techSmartHire

Top 30 Playwright TypeScript/ JavaScript Interview Questions

For comprehensive QA Learning tutorials - visit - **rahulshettyacademy.com**
For QA Jobs, Assessments, Certificates - visit - **techsmarthire.com**

1. What is Playwright and how does it differ from Selenium?

Answer: Playwright is a modern end-to-end testing framework developed by Microsoft that supports Chromium, Firefox, and WebKit browsers. Key differences from Selenium:

- Auto-waits for elements to be actionable before performing actions
- Built-in support for modern web features (iframes, shadow DOM, web components)
- Native support for multiple browser contexts and pages
- Better handling of dynamic content with built-in wait mechanisms
- Single API for all browsers without needing browser-specific drivers

2. How do you handle multiple browser contexts in Playwright?

Answer:

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

test('multiple contexts example', async () => {
  const browser = await chromium.launch();

  // Create two independent browser contexts
  const context1 = await browser.newContext();
  const context2 = await browser.newContext();

  const page1 = await context1.newPage();
  const page2 = await context2.newPage();

  // Each context has independent cookies/storage
  await page1.goto('https://example.com');
  await page2.goto('https://example.com');

  await context1.close();
  await context2.close();
```

tSH

```
  await browser.close();
});
```

3. Explain Playwright's auto-waiting mechanism

Answer: Playwright automatically waits for elements to be actionable before performing actions. It checks:

- Element is attached to DOM
- Element is visible
- Element is stable (not animating)
- Element receives events (not obscured)
- Element is enabled (for interactive elements)

This eliminates most explicit waits and reduces flakiness in tests.

4. How do you handle iframes in Playwright?

Answer:

```
// Method 1: Using frameLocator
await page.frameLocator('iframe#myframe')
  .locator('button')
  .click();

// Method 2: Using frame() for named frames
const frame = page.frame('frameName');
await frame.locator('button').click();

// Method 3: Using contentFrame() for iframe elements
const iframeElement = page.locator('iframe#myframe');
const frame = await iframeElement.contentFrame();
await frame.locator('button').click();
```

5. What are locator strategies in Playwright and which is recommended?

Answer: Playwright supports multiple locator strategies:

- `page.getByRole()` - **Recommended** (accessibility-focused)
- `page.getByText()` - For visible text
- `page.getByLabel()` - For form fields with labels

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

// test file
test('login test', async ({ page }) => {
  const loginPage = new LoginPage(page);
  await loginPage.goto();
  await loginPage.login('user@example.com', 'password123');
});
```

8. How do you handle API testing in Playwright?

Answer:

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

test('API testing example', async ({ request }) => {
  // GET request
  const response = await request.get('https://api.example.com/users');
  expect(response.ok()).toBeTruthy();
  const users = await response.json();

  // POST request
  const createResponse = await request.post('https://api.example.com/users', {
    data: {
      name: 'John Doe',
      email: 'john@example.com'
    },
    headers: {
      'Authorization': 'Bearer token123'
    }
  });
  expect(createResponse.status()).toBe(201);

  // PUT/DELETE similar pattern
});
```

9. How do you implement custom fixtures in Playwright?

Answer:

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

type MyFixtures = {
  loginPage: LoginPage;
  authenticatedPage: Page;
};

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

```
  },

  authenticatedPage: async ({ page }, use) => {
    await page.goto('/login');
    await page.fill('#username', 'testuser');
    await page.fill('#password', 'password');
    await page.click('button[type="submit"]');
    await use(page);
  }
});
```

10. How do you handle network interception and mocking?

Answer:

```
test('mock API response', async ({ page }) => {
  // Intercept and mock API calls
  await page.route('**/api/users', route => {
    route.fulfill({
      status: 200,
      contentType: 'application/json',
      body: JSON.stringify([
        { id: 1, name: 'Mock User' }
      ])
    });
  });

  // Abort specific requests
  await page.route('**/*.{png,jpg,jpeg}', route => route.abort());

