Form Elements Automation - Complete Learning Guide

> A comprehensive guide covering Playwright methods, concepts, Selenium comparison, and interview preparation for form element automation.

Table of Contents

1. Overview
2. Form Element Types
3. Playwright Methods Used
4. Key Concepts
5. Selenium vs Playwright Comparison
6. Code Examples
7. Interview Questions & Answers

1. Overview

Form elements are the building blocks of user input on web applications. Automating form interactions is one of the most common testing scenarios. This guide covers all form element types and their automation techniques.

Form Elements Covered

Element	HTML Tag	Playwright ID
Email Input	`<input type="email">`	`#emailInput`
Number Input	`<input type="number">`	`#numberInput`
Radio Buttons	`<input type="radio">`	`#r1` , `#r2` , `#r3`
Checkboxes	`<input type="checkbox">`	`#option1` , `#option2` , `#option3`
Single Dropdown	Custom Combobox	`#selectOption`
Multi-Select	`<select multiple>`	`#multiSelect`
Textarea	`<textarea>`	`#textareaInput`

Element	HTML Tag	Playwright ID
Buttons	`<button>`	Reset, Submit

2. Form Element Types

2.1 Text Input Fields

```
// Email input - accepts text with email validation
readonly emailInput: Locator = page.locator('#emailInput');

// Number input - accepts only numeric values
readonly numberInput: Locator = page.locator('#numberInput');
```

2.2 Radio Buttons

- **Behavior**: Only ONE can be selected at a time in a group
- **Use Case**: Single choice from multiple options

```
readonly radioOption1: Locator = page.locator('#r1');
readonly radioOption2: Locator = page.locator('#r2');
readonly radioOption3: Locator = page.locator('#r3');
```

2.3 Checkboxes

- **Behavior**: Multiple can be selected independently
- **Use Case**: Multiple selections from options

```
readonly checkbox1: Locator = page.locator('#option1');
readonly checkbox2: Locator = page.locator('#option2');
readonly checkbox3: Locator = page.locator('#option3');
```

2.4 Dropdowns (Select)

Native Select - Uses standard `<select>` HTML element
Custom Combobox - Built with JavaScript/React (Radix UI)

```
// Custom combobox (requires click interaction)
readonly singleSelectDropdown: Locator = page.locator('#selectOption');

// Native multi-select
readonly multiSelect: Locator = page.locator('#multiSelect');
```

2.5 Textarea

- **Behavior**: Multi-line text input
- **Use Case**: Comments, descriptions, long text

```
readonly textareaInput: Locator = page.locator('#textareaInput');
```

3. Playwright Methods Used

3.1 Input Methods

Method	Description	Example
`fill(value)`	Clears and types text	`await input.fill('test@email.com')`
`clear()`	Clears input field	`await input.clear()`
`type(text)`	Types text character by character	`await input.type('hello')`
`inputValue()`	Gets current value	`await input.inputValue()`

```
// Fill method - Most commonly used
async fillEmail(email: string): Promise<void> {
    await this.emailInput.fill(email);
}

// Get current value
async getTextareaValue(): Promise<string> {
    return await this.textareaInput.inputValue();
}
```

3.2 Radio & Checkbox Methods

Method	Description	Example
`check()`	Checks/selects the element	`await checkbox.check()`

Method	Description	Example
`uncheck()`	Unchecks the element	`await checkbox.uncheck()`
`isChecked()`	Returns boolean state	`await radio.isChecked()`
`setChecked(value)`	Sets check state	`await checkbox.setChecked(true)`

```
// Check a checkbox
async checkCheckbox(option: 1 | 2 | 3): Promise<void> {
    const checkbox = this.getCheckboxLocator(option);
    await checkbox.check();
}

// Verify checkbox state
async isCheckboxChecked(option: 1 | 2 | 3): Promise<boolean> {
    const checkbox = this.getCheckboxLocator(option);
    return await checkbox.isChecked();
}

// Select radio button
async selectRadioOption(option: 1 | 2 | 3): Promise<void> {
    switch (option) {
        case 1: await this.radioOption1.check(); break;
        case 2: await this.radioOption2.check(); break;
        case 3: await this.radioOption3.check(); break;
    }
}
```

