

Playwright with Java: From Zero to Real Tests

You already know Java. This guide teaches only what is new: Playwright words, objects and methods, and how to use them to test a real web page. Every code example is Java and was run successfully.

IMPORTANT

Red boxes and rows: learn these first. They are used in almost every test and asked in almost every interview.

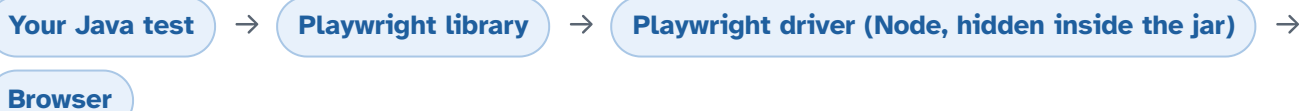
GOOD TO KNOW

Blue boxes and rows: useful, but you can learn them later or only when needed.

1. What is Playwright and how does it work?

Playwright is a Java library, made by Microsoft, that controls a real browser (Chromium, Firefox or WebKit). Your Java code says "open this page", "fill this box", "click that button". Then your code checks the result.

Compared with Selenium, Playwright **waits automatically** before every action, so you write far fewer waits. It also has built-in screenshots, video, tracing and network control.



You never see the middle part. When you run your Java code, Playwright starts it by itself.

The five objects you must understand

IMPORTANT Object hierarchy

OBJECT	WHAT IT IS	REAL LIFE PICTURE
Playwright	The starting point. Created once.	The power switch
Browser	One browser program (Chrome).	The browser application

BrowserContext	A clean, isolated browser profile with its own cookies and storage.	One incognito window
Page	One tab inside a context. Almost all your code uses <code>page</code> .	One tab
Locator	A way to find an element on the page. It finds the element again every time you use it.	A description: "the Login button"

The chain is: `Playwright` → `Browser` → `BrowserContext` → `Page` → `Locator`. The best habit is **one new context per test**. It is very cheap and every test starts clean, with no cookies left from the last test.

Words you will hear

WORD	SIMPLE MEANING
Auto-waiting	Before clicking, Playwright waits until the element is ready. You do not write <code>Thread.sleep</code> or explicit waits.
Web-first assertions	Assertions like <code>assertThat(locator).hasText(..)</code> retry for a few seconds until true. This kills flaky tests.
Headless / headed	Headless = no window. Headed = you can watch the browser.
Channel	Which installed browser to use. <code>setChannel("chrome")</code> uses your normal Google Chrome.
Trace	A recording of the test (steps, screenshots, network). You open it later to see what went wrong.
Codegen	A tool that records your clicks and writes Java code for you.
Inspector	A debugging window that opens when you call <code>page.pause()</code> .
Strict mode	If a locator matches more than one element, an action on it throws an error. Prevents clicking the wrong one.

2. Setup in Java (Maven)

IMPORTANT Dependencies

Add this to the `dependencies` of your `pom.xml`, plus TestNG for running tests.

```
<dependency>
  <groupId>com.microsoft.playwright</groupId>
  <artifactId>playwright</artifactId>
  <version>1.49.0</version>
</dependency>
<dependency>
  <groupId>org.testng</groupId>
  <artifactId>testng</artifactId>
  <version>7.10.2</version>
</dependency>
```

What it does: One dependency is enough for Playwright. You can use a newer version from Maven Central.

The first time you run Playwright, it downloads its own browsers (a few hundred MB). To use the Chrome already on your computer, add `.setChannel("chrome")` when launching, as the examples below do. To download the browsers by hand:

```
mvn exec:java -e -D exec.mainClass=com.microsoft.playwright.CLI -D
exec.args="install"
```

What it does: Installs Chromium, Firefox and WebKit for Playwright.

Your first test (complete)

```
import com.microsoft.playwright.*;

public class FirstTest {
    public static void main(String[] args) {
        try (Playwright playwright = Playwright.create()) {
            // 1. start Playwright
            Browser browser = playwright.chromium().launch(
                new BrowserType.LaunchOptions().setHeadless(false)); // 2. open Chromium with a window
            Page page = browser.newPage(); // 3. open a new tab
            page.navigate("https://example.com"); // 4. go to a page
            System.out.println(page.title()); // 5. read the title
            page.locator("a").click(); // 6. find a link and click it
            browser.close(); // 7. close the browser
        }
    }
}
```

What it does: The same five steps as in any browser test: open, navigate, act, check, close. The `try (...)` block closes Playwright for you at the end.

