**PLAYWRIGHT AUTOMATION TESTING**

**1. Q: What is Playwright and what are its key features?**

A: Playwright is a modern automation testing framework developed by Microsoft that enables reliable end-to-end testing for web applications. Key features include:

- Cross-browser support (Chromium, Firefox, WebKit)

- Auto-wait capabilities

- Built-in test runner and assertions

- Network interception

- Mobile emulation

- Parallel test execution

- Built-in reporting

**2. Q: How do you install Playwright in a new project?**

A: There are two main ways:

1. Using NPM init:

npm init playwright@latest

2. Manual installation:

npm install --save-dev playwright

npx playwright install

**3. Q: What are the different ways to locate elements in Playwright?**

A: Based on our project, we use several locator strategies:

- Property locators: page.locator('#login2')

- CSS locators: page.locator('.header-logo')

- XPath locators: page.locator("//div[@id='tbodyid']//h4/a")

- Built-in locators:

\* getByRole()

\* getByText()

\* getByPlaceholder()

\* getByAltText()

\* getByLabel()

\* getByTitle()

\* getByTestId()

**4. Q: How do you handle different types of assertions in Playwright?**

A: Our project demonstrates two types of assertions:

1. Hard Assertions:

await expect(page).toHaveTitle('STORE');

await expect(page).toHaveURL('https://demoblaze.com/index.html');

2. Soft Assertions:

await expect.soft(page).toHaveTitle('STORE123');

**5. Q: How do you handle dropdowns in Playwright?**

A: Based on our Dropdowns.spec.js, there are multiple ways:

// By value

await page.selectOption('#country', 'Germany');

// By label

await page.selectOption('#country', {label: 'Germany'});

// By index

await page.selectOption('#country', {index: 3});

**6. Q: How do you handle checkboxes and radio buttons in Playwright?**

A: From our project examples:

**// Checkbox**

await page.locator('#checkbox').check();

await page.locator('#checkbox').uncheck();

**// Radio button**

await page.locator('#male').check();

await expect(page.locator('#male')).toBeChecked();

**7. Q: How do you run tests in different browsers using Playwright?**

A: Our project configuration shows:

**# Run in Chromium**

npx playwright test --project=chromium

**# Run in Firefox**

npx playwright test --project=firefox

**# Run in WebKit**

npx playwright test --project=webkit

**8. Q: How do you handle multiple elements in Playwright?**

A: From LocatingMultipleElements.spec.js:

const links = await page.$$('a');

for (const link of links) {

console.log(await link.textContent());

}

**9. Q: What is the difference between page.locator() and page.$() in Playwright?**

A: - page.locator() is the recommended modern way to locate elements

- page.$() is the older jQuery-style selector which returns the first matching element

- page.locator() has better built-in auto-waiting mechanisms

**10. Q: How do you handle test reports in Playwright?**

A: Our project uses HTML reporter configured in playwright.config.js:

reporter: 'html'

To view reports:

npx playwright show-report

**11. Q: How do you handle test retries in Playwright?**

A: From our playwright.config.js:

retries: process.env.CI ? 2 : 0,

This configures 2 retries in CI environment and 0 in local.

**12. Q: How do you run tests in parallel in Playwright?**

A: Our config shows:

fullyParallel: true,

workers: process.env.CI ? 1 : undefined,

This enables parallel execution locally while running sequentially in CI.

**13. Q: How do you debug tests in Playwright?**

A: Our project supports several debugging methods:

**# Debug mode**

npx playwright test --debug

**# Headed mode**

npx playwright test --headed

**# Both**

npx playwright test --headed --debug

**14. Q: What is the purpose of test.beforeEach() in Playwright?**

A: It's used to set up test prerequisites that should run before each test case. Example from our project:

test.beforeEach(async ({ page }) => {

await page.goto('https://demo.playwright.dev/todomvc');

});

**15. Q: How do you handle iframes in Playwright?**

A: While not shown in our current project, you can handle iframes using:

const frame = page.frameLocator('#iframe-id');

await frame.locator('.some-element').click();

**16. Q: How do you implement Page Object Model (POM) in Playwright?**

A: From our POM.spec.js and pages/ directory:

// LoginPage.js

export class LoginPage {

constructor(page) {

this.page = page;

this.loginLink = "#login2";

this.usernameInput = "#loginusername";

this.passwordInput = "#loginpassword";

this.loginButton = '//button[normalize-space()="Log in"]';