3.3 Dropdown Methods

Method	Description	Example
`selectOption(value)`	Selects by value	`await select.selectOption('opt1')`
`selectOption({label})`	Selects by visible text	`await select.selectOption({label: 'Item 1'})`
`selectOption([values])`	Multi-select	`await select.selectOption(['a', 'b'])`

```
// Native select - use selectOption
await this.multiSelect.selectOption({ label: 'Multi Option 1' });

// Custom combobox - use click interaction
async selectDropdownOption(optionText: string): Promise<void> {
    await this.singleSelectDropdown.click();
    await this.page.waitForTimeout(200);
    const option = this.page.locator(`[role="option"]:has-text("${optionText}")`);
    await option.click();
}
```

3.4 Click Methods

Method	Description	Example
`click()`	Standard click	`await button.click()`
`dblclick()`	Double click	`await element.dblclick()`
`click({button})`	Right/middle click	`await el.click({button: 'right'})`

```
async clickSubmit(): Promise<void> {
    await this.submitButton.click();
}

async clickReset(): Promise<void> {
    await this.resetButton.click();
}
```

3.5 Keyboard Methods

Method	Description	Example
`press(key)`	Presses a key	`await page.keyboard.press('Tab')`
`type(text)`	Types text	`await page.keyboard.type('hello')`
`down(key)`	Holds key down	`await page.keyboard.down('Shift')`
`up(key)`	Releases key	`await page.keyboard.up('Shift')`

```
// Tab navigation for accessibility testing
async getTabOrder(): Promise<string[]> {
    const elementOrder: string[] = [];
    await this.emailInput.focus();

    for (let i = 0; i < 15; i++) {
        await this.page.keyboard.press('Tab');
        const focusedId = await this.page.evaluate(
            () => document.activeElement?.id
        );
        if (focusedId) elementOrder.push(focusedId);
    }
    return elementOrder;
}
```

3.6 Assertion Methods

Method	Description	Example
`toBeVisible()`	Element is visible	`await expect(el).toBeVisible()`
`toBeEnabled()`	Element is enabled	`await expect(el).toBeEnabled()`
`toBeChecked()`	Checkbox/radio is checked	`await expect(cb).toBeChecked()`
`toHaveValue(val)`	Input has value	`await expect(input).toHaveValue('test')`
`toContain(text)`	Contains text	`expect(str).toContain('hello')`

4. Key Concepts

4.1 Page Object Model (POM)

What: Design pattern that creates a class for each page/component with locators and methods.

Why Use It:

- ✅ Maintainability - Change locator in one place
- ✅ Reusability - Methods can be reused across tests
- ✅ Readability - Tests read like user stories

```
export class FormElementsPage {
    readonly page: Page;
    readonly emailInput: Locator;

    constructor(page: Page) {
        this.page = page;
        this.emailInput = page.locator('#emailInput');
    }

    async fillEmail(email: string): Promise<void> {
        await this.emailInput.fill(email);
    }
}
```

4.2 Locator Strategies

Strategy	Playwright	Priority
ID	`page.locator('#id')`	⭐⭐⭐ Best
CSS Selector	`page.locator('.class')`	⭐⭐ Good
Text	`page.locator('text=Submit')`	⭐⭐ Good
Role	`page.locator('[role="button"]')`	⭐⭐ Good
XPath	`page.locator('//div[@id="x"]')`	⭐ Avoid

4.3 Custom Component Handling

Modern web apps use custom components (React, Angular). They don't use native HTML elements:

```
// Native <select> - use selectOption
await nativeSelect.selectOption('value');

// Custom Radix UI combobox - use click interaction
await customDropdown.click();
await page.locator('[role="option"]:has-text("Item 1")').click();
```

4.4 Async/Await Pattern

All Playwright operations are asynchronous:

```
// ✅ Correct - using await
async fillForm(): Promise<void> {
    await this.emailInput.fill('test@example.com');
    await this.submitButton.click();
}

// ❌ Wrong - missing await
async fillForm(): Promise<void> {
    this.emailInput.fill('test@example.com'); // Won't wait!
}
```

4.5 Auto-Waiting

Playwright automatically waits for elements before interacting:

- Waits for element to be attached to DOM
- Waits for element to be visible
- Waits for element to be enabled
- Waits for element to stop moving

```
// No explicit wait needed - Playwright auto-waits
await this.submitButton.click();
```