IMPORTANT Cross-browser in one line

`playwright.chromium()` can also be `firefox()` or `webkit()`. The rest of the code stays the same. That is how you test on three browser engines.

3. The core objects and their methods

Playwright and BrowserType

METHOD	WHAT IT DOES	EXAMPLE

IMPORTANT	<code>Playwright.create()</code>	Starts Playwright. Always close it at the end (<code>try-with-resources</code> or <code>playwright.close()</code>).	<code>Playwright pw = Playwright.create();</code>
IMPORTANT	<code>playwright.chromium() / firefox() / webkit()</code>	Choose a browser engine.	<code>BrowserType bt = playwright.chromium();</code>
IMPORTANT	<code>browserType.launch(LaunchOptions)</code>	Opens the browser and returns a <code>Browser</code> .	<code>bt.launch(new BrowserType.LaunchOptions().setHeadless(false));</code>
IMPORTANT	<code>LaunchOptions.setHeadless(boolean)</code>	false = show the window. Default is true (no window).	
IMPORTANT	<code>LaunchOptions.setChannel("chrome")</code>	Use the installed Google Chrome (or "msedge").	
GOOD TO KNOW	<code>LaunchOptions.setSlowMo(500)</code>	Slows every action by 500 ms so you can watch.	
GOOD TO KNOW	<code>LaunchOptions.setArgs(List.of("--start-maximized"))</code>	Pass extra browser arguments.	

Browser

	METHOD	WHAT IT DOES	EXAMPLE
IMPORTANT	<code>browser.newContext()</code>	Creates a clean <code>BrowserContext</code> .	<code>BrowserContext ctx = browser.newContext();</code>
IMPORTANT	<code>browser.newPage()</code>	Shortcut: creates a new context AND a page in one go.	<code>Page page = browser.newPage();</code>

IMPORTANT	<code>browser.close()</code>	Closes the browser and all its pages.
GOOD TO KNOW	<code>browser.newContext(new Browser.NewContextOptions())</code>	Options: <code>setViewportSize(1280, 720)</code> , <code>setLocale("en-IN")</code> , <code>setGeolocation(...)</code> , <code>setRecordVideoDir(path)</code> , <code>setStorageState(path)</code> .

BrowserContext

	METHOD	WHAT IT DOES	EXAMPLE
IMPORTANT	<code>context.newPage()</code>	Opens a new tab.	<code>Page page = context.newPage();</code>
IMPORTANT	<code>context.close()</code>	Closes this context and its pages.	
GOOD TO KNOW	<code>context.cookies()</code> / <code>addCookies(...)</code> / <code>clearCookies()</code>	Read, add, or clear cookies.	
GOOD TO KNOW	<code>context.storageState(...)</code>	Saves login state (cookies + storage) to a file so other tests can skip the login.	
GOOD TO KNOW	<code>context.tracing().start(...)</code> / <code>stop(...)</code>	Records a trace of the test. See the debugging section.	
GOOD TO KNOW	<code>context.setDefaultTimeout(ms)</code>	Default wait time for every action in this context.	
GOOD TO KNOW	<code>context.route(...)</code>	Intercept or fake network calls for the whole context.	

Page (the tab)

	METHOD	WHAT IT DOES	EXAMPLE
IMPORTANT	<code>page.navigate(string url)</code>	Opens a URL and waits for it to load.	<code>page.navigate("https://shop.com");</code>