  // Continue with modifications
  await page.route('**/api/**', route => {
    route.continue({
      headers: {
        ...route.request().headers(),
        'Authorization': 'Bearer mock-token'
      }
    });
  });

  await page.goto('/users');
});
```

11. What are the different types of waits in Playwright?

Answer:

```
// 1. Auto-wait (built-in, most common)
await page.click('button'); // Automatically waits

// 2. Wait for element state
await page.locator('button').waitFor({ state: 'visible' });
await page.locator('button').waitFor({ state: 'hidden' });

// 3. Wait for navigation
await Promise.all([
```

```
  page.waitForNavigation(),
  page.click('a')
]);

// 4. Wait for load state
await page.goto('https://example.com');
await page.waitForLoadState('networkidle');

// 5. Wait for selector
await page.waitForSelector('.dynamic-content');

// 6. Wait for function/condition
await page.waitForFunction(() => {
  return document.querySelectorAll('.items').length > 5;
});

// 7. Wait for timeout (not recommended)
await page.waitForTimeout(3000);

// 8. Wait for event
await page.waitForEvent('dialog');
```

12. How do you handle authentication and session storage?

Answer:

```
// Method 1: Save and reuse authentication state
test('login once', async ({ page }) => {
  await page.goto('/login');
  await page.fill('#username', 'user');
  await page.fill('#password', 'pass');
  await page.click('button[type="submit"]');

  // Save storage state
  await page.context().storageState({ path: 'auth.json' });
});

// playwright.config.ts
export default defineConfig({
  use: {
    storageState: 'auth.json' // Use saved state
  }
});

// Method 2: Setup project for authentication
export default defineConfig({
  projects: [
    {
      name: 'setup',
      testMatch: /.*\.setup\.ts/
    },
    {
      name: 'chromium',
      use: {
        ...devices['Desktop Chrome'],
        storageState: 'auth.json'
      },
      dependencies: ['setup']
    }
  ]
});
```

techSmartHire

13. How do you perform visual regression testing?

Answer:

```
test('visual regression test', async ({ page }) => {
  await page.goto('https://example.com');

  // Full page screenshot comparison
  await expect(page).toHaveScreenshot('homepage.png');

  // Element screenshot comparison
  await expect(page.locator('.header'))
    .toHaveScreenshot('header.png');

  // With custom threshold
  await expect(page).toHaveScreenshot('page.png', {
    maxDiffPixels: 100,
    threshold: 0.2
  });

  // Masked areas (ignore dynamic content)
  await expect(page).toHaveScreenshot({
    mask: [page.locator('.timestamp')],
    fullPage: true
  });
});
```

14. How do you handle browser contexts for parallel test isolation?

Answer:

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

// Each test gets a fresh browser context automatically
test('test 1', async ({ page, context }) => {
  // Isolated context with own cookies/storage
  await page.goto('https://example.com');
});

test('test 2', async ({ page, context }) => {
  // Different isolated context
  await page.goto('https://example.com');
});

// Custom context configuration
test.use({
  contextOptions: {
    viewport: { width: 1920, height: 1080 },
    geolocation: { latitude: 40.7128, longitude: -74.0060 },
    permissions: ['geolocation'],
    locale: 'en-US',
    timezoneId: 'America/New_York'
```

tSH

```
  }
});
```

15. How do you debug Playwright tests?

Answer:

```
// 1. Headed mode
npx playwright test --headed

// 2. Debug mode with inspector
npx playwright test --debug

// 3. Pause execution
test('debug test', async ({ page }) => {
  await page.goto('https://example.com');
  await page.pause(); // Opens Playwright Inspector
});

// 4. Slow motion
test.use({
  launchOptions: { slowMo: 1000 }
});

// 5. Screenshots on failure (in config)
use: {
  screenshot: 'only-on-failure',
  video: 'retain-on-failure',
  trace: 'on-first-retry'
}

// 6. Console logs
page.on('console', msg => console.log(msg.text()));

// 7. Playwright Inspector
PWDEBUG=1 npx playwright test

// 8. Trace viewer
npx playwright show-trace trace.zip
```