5. Selenium vs Playwright Comparison

5.1 Method Comparison Table

Action	Playwright (TypeScript)	Selenium (Java)
Find Element	`page.locator('#id')`	`driver.findElement(By.id("id"))`
Fill Input	`await input.fill('text')`	`element.clear(); element.sendKeys("text");`
Click	`await button.click()`	`element.click()`
Get Text	`await el.textContent()`	`element.getText()`
Get Value	`await input.inputValue()`	`element.getAttribute("value")`
Check Checkbox	`await checkbox.check()`	`if(!cb.isSelected()) cb.click()`

Action	Playwright (TypeScript)	Selenium (Java)
Uncheck	`await checkbox.uncheck()`	`if(cb.isSelected()) cb.click()`
Is Checked	`await radio.isChecked()`	`element.isSelected()`
Select Dropdown	`await select.selectOption('val')`	`new Select(el).selectByValue("val")`
Select by Text	`await select.selectOption({label: 'text'})`	`new Select(el).selectByVisibleText("text")`
Multi-Select	`await select.selectOption(['a','b'])`	Loop with `selectByValue()`
Press Key	`await page.keyboard.press('Tab')`	`actions.sendKeys(Keys.TAB).perform()`
Wait	Auto-waiting built-in	`WebDriverWait + ExpectedConditions`
Assertions	`expect(el).toBeVisible()`	`Assert.assertTrue(el.isDisplayed())`

5.2 Code Comparison Examples

Fill Input Field

Playwright (TypeScript)

```
await page.locator('#emailInput').fill('test@example.com');
```

Selenium (Java)

```
WebElement email = driver.findElement(By.id("emailInput"));
email.clear();
email.sendKeys("test@example.com");
```

Select Radio Button

Playwright (TypeScript)

```
await page.locator('#r1').check();
const isSelected = await page.locator('#r1').isChecked();
```

Selenium (Java)

```
WebElement radio = driver.findElement(By.id("r1"));
radio.click();
boolean isSelected = radio.isSelected();
```

Handle Checkbox

Playwright (TypeScript)

```
// Check
await page.locator('#option1').check();

// Uncheck
await page.locator('#option1').uncheck();

// Toggle based on condition
await page.locator('#option1').setChecked(true);
```

Selenium (Java)

```
WebElement checkbox = driver.findElement(By.id("option1"));

// Check
if (!checkbox.isSelected()) {
    checkbox.click();
}

// Uncheck
if (checkbox.isSelected()) {
    checkbox.click();
}
```

Select from Dropdown

Playwright (TypeScript)

```
// By value
await page.locator('#selectOption').selectOption('option1');

// By label
await page.locator('#selectOption').selectOption({ label: 'Item 1' });

// Multi-select
await page.locator('#multiSelect').selectOption(['opt1', 'opt2']);
```

Selenium (Java)

```
Select dropdown = new Select(driver.findElement(By.id("selectOption")));

// By value
dropdown.selectByValue("option1");

// By visible text
dropdown.selectByVisibleText("Item 1");

// Multi-select
dropdown.selectByValue("opt1");
dropdown.selectByValue("opt2");
```

Handle Custom Dropdown (Non-native)

Playwright (TypeScript)

```
// Click to open
await page.locator('#selectOption').click();

// Select option by text
await page.locator('[role="option"]:has-text("Item 1")').click();
```

Selenium (Java)

```
// Click to open
driver.findElement(By.id("selectOption")).click();

// Wait and select option
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
wait.until(ExpectedConditions.visibilityOfElementLocated(
    By.xpath("//div[@role='option' and contains(text(),'Item 1')]")
)).click();
```

5.3 Key Differences Summary

Feature	Playwright	Selenium
Auto-waiting	✅ Built-in	❌ Manual waits required
Language	JS/TS, Python, C#, Java	Java, Python, C#, Ruby, JS
Speed	Faster	Slower
Check/Uncheck	Dedicated methods	Click with condition
Assertions	Built-in `expect`	External libraries (TestNG/JUnit)

Feature	Playwright	Selenium
Network Control	✅ Built-in mocking	❌ Requires proxy tools
Screenshots	✅ Automatic on failure	Manual configuration
Parallel Execution	✅ Native support	Grid/TestNG setup