IMPORTANT	<code>page.title()</code>	Returns the page title.	<code>String t = page.title();</code>
IMPORTANT	<code>page.url()</code>	Returns the current URL.	
IMPORTANT	<code>page.locator(String selector)</code>	Creates a Locator from CSS or XPath.	<code>page.locator("#email")</code>
IMPORTANT	<code>page.getByRole()</code> / <code>getByLabel()</code> / <code>getByText()</code> / ...	Creates a Locator the way a user sees the page. See the locators section.	
IMPORTANT	<code>page.reload()</code> / <code>goBack()</code> / <code>goForward()</code>	Browser reload, back and forward.	<code>page.goBack();</code>
IMPORTANT	<code>page.onDialog(handler)</code> / <code>onceDialog(handler)</code>	Handle alert, confirm and prompt popups. Register BEFORE the action that opens them.	
IMPORTANT	<code>page.screenshot(...)</code>	Takes a screenshot.	<code>page.screenshot(new Page.ScreenshotOptions().setPath(Paths.get("a.png")).setFullPage(true));</code>
IMPORTANT	<code>page.close()</code>	Closes the tab.	
GOOD TO KNOW	<code>page.waitForLoadState()</code> / <code>waitForURL(...)</code>	Wait for the page to finish loading or for the URL to change. Rarely needed because of auto-wait.	
GOOD TO KNOW	<code>page.waitForPopup(Runnable)</code> / <code>context.waitForPage(...)</code>	Get the new tab that an action opens.	
GOOD TO KNOW	<code>page.waitForDownload(Runnable)</code>	Get the file that an action downloads.	
GOOD TO KNOW	<code>page.frameLocator(String)</code>	Work inside an iframe.	

GOOD TO KNOW	<code>page.evaluate(string js)</code>	Run JavaScript in the page and get the result.	<code>page.evaluate("document.title")</code>
GOOD TO KNOW	<code>page.pause()</code>	Opens the Inspector so you can debug step by step (needs headed mode).	
GOOD TO KNOW	<code>page.setViewportSize(w, h)</code>	Change the window size, for example a phone size.	<code>page.setViewportSize(375, 812);</code>
GOOD TO KNOW	<code>page.content()</code>	The full HTML of the page.	
GOOD TO KNOW	<code>page.keyboard() / page.mouse()</code>	Low level keys and mouse: <code>keyboard().press("Control+A")</code> , <code>mouse().click(x, y)</code> .	
GOOD TO KNOW	<code>page.route(...) / waitForResponse(...)</code>	Fake, block or watch network requests.	
GOOD TO KNOW	<code>page.setDefaultTimeout(ms)</code>	Default wait time for this page.	<code>page.setDefaultTimeout(10000);</code>

4. Locators: how to find an element

A **Locator** describes an element. It is lazy: nothing is searched until you act on it, and it searches again on every action. So it never becomes "stale" like a Selenium element can.

IMPORTANT Best order of choice

Playwright prefers locators that match **how a user sees the page**: by role (button, link, textbox), by label, by text. These survive HTML changes better than ids and XPath. Use CSS or XPath when there is no better choice.

METHOD	WHAT IT DOES	EXAMPLE

IMPORTANT	<code>page.getByRole(AriaRole.BUTTON, new Page.GetByRoleOptions().setName("Login"))</code>	Finds by role and its visible name. Best for buttons, links, headings, checkboxes.
IMPORTANT	<code>page.getByLabel("Email")</code>	Finds an input by its <code><label></code> text. Best for forms.
IMPORTANT	<code>page.getByPlaceholder("Search products")</code>	Finds an input by its placeholder.
IMPORTANT	<code>page.getByText("Welcome")</code>	Finds by visible text (partial, ignores case by default). Add <code>setExact(true)</code> for exact match.
IMPORTANT	<code>page.locator("#email")</code>	CSS selector (or XPath: <code>page.locator("//input[@id='email']")</code>).
IMPORTANT	<code>page.getByTestId("submit")</code>	Finds by the <code>data-testid</code> attribute. Developers add this just for testing. Most stable.
GOOD TO KNOW	<code>page.getByAltText("Logo")</code> <code>/ getByTitle("Close")</code>	Find an image by alt text, or an element by title.