16. How do you handle dynamic content and shadow DOM?

Answer:

```
// Shadow DOM
test('shadow DOM handling', async ({ page }) => {
  // Method 1: Using piercing selector
  await page.locator('pierce=#shadow-button').click();

  // Method 2: Navigate shadow root explicitly
  const shadowHost = page.locator('#shadow-host');
  const shadowRoot = await shadowHost.evaluateHandle(
    el => el.shadowRoot
  );
  await shadowRoot.locator('button').click();
});
```

```
// Dynamic content
test('dynamic content', async ({ page }) => {
  // Auto-wait for element to appear
  await page.locator('.dynamic-element').click();

  // Wait for specific number of elements
  await page.locator('.list-item').nth(4).waitFor();

  // Wait for element to have text
  await expect(page.locator('.status'))
    .toHaveText('Complete', { timeout: 10000 });
});
```

17. How do you implement data-driven testing?

Answer:

```
// Method 1: Using test.describe with loop
const testData = [
  { username: 'user1', password: 'pass1' },
  { username: 'user2', password: 'pass2' },
  { username: 'user3', password: 'pass3' }
];

testData.forEach(data => {
  test(`login with ${data.username}`, async ({ page }) => {
    await page.goto('/login');
    await page.fill('#username', data.username);
    await page.fill('#password', data.password);
    await page.click('button[type="submit"]');
  });
});

// Method 2: Using fixtures from external file
import testCases from './testdata.json';

for (const testCase of testCases) {
  test(`test case: ${testCase.name}`, async ({ page }) => {
    // Test implementation using testCase data
  });
}

// Method 3: CSV/Excel data
import csv from 'csv-parser';
import fs from 'fs';

const users = [];
fs.createReadStream('users.csv')
  .pipe(csv())
  .on('data', (row) => users.push(row));
```

18. How do you handle alerts, dialogs, and popups?

Answer:

```
test('handle dialogs', async ({ page }) => {
  // Alert
  page.on('dialog', async dialog => {
    console.log(dialog.message());
    await dialog.accept();
  });

  // Confirm dialog
  page.on('dialog', async dialog => {
    expect(dialog.type()).toBe('confirm');
    await dialog.accept(); // or dialog.dismiss()
  });

  // Prompt dialog
  page.on('dialog', async dialog => {
    await dialog.accept('My input text');
  });

  // New window/tab popup
  const [popup] = await Promise.all([
    page.waitForEvent('popup'),
    page.click('a[target="_blank"]')
  ]);
  await popup.waitForLoadState();
  await popup.locator('h1').textContent();
});
```

19. How do you implement retry logic and test stability?

Answer:

```
// playwright.config.ts
export default defineConfig({
  retries: process.env.CI ? 2 : 0,

  use: {
    actionTimeout: 10000,
    navigationTimeout: 30000,

    // Automatic retries for assertions
    expect: {
      timeout: 5000,
      toHaveScreenshot: {
        maxDiffPixels: 100
      }
    }
  }
});

// Custom retry logic
test('with custom retry', async ({ page }) => {
  let attempts = 0;
  const maxAttempts = 3;

  while (attempts < maxAttempts) {
    try {
      await page.goto('https://flaky-site.com');
      await expect(page.locator('h1')).toBeVisible();
      break;
    } catch (error) {
      attempts++;
      if (attempts === maxAttempts) throw error;
      await page.waitForTimeout(1000);
```

```
    }
  }
});

// Soft assertions (continue on failure)
test('soft assertions', async ({ page }) => {
  await expect.soft(page.locator('.title')).toHaveText('Expected');
  await expect.soft(page.locator('.subtitle')).toBeVisible();
  // Test continues even if assertions fail
});
```