6. Code Examples

6.1 Complete Form Submission Test

```
test('Complete form submission', async ({ page }) => {
    const formPage = new FormElementsPage(page);
    await formPage.navigate();

    // Fill all fields
    await formPage.fillEmail('test@example.com');
    await formPage.fillNumber('42');
    await formPage.selectRadioOption(2);
    await formPage.checkCheckbox(1);
    await formPage.checkCheckbox(2);
    await formPage.selectDropdownOption('Item 2');
    await formPage.fillTextarea('Test message');

    // Submit
    await formPage.clickSubmit();
});
```

6.2 Reset Validation Test

```
test('Reset button clears form', async ({ page }) => {
    const formPage = new FormElementsPage(page);
    await formPage.navigate();

    // Fill form
    await formPage.fillEmail('test@example.com');
    await formPage.fillNumber('100');

    // Reset
    await formPage.clickReset();

    // Verify cleared
    const email = await formPage.emailInput.inputValue();
    expect(email).toBe('');
});
```

6.3 Accessibility Tab Order Test

```
test('Tab navigation accessibility', async ({ page }) => {
    const formPage = new FormElementsPage(page);
    await formPage.navigate();

    const tabOrder = await formPage.getTabOrder();

    // Verify logical order
    expect(tabOrder).toContain('emailInput');
    expect(tabOrder).toContain('numberInput');
    expect(tabOrder).toContain('textareaInput');
});
```

7. Interview Questions & Answers

Q1: How do you handle input fields in Playwright?

Answer: Playwright provides the `fill()` method which clears the existing content and types the new value. For character-by-character typing, use `type()`. To get the current value, use `inputValue()`.

```
// Fill clears and types
await page.locator('#email').fill('test@example.com');

// Get current value
const value = await page.locator('#email').inputValue();
```

Q2: What is the difference between `check()` and `click()` for checkboxes?

Answer:

- `check()` - Only checks if unchecked, does nothing if already checked (idempotent)
- `click()` - Toggles the state regardless of current state
- `uncheck()` - Only unchecks if checked, does nothing if already unchecked
- `setChecked(boolean)` - Sets to specific state

```
await checkbox.check();     // Ensures checked
await checkbox.uncheck();   // Ensures unchecked
await checkbox.click();     // Toggles state
await checkbox.setChecked(true);  // Sets to checked
```

Q3: How do you handle custom dropdowns (non-native select)?

Answer: Custom dropdowns built with frameworks like React/Radix UI require click-based interaction:

1. Click to open the dropdown
2. Wait for options to appear
3. Click on the desired option

```
async selectDropdownOption(optionText: string): Promise<void> {
    await this.dropdown.click();
    await this.page.waitForTimeout(200);
    await this.page.locator(`[role="option"]:has-text("${optionText}")`).click();
}
```

Q4: What is Page Object Model and why is it important?

Answer: POM is a design pattern where each web page is represented as a class. It contains:

- Locators as class properties
- Page actions as methods

Benefits:

- **Maintainability**: Update locators in one place
- **Reusability**: Use same page object across multiple tests
- **Readability**: Tests read like user stories
- **Separation of Concerns**: Locators separate from test logic

Q5: How does Playwright handle waiting compared to Selenium?

Answer:

- **Playwright**: Auto-waiting is built-in. It automatically waits for elements to be actionable (visible, enabled, stable)
- **Selenium**: Requires explicit waits using `WebDriverWait` and `ExpectedConditions`

```
// Playwright - no explicit wait needed
await button.click(); // Auto-waits

// Selenium equivalent
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
wait.until(ExpectedConditions.elementToBeClickable(by)).click();
```

Q6: How do you verify radio button selection?

Answer: Use `isChecked()` method to verify selection state:

```
await page.locator('#r1').check();

// Verify selection
expect(await page.locator('#r1').isChecked()).toBe(true);
expect(await page.locator('#r2').isChecked()).toBe(false);
```

Q7: How do you select multiple options in a multi-select dropdown?

Answer: Pass an array to `selectOption()`:

```
// By values
await page.locator('#multiSelect').selectOption(['opt1', 'opt2', 'opt3']);

// By labels
await page.locator('#multiSelect').selectOption([
    { label: 'Option 1' },
    { label: 'Option 3' }
]);
```

Q8: What are the key locator strategies in Playwright?

