yml
name: Playwright Tests
on:
  push:
    branches: [ "main" ]
  pull_request:

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: "lts/*"
      - name: Install deps
        run: npm ci
      - name: Install browsers
        run: npx playwright install --with-deps
      - name: Run tests
```

```
        env:
          BASE_URL: ${{ secrets.BASE_URL }}
          USERNAME: ${{ secrets.USERNAME }}
          PASSWORD: ${{ secrets.PASSWORD }}
        run: npx playwright test
      - name: Upload HTML report
        if: always()
        uses: actions/upload-artifact@v4
        with:
          name: playwright-report
          path: playwright-report
```

11.2 Jenkins Pipeline

```
pipeline {
  agent any
  stages {
    stage('Checkout') { steps { checkout scm } }
    stage('Install') { steps { sh 'npm ci' } }
    stage('Browsers') { steps { sh 'npx playwright install --with-deps' } }
    stage('Test') {
      steps { sh 'npx playwright test' }
      post {
        always {
          archiveArtifacts artifacts: 'playwright-report/**', fingerprint: true
          archiveArtifacts artifacts: 'test-results/**', fingerprint: true
        }
      }
    }
  }
}
```

12. Advanced framework topics

12.1 Parallelism and sharding

- workers controls concurrency. Too many workers can overload the test environment.
- Use sharding to split runs across machines in CI.

```
npx playwright test --shard=1/3
npx playwright test --shard=2/3
npx playwright test --shard=3/3
```

12.2 Retry strategy

- retries reruns entire failing tests (CI stability).
- expect has polling built in; prefer expect over manual waits.
- Avoid hiding real bugs with too many retries.

12.3 Test isolation: browser, context, page

- Browser: one process, multiple isolated contexts.
- Context: incognito-like; separate cookies/storage; recommended per test.
- Page: a tab. Most tests use one page per test.

12.4 API testing (request fixture)

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

test('GET users @api', async ({ request }) => {
  const res = await request.get('/api/users');
  expect(res.ok()).toBeTruthy();
  const body = await res.json();
  expect(Array.isArray(body)).toBeTruthy();
});
```

12.5 Visual testing (screenshots)

```
await expect(page).toHaveScreenshot('home.png', { fullPage: true });
```

12.6 Linting and formatting

```
npm i -D eslint @typescript-eslint/parser @typescript-eslint/eslint-plugin prettier
eslint-config-prettier
```

```
// .eslintrc.json (baseline)
{
  "parser": "@typescript-eslint/parser",
  "plugins": ["@typescript-eslint"],
  "extends": [
    "eslint:recommended",
    "plugin:@typescript-eslint/recommended",
    "prettier"
  ],
  "env": { "node": true, "es2022": true }
}
```

12.7 Common flaky test causes + fixes

- Unstable selectors -> use data-testid or role-based locators.
- Race conditions -> rely on expect(...) instead of sleeps.
- Animations -> disable in test env or wait for stable states.
- Shared state -> ensure isolation; avoid reusing state incorrectly.
- Backend slowness -> tune expect timeout selectively.

13. Mini reference framework (copy-paste starter)

13.1 Example page + fixture + test

```
// src/pages/DashboardPage.ts
import { Page, expect } from '@playwright/test';

export class DashboardPage {
  constructor(private page: Page) {}
  async assertLoaded() {
    await expect(this.page.getByRole('heading', { name: /dashboard/i })).toBeVisible();
  }
}
```

```
// src/fixtures/appFixture.ts
import { test as base } from '@playwright/test';
import { LoginPage } from '@pages/LoginPage';
import { DashboardPage } from '@pages/DashboardPage';

type AppFixtures = { login: LoginPage; dashboard: DashboardPage };

export const test = base.extend<AppFixtures>({
  login: async ({ page }, use) => await use(new LoginPage(page)),
  dashboard: async ({ page }, use) => await use(new DashboardPage(page))
});

export { expect } from '@playwright/test';
```

```
// tests/e2e/login-flow.spec.ts
import { test } from '@fixtures/appFixture';

test('login flow @regression', async ({ login, dashboard }) => {
  await login.goto();
  await login.login(process.env.USERNAME!, process.env.PASSWORD!);
  await dashboard.assertLoaded();
});
```

14. Interview-level framework points

- Why Playwright: auto-waiting, faster execution, built-in parallelization, tracing.
- Design patterns: POM + Fixtures keep tests clean and scalable.
- Observability: traces, screenshots, videos, logs, and consistent reporting.
- CI readiness: headless runs, retries, sharding, artifacts, secrets via env vars.

15. Troubleshooting checklist

4. Install issues (Linux CI) -> npx playwright install --with-deps
5. Tests not discovered -> ensure testDir + naming like *.spec.ts
6. Auth issues -> confirm BASE_URL + storageState path; re-generate auth.json

7. Selector failures -> prefer getByRole/getByTestId; validate in trace viewer
8. Flaky waits -> remove timeouts; add expect() for stable conditions

— Thank You —