20. How do you optimize Playwright test execution?

Answer:

```
// playwright.config.ts
export default defineConfig({
  // Run tests in parallel
  workers: process.env.CI ? 4 : undefined,

  // Shard tests across multiple machines
  shard: { total: 4, current: 1 },

  // Reuse browser context
  fullyParallel: true,

  use: {
    // Reduce trace overhead
    trace: 'on-first-retry',
    video: 'retain-on-failure',

    // Faster navigation
    waitUntil: 'domcontentloaded'
  },

  // Test timeout
  timeout: 30000,

  // Group related tests
  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] }
    }
  ]
});

// Test optimization techniques
test.describe.configure({ mode: 'parallel' });

// Skip unnecessary tests
test.skip(({ browserName }) => browserName !== 'chromium');

// Use API calls instead of UI for setup
test.beforeEach(async ({ request }) => {
  await request.post('/api/setup', { data: { user: 'test' } });
```

techSmartHire

21. What is Playwright Codegen and how do you use it?

Answer: Playwright Codegen is a test generator tool that records user interactions and generates test code automatically.

```
# Basic codegen
npx playwright codegen https://example.com

# Codegen with specific browser
npx playwright codegen --browser=firefox https://example.com

# Codegen with custom viewport
npx playwright codegen --viewport-size=1280,720 https://example.com

# Codegen with device emulation
npx playwright codegen --device="iPhone 12" https://example.com

# Codegen with authentication (load saved state)
npx playwright codegen --load-storage=auth.json https://example.com

# Save generated test to file
npx playwright codegen --output=tests/generated.spec.ts https://example.com

# Codegen with specific language
npx playwright codegen --target=python https://example.com
# Targets: javascript, python, java, csharp

# Codegen with custom user agent
npx playwright codegen --user-agent="Custom Bot" https://example.com

# Record in a specific locale
npx playwright codegen --lang=es-ES https://example.com
```

Key Features:

- Records clicks, fills, navigation
- Generates locators using best practices
- Shows live preview of generated code
- Supports assertion generation (click Assert button)
- Can resume recording on existing tests

22. Explain Playwright UI Mode and its features

Answer: Playwright UI Mode is an interactive GUI for running, debugging, and analyzing tests.

```
# Launch UI Mode
npx playwright test --ui

# UI Mode with specific tests
npx playwright test tests/login.spec.ts --ui
```

tSH

```
# UI Mode with specific project
npx playwright test --project=chromium --ui
```

Key Features:

1. **Watch Mode:** Auto-reruns tests on file changes
2. **Time Travel:** Step through test execution with DOM snapshots
3. **Pick Locator:** Interactive locator picker tool
4. **Network Tab:** View all network requests
5. **Console Logs:** See browser console output
6. **Screenshots:** View screenshots at each step
7. **Trace Viewer:** Integrated trace viewing
8. **Filtering:** Filter tests by name, status, project
9. **Debugging:** Set breakpoints, step through code

```
// UI Mode vs Headed Mode:
// UI Mode benefits:
// - Interactive debugging
// - Visual test execution
// - Time travel through test steps
// - Better for development

// Headed Mode:
npx playwright test --headed // Just shows browser, less control
```

23. How do you configure NPM scripts for Playwright in package.json?

Answer:

```
{
"name": "playwright-project",
"version": "1.0.0",
"scripts": {
  "test": "playwright test",
  "test:headed": "playwright test --headed",
  "test:ui": "playwright test --ui",
  "test:debug": "playwright test --debug",
  "test:chrome": "playwright test --project=chromium",
  "test:firefox": "playwright test --project=firefox",
  "test:webkit": "playwright test --project=webkit",
  "test:all-browsers": "playwright test --project=chromium --project=firefox --project=webkit",

  "test:smoke": "playwright test --grep @smoke",
  "test:regression": "playwright test --grep @regression",
  "test:api": "playwright test tests/api",
  "test:e2e": "playwright test tests/e2e",

  "test:parallel": "playwright test --workers=4",
  "test:serial": "playwright test --workers=1",
  "test:shard": "playwright test --shard=1/4",

  "test:retry": "playwright test --retries=2",
```

techSmartHire

```
    "test:report": "playwright show-report",
    "test:trace": "playwright show-trace",

    "test:ci": "playwright test --reporter=blob",
    "test:local": "playwright test --headed --workers=1",

    "codegen": "playwright codegen",
    "codegen:auth": "playwright codegen --load-storage=auth.json",

    "install:browsers": "playwright install",
    "install:chrome": "playwright install chromium",
    "install:deps": "playwright install-deps",

    "test:update-snapshots": "playwright test --update-snapshots",
    "test:list": "playwright test --list",

    "test:specific": "playwright test tests/login.spec.ts",
    "test:grep": "playwright test --grep 'login'",
    "test:grep-invert": "playwright test --grep-invert 'slow'",

    "pretest": "npm run lint",
    "posttest": "npm run test:report"
  },
  "devDependencies": {
    "@playwright/test": "^1.48.0",
    "@types/node": "^20.10.0",
    "typescript": "^5.3.3"
  }
}
```

