Great choice! **Playwright** is one of the most powerful and modern tools for web automation testing. Here’s a comprehensive checklist of what you'll need to complete a Playwright-based QA automation project—from setup to execution and CI/CD integration.

**✅ Playwright QA Project Requirements Checklist**

**1. 🔧 Environment Setup**

* ✅ Node.js (v16 or higher recommended)
* ✅ Code editor (e.g., **VS Code**)
* ✅ Install Playwright:
* npm init playwright@latest

**2. 🧪 Testing Framework**

Playwright comes with its **own test runner** (@playwright/test), so you don’t need to install Jest or Mocha separately.

* 📦 Playwright Test features:
  + Parallel test execution
  + Auto-waiting
  + Screenshot & video capture
  + Built-in reporters

**3. 📁 Project Structure**

Organize your tests like this:

project-root/

├── tests/

│ ├── login.spec.ts

│ ├── checkout.spec.ts

├── pages/

│ ├── LoginPage.ts

│ ├── CheckoutPage.ts

├── utils/

│ ├── test-data.ts

│ ├── helpers.ts

├── playwright.config.ts

├── package.json

**4. 💡 Core Concepts to Implement**

* **Page Object Model (POM)** — maintainable, reusable page interactions
* **Test data management** — static JSON, .env, or dynamic data from APIs
* **Assertions** — use expect() for validations
* **Hooks** — beforeEach, afterAll, etc.
* **Locators** — prefer getByRole(), getByTestId(), etc.

**5. 🧪 Types of Tests to Include**

* ✅ **Smoke Tests**
* ✅ **Regression Tests**
* ✅ **Negative Tests**
* ✅ **Cross-browser Tests** (chromium, firefox, webkit)
* ✅ **Visual/Screenshot Testing** (optional)
* ✅ **API Integration Tests** (optional with request.newContext())

**6. 📦 Advanced Features**

* 🔁 Retry logic for flaky tests
* 📷 Screenshots & videos on failure
* ⏱ Tracing (time-travel debugging)
* 📊 Custom reporters (HTML, Allure, JUnit)
* 🔄 Test parallelism & sharding

**7. ⚙️ CI/CD Integration**

Integrate into GitHub Actions, GitLab CI, Azure Pipelines, or Jenkins.

Example GitHub Actions file:

name: Playwright Tests

on: [push, pull\_request]

jobs:

test:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- uses: actions/setup-node@v3

with:

node-version: '18'

- run: npm ci

- run: npx playwright install --with-deps

- run: npx playwright test

**8. 📚 Documentation and Best Practices**

* Use comments and proper naming
* Keep tests atomic
* Use tags to group test types
* Version control (Git)
* Use .env files for credentials/secrets

**9. 🔐 Security & Secrets**

* Never hard-code credentials
* Use .env, GitHub secrets, or CI secrets manager

**10. 📦 Optional Integrations**

* Allure for reporting
* Docker for environment consistency
* BrowserStack/Sauce Labs (for real device/cloud testing)
* Slack/email notifications for test failures

## ✅ Getting Started with Playwright + JavaScript

Here’s how to set up and use **Playwright with JavaScript** step by step:

### 1. 🔧 ****Setup Your Project****

Open your terminal:

bash

CopyEdit

mkdir playwright-js-demo

cd playwright-js-demo

npm init playwright@latest

When prompted:

* ✅ Choose **JavaScript**
* ✅ Select browsers you want (Chromium, Firefox, WebKit)
* ✅ Say "yes" to GitHub Actions (if you want CI)

This will install everything and create a starter project.

### 2. 📁 ****Project Structure****

After setup, your folder will look like this:

lua

CopyEdit

playwright.config.js

tests/

├── example.spec.js

package.json

### 3. ✍️ ****Write a Sample Test****

In tests/example.spec.js:

js

CopyEdit

const { test, expect } = require('@playwright/test');

test('homepage has title', async ({ page }) => {

await page.goto('https://example.com');

await expect(page).toHaveTitle(/Example Domain/);

});

### 4. ▶️ ****Run Your Tests****

bash

CopyEdit

npx playwright test

Use:

* npx playwright test --headed → See the browser
* npx playwright test --project=firefox → Run in Firefox

### 5. 📸 ****View Reports****

After running tests, you can view an HTML report:

bash

CopyEdit

npx playwright show-report

### 6. ✅ ****Optional: Add Page Object Model****

Create a new file: pages/LoginPage.js

js

CopyEdit

exports.LoginPage = class LoginPage {

constructor(page) {

this.page = page;

this.usernameInput = page.locator('#username');

this.passwordInput = page.locator('#password');

this.loginButton = page.locator('#login');

}

async login(username, password) {

await this.usernameInput.fill(username);

await this.passwordInput.fill(password);

await this.loginButton.click();

}

};

Use it in your test:

js

CopyEdit

const { LoginPage } = require('../pages/LoginPage');

test('user can login', async ({ page }) => {

const loginPage = new LoginPage(page);

await page.goto('https://example.com/login');

await loginPage.login('admin', 'secret');

await expect(page.locator('#welcome')).toBeVisible();

});

### 🔄 Extras You Can Add Later:

* 💡 .env support with dotenv
* 📦 Reporting with Allure
* 🧪 API testing with request.newContext()
* 🔁 CI/CD integration with GitHub Actions