Master Playwright Automation: Click, Type, and Hover Interactions

Why Mastering Basic Interactions Matters

Before we dive into the code, let's understand why these seemingly simple interactions deserve our attention:

- They simulate the most common **user behaviors** on web applications
- Proper implementation ensures **test reliability**
- Understanding nuances prevents **flaky tests**
- These actions are the building blocks for **complex test scenarios**

Clicking Elements in Playwright

Clicking is the most common interaction in web testing. Playwright offers several clicking options to handle various scenarios:

```
// Basic click
await page.click('.submit-button');

// Click with modifiers (Ctrl, Shift, Alt)
await page.click('.item', { modifiers: ['Shift'] });

// Double click
await page.dblclick('#item');

// Right click
await page.click('.context-menu-item', { button: 'right' });

// Force click (bypasses actionability checks)
await page.click('.covered-element', { force: true });

// Click at specific position within element
await page.click('.canvas', { position: { x: 10, y: 20 } });

// Click and wait for navigation
await Promise.all([
  page.waitForNavigation(),
  page.click('a.nav-link')
]);

// Click and wait for network request
await Promise.all([
  page.waitForRequest(/api\/data/),
  page.click('#fetch-data')
]);
```

Best Practices for Clicking

1. **Avoid force: true** unless absolutely necessary
2. **Use precise selectors** instead of complex XPaths
3. **Wait for navigations** when clicks cause page changes
4. **Consider click positions** for canvas or custom elements
5. **Handle expected dialogs** before clicks that trigger them

Typing Text with Playwright

Typing (or filling) inputs is another crucial interaction:

```
// Basic fill (clears field and types text)
await page.fill('#username', 'testuser');

// Type character by character (simulates real typing)
await page.type('#search', 'search term', { delay: 100 });

// Press individual keys
await page.press('#search', 'Enter');

// Key combinations
await page.press('body', 'Control+A');
await page.press('body', 'Control+c');

// Fill a form efficiently
await page.fill('#username', 'testuser');
await page.fill('#password', 'securepass');
await page.selectOption('#country', 'US');
await page.check('#agree-terms');
await page.click('#submit');

// Clear input
await page.fill('#search', '');
```

Input Handling Techniques

```
// Handle date inputs
await page.fill('input[type="date"]', '2023-10-15');

// Handle file inputs
await page.setInputFiles('input[type="file"]', 'path/to/file.jpg');
// Or multiple files
await page.setInputFiles('input[type="file"]', [
  'path/to/file1.jpg',
  'path/to/file2.jpg'
]);

// Handle contentEditable elements
await page.fill('div[contenteditable]', 'Formatted text');
```

Best Practices for Text Input

1. Use fill() for most cases (it's more reliable than type())
2. Use type() when testing character-by-character input is important
3. Clear fields explicitly if needed before typing
4. Validate input state after filling when testing validation
5. Consider special input types (date, file, etc.)

Hover Interactions in Playwright

Hovering over elements is essential for testing menus, tooltips, and hover states:

```
// Basic hover
await page.hover('.dropdown-toggle');

// Hover and then click child element
await page.hover('.dropdown-toggle');
await page.click('.dropdown-menu .item');

// Hover with position
await page.hover('.image-map', { position: { x: 100, y: 150 } });

// Force hover (bypass actionability checks)
await page.hover('.obscured-element', { force: true });

// Verify hover states
await page.hover('.tooltip-trigger');
await expect(page.locator('.tooltip')).toBeVisible();
```

Best Practices for Hover Actions

1. **Check for hover animations completion before subsequent actions**
2. **Verify hover effects as part of your test assertions**
3. **Combine with other actions for interaction chains**
4. **Handle timing-sensitive hover menus carefully**
5. **Consider mobile testing limitations where hover doesn't exist**

Advanced Interaction Patterns

Let's combine these actions for real-world scenarios:

Drag and Drop

```
// Method 1: Using mouse actions
await page.mouse.move(100, 100);
await page.mouse.down();
await page.mouse.move(200, 200);
await page.mouse.up();
```

```
// Method 2: Using drag-and-drop helper function
async function dragAndDrop(page, sourceSelector, targetSelector) {
  const source = await page.$(sourceSelector);
  const target = await page.$(targetSelector);

  const sourceBox = await source.boundingBox();
  const targetBox = await target.boundingBox();

  await page.mouse.move(
    sourceBox.x + sourceBox.width / 2,
    sourceBox.y + sourceBox.height / 2
  );
  await page.mouse.down();
  await page.mouse.move(
    targetBox.x + targetBox.width / 2,
    targetBox.y + targetBox.height / 2
  );
  await page.mouse.up();
}
```

Handling Complex UI Components

```
// Interacting with rich text editors
await page.click('.editor');
await page.keyboard.type('Hello World');
await page.keyboard.press('Control+b'); // Bold text
await page.keyboard.type(' in bold');

// Handling sliders
async function setSliderValue(page, sliderSelector, value) {
  const slider = await page.$(sliderSelector);
  const box = await slider.boundingBox();
  const min = parseFloat(await slider.getAttribute('min')) || 0;
  const max = parseFloat(await slider.getAttribute('max')) || 100;

  const targetX = box.x + box.width * ((value - min) / (max - min));
  await page.mouse.move(targetX, box.y + box.height / 2);
  await page.mouse.down();
  await page.mouse.up();
}
```

Real-World Example: E-commerce Product Selection

Let's put it all together with a real-world e-commerce test:

```
async function selectAndAddProductToCart(page) {
  // Navigate to products page
  await page.goto('https://example-shop.com/products');

  // Hover over product category menu
  await page.hover('.category-menu');

  // Click on specific category
```

```
  await page.click('.category-menu .electronics');

  // Type in search field
  await page.fill('#search-input', 'smartphone');
  await page.press('#search-input', 'Enter');

  // Wait for search results
  await page.waitForSelector('.product-grid');

  // Hover over a product to see quick view
  await page.hover('.product-card:first-child');

  // Click quick view button that appears on hover
  await page.click('.product-card:first-child .quick-view');

  // Select product options
  await page.selectOption('.color-select', 'black');
  await page.click('.storage-option[data-value="128gb"]');

  // Add to cart
  await Promise.all([
    page.waitForResponse(res => res.url().includes('/api/cart') &&
res.status() === 200),
    page.click('#add-to-cart')
  ]);

  // Verify cart update
  await expect(page.locator('.cart-count')).toContainText('1');
}
```