}

async login(username, password) {

await this.page.locator(this.loginLink).click();

await this.page.locator(this.usernameInput).fill(username);

await this.page.locator(this.passwordInput).fill(password);

await this.page.locator(this.loginButton).click();

}

}

**17. Q: How do you handle multiple windows/pages in Playwright?**

A: From our HandlingWindows.spec.js:

const browser = await chromium.launch()

const context = await browser.newContext()

const page1 = await context.newPage()

const page2 = await context.newPage()

const allPages = context.pages()

console.log("No of Pages created:", allPages.length)

**18. Q: How do you handle screenshots in Playwright?**

A: From our Screenshot.spec.js:

await page.screenshot({

path: 'tests/screenshots/' + Date.now() + 'HomePage.png'

});

**19. Q: How do you handle video recording in Playwright?**

A: Configure in playwright.config.js and use in VideoRecording.spec.js:

**// In config**

use: {

video: 'on-first-retry'

}

**// In test**

test('Video Recording', async ({ page }) => {

// Test actions will be recorded automatically

});

**20. Q: How do you handle tracing in Playwright?**

A: From our Tracing.spec.js:

**// Configure in playwright.config.js**

use: {

trace: 'on-first-retry'

}

**// View trace**

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

**21. Q: How do you handle test hooks in Playwright?**

A: From our Hooks.spec.js:

test.beforeAll(async ({ browser }) => {

const context = await browser.newContext();

page = await context.newPage();

await page.goto('https://demoblaze.com/index.html');

// Login setup

});

**22. Q: How do you handle test reporting in Playwright?**

A: From our Reporters.spec.js and configuration:

**// In playwright.config.js**

reporter: [

['html'],

['list']

]

**// Run tests and generate report**

npx playwright test

npx playwright show-report

**23. Q: How do you handle test tags in Playwright?**

A: From our Tags.spec.js:

test('Test1@sanity', async ({ page }) => {

// Test code

});

**// Run with tags**

npx playwright test --grep "@sanity"

npx playwright test --grep-invert "@regression"

**24. Q: How do you handle test annotations in Playwright?**

A: From our Annotations.spec.js:

test.skip('Test2', async ({ page }) => {

// This test will be skipped

});

test.fixme('Test4', async ({ page }) => {

// This test needs fixing

});

**25. Q: How do you handle API testing in Playwright?**

A: From our APITests.spec.js:

test('API Test', async ({ request }) => {

**// POST request**

const response = await request.post('https://reqres.in/api/users', {

data: {

"name": "shehroz",

"job": "trainer"

},

headers: {

"Accept": "application/json"

}

});

expect(response.status()).toBe(201);

const jsonResponse = await response.json();

console.log(jsonResponse);

});

**26. Q: How do you implement test retries for flaky tests in Playwright?**

A: From our RetryTest.spec.js and configuration:

**// In playwright.config.js**

use: {

retries: 2,

timeout: 30000,

}

**// Running tests with retries**

npx playwright test --retries=3

**27. Q: How do you handle test data in Playwright?**

A: Our project demonstrates several approaches:

- Page Object Model for test data organization

- External test data files

- API response data handling

- Environment variables for configuration

**28. Q: What is the difference between hard and soft assertions in Playwright?**

A: From our Assertions\_soft.spec.js:

**// Hard assertions - test stops on failure**

await expect(page).toHaveTitle('STORE');

**// Soft assertions - test continues on failure**

await expect.soft(page).toHaveTitle('STORE123');

**29. Q: How do you implement Page Object Model with API testing?**

A: Our project combines POM with API testing:

**// API methods in page objects**

class APIPage {

constructor(request) {

this.request = request;

}

async createUser(userData) {

return await this.request.post('https://reqres.in/api/users', {

data: userData

});

}

}

**30. Q: How do you handle test reporting with Allure in Playwright?**

A: From our project configuration:

**// Install allure-playwright**

npm i -D @playwright/test allure-playwright

**// Configure in playwright.config.js**

reporter: [['allure-playwright', {outputFolder: 'allure-results'}]]

**// Generate reports**

npx playwright test --reporter=allure-playwright

**// View reports**

allure serve allure-results

**31. Q: What are the advantages of using Playwright over other automation tools?**

A: Key advantages include:

- Cross-browser support with a single API

- Auto-wait capabilities reducing flaky tests

- Network interception and mocking

- Modern architecture with async/await support

- Better performance with native automation

- Built-in test runner and assertions

- Mobile emulation support

- Strong TypeScript support

**32. Q: How do you handle dynamic web elements in Playwright?**

A: Several approaches from our project:

**// Wait for element to be visible**

await page.waitForSelector('.dynamic-element');

**// Use regex in locators**

await page.locator('text=/Submit$/').click();

**// Dynamic ID handling**

await page.locator('div[id\*="partial-id"]').click();

**// Wait for network idle**

await page.waitForLoadState('networkidle');

**33. Q: How do you handle authentication in Playwright tests?**

A: Multiple approaches:

**// 1. Login before tests using hooks**

test.beforeAll(async ({ browser }) => {

const context = await browser.newContext();

const page = await context.newPage();

await page.goto('/login');

await page.fill('#username', 'user');

await page.fill('#password', 'pass');

await page.click('#login');

// Save storage state

await context.storageState({ path: 'auth.json' });

});

**// 2. Reuse authentication state**

const context = await browser.newContext({

storageState: 'auth.json'

});

**34. Q: How do you handle downloads in Playwright?**

A: Example implementation:

**// Wait for download**

const downloadPromise = page.waitForEvent('download');

await page.click('#download-button');

const download = await downloadPromise;

**// Save downloaded file**

await download.saveAs('./downloads/file.pdf');

**35. Q: How do you handle shadow DOM elements in Playwright?**

A: Playwright handles shadow DOM automatically:

**// Access element in shadow DOM**

await page.locator('pierce/#shadow-button').click();

**// Using deep selector**

await page.locator('div >> #shadow-content').click();

**36. Q: How do you implement data-driven testing in Playwright?**

A: Example using test data:

const testData = [

{ username: 'user1', password: 'pass1' },

{ username: 'user2', password: 'pass2' }

];

for (const data of testData) {

test(`Login test for ${data.username}`, async ({ page }) => {

await page.fill('#username', data.username);

await page.fill('#password', data.password);

});

}

**37. Q: How do you handle network requests and responses in Playwright?**

A: Multiple approaches:

**// 1. Route interception**

await page.route('\*\*/api/users', route => {

route.fulfill({

status: 200,

body: JSON.stringify({ data: 'mocked' })

});

});

**// 2. Wait for response**

const responsePromise = page.waitForResponse('\*\*/api/users');

await page.click('#submit');

const response = await responsePromise;

expect(response.status()).toBe(200);

**38. Q: How do you handle parallel test execution in Playwright?**

A: Configuration in playwright.config.js:

module.exports = {

workers: 3, **// Number of parallel workers**

fullyParallel: true, **// Run tests in parallel**

projects: [

{

name: 'chromium',

use: { ...devices['Desktop Chrome'] }

},

{

name: 'firefox',

use: { ...devices['Desktop Firefox'] }

}

]

};

```

**39. Q: How do you handle environment variables in Playwright?**

A: Multiple approaches:

**// 1. Using dotenv**

require('dotenv').config();

**// 2. In playwright.config.js**

use: {

baseURL: process.env.BASE\_URL,

extraHTTPHeaders: {

'API-Key': process.env.API\_KEY

}

}

**// 3. In test**

test('Environment test', async ({ page }) => {

await page.goto(process.env.TEST\_URL);

});

**40. Q: How do you handle test timeouts in Playwright?**

A: Various timeout configurations:

**// 1. Global timeout in config**

timeout: 30000,

**// 2. Test-level timeout**

test('Slow test', async ({ page }) => {

test.setTimeout(60000);

// Test code

});

**// 3. Action-level timeout**

await page.click('button', { timeout: 10000 });

**41. Q: How do you handle visual testing in Playwright?**

A: Using screenshot comparison:

**// 1. Take screenshot**

await page.screenshot({

path: 'screenshot.png',

fullPage: true

});

**// 2. Compare screenshots**

await expect(page).toHaveScreenshot('baseline.png', {

maxDiffPixels: 100

});

**42. Q: What are fixtures in Playwright and how do you use them?**

A: Custom fixtures example:

const test = base.extend({

loggedInPage: async ({ page }, use) => {

await page.goto('/login');

await page.fill('#username', 'user');

await page.click('#login');

await use(page);

}

});

test('using custom fixture', async ({ loggedInPage }) => {

await loggedInPage.click('#profile');

});