Usage Examples:

```
npm test                    # Run all tests
npm run test:headed         # Run in headed mode
npm run test:ui             # Open UI mode
npm run test:smoke          # Run smoke tests only
npm run test:chrome         # Run in Chrome only
npm run codegen             # Start codegen tool
```

24. What are the essential Playwright dependencies and their purposes?

Answer:

```
{
  "devDependencies": {
    // Core Playwright package - required
    "@playwright/test": "^1.48.0",

    // TypeScript support (if using TS)
    "typescript": "^5.3.3",
    "@types/node": "^20.10.0",

    // Additional reporters
    "playwright-html-reporter": "^1.0.0",
    "allure-playwright": "^2.15.0",

    // Test utilities
    "dotenv": "^16.3.1",              // Environment variables
```

tSH

```
    "faker": "^6.6.6",                // Test data generation

    // API testing helpers
    "axios": "^1.6.0",

    // Excel/CSV data handling
    "xlsx": "^0.18.5",
    "csv-parser": "^3.0.0",

    // Visual regression
    "@playwright/test": "^1.48.0"     // Built-in screenshot comparison
  },
  "dependencies": {
    // Usually keep empty - tests shouldn't have runtime dependencies
  }
}
```

Key Dependencies Explained:

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

// @types/node - Node.js type definitions
import * as fs from 'fs';
import * as path from 'path';

// dotenv - Environment configuration
import dotenv from 'dotenv';
dotenv.config();
const API_URL = process.env.API_URL;

// faker - Test data generation
import { faker } from '@faker-js/faker';
const email = faker.internet.email();
```

Installation Commands:

```
# Install Playwright
npm install -D @playwright/test

# Install browsers
npx playwright install

# Install system dependencies
npx playwright install-deps

# Install specific browser
npx playwright install chromium

# Install with TypeScript
npm install -D @playwright/test typescript @types/node
```

25. What are the most important Playwright terminal commands?

Answer:

Test Execution Commands:

```
# Run all tests
npx playwright test

# Run specific test file
```

```
npx playwright test tests/login.spec.ts

# Run tests matching pattern
npx playwright test login

# Run tests in specific directory
npx playwright test tests/e2e/

# Run with specific browser
npx playwright test --project=chromium
npx playwright test --project=firefox
npx playwright test --project=webkit

# Run in headed mode
npx playwright test --headed

# Run in debug mode
npx playwright test --debug

# Run specific test by line number
npx playwright test login.spec.ts:42

# Run tests by title/grep
npx playwright test --grep "login"
npx playwright test --grep "@smoke"
npx playwright test --grep-invert "@slow"

# Run with workers (parallel execution)
npx playwright test --workers=4
npx playwright test --workers=1  # Serial execution

# Run with retries
npx playwright test --retries=2

# Run with specific timeout
npx playwright test --timeout=60000

# Run and update snapshots
npx playwright test --update-snapshots

# Run with specific reporter
npx playwright test --reporter=html
npx playwright test --reporter=json
npx playwright test --reporter=junit
npx playwright test --reporter=list
npx playwright test --reporter=dot

# List all tests without running
npx playwright test --list

# Run last failed tests
npx playwright test --last-failed

# Run with maximum failures
npx playwright test --max-failures=3

# Shard tests (CI/CD)
npx playwright test --shard=1/4
npx playwright test --shard=2/4
```

Codegen & UI Commands:

```
# Launch codegen
npx playwright codegen https://example.com

# Codegen with options
npx playwright codegen --browser=firefox
npx playwright codegen --device="iPhone 12"
npx playwright codegen --viewport-size=1920,1080
npx playwright codegen --load-storage=auth.json
```