Answer (Priority order):

1. **ID**: `page.locator('#elementId')` - Most reliable
2. **Test ID**: `page.getByTestId('submit-btn')` - Recommended
3. **Role**: `page.getByRole('button', { name: 'Submit' })` - Semantic
4. **Text**: `page.getByText('Submit')` - User-facing
5. **CSS**: `page.locator('.class-name')` - Flexible
6. **XPath**: `page.locator('//div[@id="x"]')` - Last resort

Q9: How do you test keyboard navigation (accessibility)?

Answer: Use `keyboard.press()` to simulate Tab navigation:

```
async getTabOrder(): Promise<string[]> {
    const order: string[] = [];
    await this.firstElement.focus();

    for (let i = 0; i < 10; i++) {
        await this.page.keyboard.press('Tab');
        const focusedId = await this.page.evaluate(
            () => document.activeElement?.id
        );
        if (focusedId) order.push(focusedId);
    }
    return order;
}
```

Q10: How do you handle form reset validation?

Answer: Fill the form, click reset, and verify all fields return to default:

```
test('Reset clears form', async ({ page }) => {
    // Fill form
    await formPage.fillEmail('test@example.com');
    await formPage.checkCheckbox(1);

    // Reset
    await formPage.clickReset();

    // Verify defaults
    expect(await formPage.emailInput.inputValue()).toBe('');
    expect(await formPage.isCheckboxChecked(1)).toBe(false);
});
```

Q11: What is the difference between `fill()` and `type()` in Playwright?

Answer:

- `fill()` : Clears existing content first, then sets the value instantly. Best for form filling.
- `type()` : Types character by character, triggers keyboard events. Best for testing autocomplete or character-by-character validation.

```
await input.fill('instant text');  // Fast, clears first
await input.type('slow text');     // Character by character
```

Q12: How do you handle boundary testing for form fields?

Answer: Test edge cases for each input type:

```
test('Boundary testing', async ({ page }) => {
    // Empty value
    await input.fill('');

    // Minimum value
    await numberInput.fill('0');

    // Negative value
    await numberInput.fill('-1');

    // Maximum value
    await numberInput.fill('999999999');

    // Special characters
    await textInput.fill('!@#$%^&*()');

    // Very long text
    await textarea.fill('A'.repeat(10000));
});
```

Q13: What are the advantages of Playwright over Selenium for form automation?

Answer:

1. **Auto-waiting**: No explicit waits needed
2. **Dedicated methods**: `check()`, `uncheck()`, `isChecked()` vs click with conditions
3. **Built-in assertions**: `expect(el).toBeVisible()`
4. **Faster execution**: Direct browser protocol communication
5. **Better TypeScript support**: Full type safety
6. **Network mocking**: Built-in request interception
7. **Trace viewer**: Visual debugging with screenshots

Q14: How do you verify all checkboxes are independent (not radio buttons)?

Answer: Select multiple checkboxes and verify all remain selected:

```
test('Checkboxes are independent', async ({ page }) => {
    // Check all
    await formPage.checkCheckbox(1);
    await formPage.checkCheckbox(2);
    await formPage.checkCheckbox(3);

    // Verify ALL are checked (unlike radio buttons)
    expect(await formPage.isCheckboxChecked(1)).toBe(true);
    expect(await formPage.isCheckboxChecked(2)).toBe(true);
    expect(await formPage.isCheckboxChecked(3)).toBe(true);
});
```

Q15: How do you get the selected value from a dropdown?

Answer:

```
// For native select - use inputValue()
const selectedValue = await page.locator('#dropdown').inputValue();

// For custom dropdown - get text content
const selectedText = await page.locator('#dropdown').textContent();
```

Quick Reference Card

```
// INPUT FIELDS
await input.fill('text');           // Fill input
await input.clear();                // Clear input
await input.inputValue();           // Get value

// CHECKBOXES & RADIO
await element.check();              // Check
await element.uncheck();            // Uncheck
await element.isChecked();          // Is checked?
await element.setChecked(true);     // Set state

// DROPDOWNS
await select.selectOption('value');              // By value
await select.selectOption({ label: 'Text' });    // By text
await select.selectOption(['a', 'b']);           // Multi-select

// KEYBOARD
await page.keyboard.press('Tab');
await page.keyboard.press('Enter');
await page.keyboard.type('text');

// ASSERTIONS
await expect(element).toBeVisible();
await expect(element).toBeEnabled();
await expect(element).toBeChecked();
await expect(input).toHaveValue('expected');
```

Document created for the Form Elements Automation Challenge
Framework: Playwright + TypeScript
Date: January 2026