**Types of Testing:**

**1. Unit Test**

**Tests a single function or component in isolation.**

* Fast, reliable, and run often
* Focused on logic or small UI parts (e.g., a button click updates state)
* Tools: Jest, Vitest, Mocha, Jasmine

**Example**:

// button.test.ts

import { add } from './math';

test('adds two numbers', () => {

expect(add(2, 3)).toBe(5);

});

**2. Integration Test**

**Tests multiple parts working together.**

* Ensures components/modules interact correctly
* More realistic than unit tests, but still doesn’t test full app
* Tools: Jest + React Testing Library, Cypress, Playwright

**Example**:

// Integration: Form submits and calls API

test('form submits and resets', () => {

render(<LoginForm />);

fireEvent.change(screen.getByLabelText('Email'), { target: { value: 'test@example.com' }});

fireEvent.click(screen.getByText('Submit'));

expect(mockApiCall).toHaveBeenCalled();

});

**3. End-to-End (E2E) Test**

**Tests the entire app flow from the user's perspective.**

* Tests real browser + real backend or mocks
* Simulates user behavior (clicking, typing, navigation)
* Slower, but critical for confidence
* Tools: Playwright, Cypress, Selenium

**Example**:

// e2e/login.spec.ts

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

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

await page.fill('#email', 'user@example.com');

await page.fill('#password', 'password123');

await page.click('button[type=submit]');

await expect(page).toHaveURL('https://yourapp.com/dashboard');

});

**4. Visual Regression Test**

**Tests how the UI looks — pixel by pixel.**

* Catches unexpected visual changes
* Takes screenshots and compares with a baseline
* Tools: Playwright + Percy, Chromatic, Loki, Storybook Test Runner

**Example**:

// Visual test using Playwright

test('Button looks correct', async ({ page }) => {

await page.goto('http://localhost:6006/?path=/story/button--primary');

expect(await page.screenshot()).toMatchSnapshot('button-primary.png');

});

**Comparison**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **What It Tests** | **Tools** | **Speed** | **Confidence** |
| **Unit Test** | One function/component | Jest, Vitest | Fast | 🟡 Low |
| **Integration Test** | Component + its dependencies | React Testing Library | Med | 🟡 Medium |
| **E2E Test** | Full user flow (browser + backend) | Playwright, Cypress | Slow | 🟢 High |
| **Visual Regression** | UI appearance (pixel differences) | Chromatic, Percy, Playwright | Slow | 🟢 High |

*Use Playwright with Storybook to do visual regression testing or interaction testing for UI components rendered in isolation.*

**Use Case**

You're testing individual components in your Storybook using Playwright—either to:

* Run end-to-end UI tests
* Do visual regression testing
* Simulate user interactions on components

**Setup**

**1. Install dependencies**

Install Storybook and Playwright your project:

# Storybook (if not already installed)

npx sb init

# Playwright

npm install --save-dev playwright @storybook/test-runner

**2. Configure Storybook for Playwright Testing**

Create a file: .storybook/test-runner.js or storybook.config.js if using newer versions.

// .storybook/test-runner.js

const { getStoryContext } = require('@storybook/test-runner');

module.exports = {

async preRender(page, context) {

// Optional: do something before each story

},

async postRender(page, context) {

const storyContext = await getStoryContext(page, context);

// Optional: use storyContext to assert something

},

};

**3. Run Storybook**

Start Storybook locally on port 6006 (or whatever you use):

npm run storybook

**4. Run Playwright Tests Against Storybook**

You can run tests using the test-runner:

npx test-storybook

Or write your own **Playwright test** using the Storybook URL

// tests/button.spec.ts

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

test('button primary renders correctly', async ({ page }) => {

await page.goto('http://localhost:6006/?path=/story/components-button--primary');

const button = await page.getByRole('button');

await expect(button).toHaveText('Primary');

});

Then run:

npx playwright test

**Next Step: Visual Regression with @storybook/test-runner**

Doing screenshot comparisons:

npm install --save-dev @storybook/test-runner-playwright

Then configure it to take snapshots and compare them.

// .storybook/test-runner.js

const { injectAxe, checkA11y } = require('axe-playwright');

module.exports = {

async postRender(page, context) {

await injectAxe(page);

await checkA11y(page, null, {

detailedReport: true,

detailedReportOptions: { html: true },

});

},

};

**Resources**

* [Playwright docs](https://playwright.dev/)