```
npx playwright codegen --save-storage=auth.json
npx playwright codegen --output=test.spec.ts

# UI Mode
npx playwright test --ui
npx playwright test tests/login.spec.ts --ui

# Open last HTML report
npx playwright show-report

# Open trace viewer
npx playwright show-trace trace.zip
```

Installation & Setup Commands:

```
# Install Playwright
npm init playwright@latest

# Install browsers
npx playwright install

# Install specific browser
npx playwright install chromium
npx playwright install firefox
npx playwright install webkit

# Install system dependencies
npx playwright install-deps

# Install browsers with dependencies
npx playwright install --with-deps

# Uninstall browsers
npx playwright uninstall

# Check Playwright version
npx playwright --version

# Get installation info
npx playwright install --dry-run
```

Trace & Debug Commands:

```
# Record trace
npx playwright test --trace on

# Record trace on failure
npx playwright test --trace on-first-retry
npx playwright test --trace retain-on-failure

# Show trace
npx playwright show-trace trace.zip

# Screenshot on failure
npx playwright test --screenshot on
npx playwright test --screenshot only-on-failure

# Video recording
npx playwright test --video on
npx playwright test --video retain-on-failure
```

26. How do you configure different environments using NPM scripts?

Answer:

```
// package.json
{
  "scripts": {
    "test:dev": "BASE_URL=https://dev.example.com playwright test",
    "test:staging": "BASE_URL=https://staging.example.com playwright test",
    "test:prod": "BASE_URL=https://example.com playwright test",

    "test:dev:ui": "BASE_URL=https://dev.example.com playwright test --ui",
    "test:staging:smoke": "BASE_URL=https://staging.example.com playwright test --grep @smoke",

    "test:ci": "playwright test --reporter=blob,html",
    "test:ci:chrome": "playwright test --project=chromium --reporter=blob",

    "test:mobile": "playwright test --config=playwright.mobile.config.ts",
    "test:desktop": "playwright test --config=playwright.desktop.config.ts",

    "test:parallel:dev": "BASE_URL=https://dev.example.com playwright test --workers=4",
    "test:serial:prod": "BASE_URL=https://example.com playwright test --workers=1"
  }
}
```

```
// playwright.config.ts
import { defineConfig } from '@playwright/test';

export default defineConfig({
  use: {
    baseURL: process.env.BASE_URL || 'http://localhost:3000',

    extraHTTPHeaders: {
      'Authorization': process.env.AUTH_TOKEN || ''
    }
  }
});
```

```
# .env.dev
BASE_URL=https://dev.example.com
API_KEY=dev-key-123

# .env.staging
BASE_URL=https://staging.example.com
API_KEY=staging-key-456

# Usage with dotenv
npm run test:dev
npm run test:staging
```

techSmartHire

27. How do you use Playwright CLI for test filtering and execution control?

Answer:

```
# Filter by test title
npx playwright test -g "login"
npx playwright test --grep "login"

# Multiple grep patterns
npx playwright test --grep "login|signup"

# Exclude tests
npx playwright test --grep-invert "slow"
npx playwright test -gv "slow"

# Combine grep and invert
npx playwright test --grep "@smoke" --grep-invert "@skip"

# Filter by file path
npx playwright test tests/auth/
npx playwright test tests/auth/login.spec.ts

# Filter by project
npx playwright test --project=chromium --project=firefox

# Global timeout
npx playwright test --global-timeout=3600000

# Test timeout
npx playwright test --timeout=30000

# Repeat tests
npx playwright test --repeat-each=3

# Forbid only (CI mode)
npx playwright test --forbid-only

# Pass with no tests
npx playwright test --pass-with-no-tests

# Ignore snapshots
npx playwright test --ignore-snapshots
```

28. How do you configure and use Playwright reporters via terminal?

Answer:

```
# Single reporter
npx playwright test --reporter=list
npx playwright test --reporter=line
npx playwright test --reporter=dot
npx playwright test --reporter=html
npx playwright test --reporter=json
npx playwright test --reporter=junit
```

tSH

```
npx playwright test --reporter=github

# Multiple reporters
npx playwright test --reporter=list,html
npx playwright test --reporter=list,json,html

# Reporter with custom output
npx playwright test --reporter=json --reporter-output=results.json
npx playwright test --reporter=junit --reporter-output=results.xml
npx playwright test --reporter=html --reporter-output=custom-report

# CI/CD blob reporter
npx playwright test --reporter=blob
npx playwright merge-reports --reporter=html ./blob-report

# Open report
npx playwright show-report
npx playwright show-report custom-report
```

Configuration in playwright.config.ts:

```
export default defineConfig({
  reporter: [
    ['list'],
    ['html', { outputFolder: 'playwright-report', open: 'never' }],
    ['json', { outputFile: 'test-results.json' }],
    ['junit', { outputFile: 'junit-results.xml' }]
  ]
});
```

29. What are Playwright CLI commands for debugging and inspection?

Answer:

```
# Debug mode (opens Inspector)
npx playwright test --debug

# Debug specific test
npx playwright test tests/login.spec.ts --debug

# Debug from specific line
npx playwright test tests/login.spec.ts:25 --debug

# Headed mode with slow motion
npx playwright test --headed --slow-mo=1000

# Browser console logs
PWDEBUG=console npx playwright test

# Verbose output
npx playwright test --verbose

# Inspector
npx playwright inspector

# Open trace viewer
npx playwright show-trace trace.zip

# Screenshot comparison
npx playwright test --update-snapshots
```

```
# Show browser
npx playwright open https://example.com
npx playwright open --browser=firefox https://example.com
npx playwright open --device="iPhone 12" https://example.com

# PDF generation
npx playwright pdf https://example.com output.pdf

# Screenshot
npx playwright screenshot https://example.com screenshot.png
```

30. How do you manage Playwright browser installations and updates?

Answer:

```
# Check current version
npx playwright --version
npm list @playwright/test

# Install all browsers
npx playwright install

# Install specific browser
npx playwright install chromium
npx playwright install firefox
npx playwright install webkit
npx playwright install msedge

# Install with system dependencies
npx playwright install --with-deps
npx playwright install chromium --with-deps

# Install only system dependencies
npx playwright install-deps
npx playwright install-deps chromium

# Dry run (see what would be installed)
npx playwright install --dry-run

# Force reinstall
npx playwright install --force

# Uninstall browsers
npx playwright uninstall
npx playwright uninstall chromium

# Update Playwright
npm install -D @playwright/test@latest
npx playwright install

# Check installed browsers
ls ~/.cache/ms-playwright/  # Linux/Mac
dir %USERPROFILE%\AppData\Local\ms-playwright\  # Windows

# Set custom browser path
PLAYWRIGHT_BROWSERS_PATH=/custom/path npx playwright install

# Skip browser download during npm install
PLAYWRIGHT_SKIP_BROWSER_DOWNLOAD=1 npm install
```

techSmartHire

package.json scripts for installation:

```
{
  "scripts": {
    "postinstall": "playwright install chromium",
    "install:all": "playwright install",
    "install:chrome": "playwright install chromium",
    "install:deps": "playwright install-deps",
    "update:playwright": "npm install -D @playwright/test@latest && playwright install"
  }
}
```

Conclusion

These 30 Playwright TypeScript/JavaScript interview questions cover comprehensive topics including:

- Core Playwright concepts and architecture
- Locator strategies and best practices
- API testing and network interception
- Authentication and session management
- Visual regression testing
- Page Object Model implementation
- Codegen and UI Mode features
- NPM scripts and package.json configuration
- Terminal commands and CLI usage
- Browser installation and management
- Debugging techniques and optimization strategies

Practice these concepts hands-on to master Playwright testing and ace your interviews!

tSH

techSmartHire

Looking for QA job opportunities?

Take a **skill-based assessment** and showcase your skills to 100+ recruiters from the techSmartHire Platform

👉 Login to **techsmarthire.com**

&

Get Hired Smarter.

tSH