Refining a locator

	METHOD	WHAT IT DOES	EXAMPLE
IMPORTANT	<code>locator.locator("button")</code>	Search inside an element (chaining).	<code>page.locator("#product-3").locator("button")</code>
IMPORTANT	<code>locator.first() / last() / nth(2)</code>	Pick one of many matches. <code>nth</code> starts at 0.	<code>page.locator(".product").nth(1)</code>
IMPORTANT	<code>locator.count()</code>	How many elements match. Returns an <code>int</code> .	<code>page.locator(".product").count()</code>

GOOD TO KNOW	<code>locator.filter(new Locator.FilterOptions().setHasText("Pen"))</code>	Keep only the matches that contain some text.
GOOD TO KNOW	<code>locator.and(other) / locator.or(other)</code>	Combine two locators.
GOOD TO KNOW	<code>locator.all()</code>	Returns a <code>List<Locator></code> to loop over. for (Locator p : page.locator(".product").all()) { ... }

IMPORTANT Two things that surprise beginners

- **Strict mode:** if `page.locator(".product").click()` matches 3 elements, Playwright throws an error instead of clicking one by chance. Fix it with a more exact locator, or `.first()`, or `.nth(i)`.
- `getByLabel("Password")` also matches "Confirm password", because text matching is partial. Use `new Page.GetByLabelOptions().setExact(true)` to match exactly.
- Test locators quickly with `npx playwright codegen` or the Inspector (see debugging).

XPath and CSS you still need

WANT TO FIND	CSS SELECTOR	XPATH
By id	<code>#email</code>	<code>//*[@id='email']</code>
Tag with attribute	<code>input[type='email']</code>	<code>//input[@type='email']</code>
By exact text	use <code>getByText</code>	<code>//button[text()='Login']</code>
Part of an attribute	<code>div[class*='err']</code>	<code>//div[contains(@class,'err')]</code>
Child inside parent	<code>#product-3 button</code>	<code>//div[@id='product-3']//button</code>
Next to a label	use <code>getByLabel</code>	<code>//label[text()='Email']/following::input[1]</code>

5. Actions: doing things on an element

Every action first runs **auto-wait checks** (see section 7), so you rarely need extra waits.

	METHOD	WHAT IT DOES	EXAMPLE
IMPORTANT	<code>click()</code>	Clicks the element. Waits until it is visible, stable, enabled and not covered.	<code>page.locator("#loginBtn").click();</code>
IMPORTANT	<code>fill(String text)</code>	Clears the input and sets the text in one step. Works on text, date and range inputs.	<code>page.getByLabel("Email").fill("a@b.com");</code>
IMPORTANT	<code>clear()</code>	Empties an input.	
IMPORTANT	<code>pressSequentially(String text)</code>	Types key by key like a real user (slower than fill). Use when the page reacts to each key, or to test <code>maxlength</code> .	<code>box.pressSequentially("hello");</code>
IMPORTANT	<code>press(String key)</code>	Presses a key or key combination.	<code>box.press("Enter");</code> <code>box.press("Control+A");</code>
IMPORTANT	<code>check()</code> / <code>uncheck()</code> / <code>setChecked(booleant)</code>	Ticks or unticks a checkbox or radio button. Safe: it checks the state first.	<code>page.getByLabel("English").check();</code>
IMPORTANT	<code>selectOption(...)</code>	Chooses option(s) in a <code><select></code> . By value or label: <code>selectOption("India")</code> ; many: <code>selectOption(new String[] {"A", "B"})</code> ; by label: <code>new SelectOption().setLabel("USA")</code> ; by index: <code>new SelectOption().setIndex(0)</code> .	
IMPORTANT	<code>setInputFiles(...)</code>	Uploads a file. Use a path <code>Paths.get("a.pdf")</code> or an in-memory <code>new FilePayload(name, mimeType, bytes)</code> .	

IMPORTANT	<code>hover()</code>	Moves the mouse over the element.	<code>page.locator("#menu").hover();</code>
GOOD TO KNOW	<code>dblclick()</code>	Double click.	
GOOD TO KNOW	<code>click(new Locator.ClickOptions().setButton(MouseButton.RIGHT))</code>	Right click. Other options: <code>setForce(true)</code> , <code>setModifiers(...)</code> , <code>setPosition(x, y)</code> , <code>setTimeout(ms)</code> .	
GOOD TO KNOW	<code>focus()</code> / <code>blur()</code>	Move keyboard focus to or away from the element.	
GOOD TO KNOW	<code>dragTo(Locator target)</code>	Drag and drop.	<code>source.dragTo(target);</code>
GOOD TO KNOW	<code>scrollIntoViewIfNeeded()</code>	Scrolls the element into view (click does this automatically).	
GOOD TO KNOW	<code>dispatchEvent("click")</code>	Sends a DOM event directly. A last option.	

Reading values from the page

	METHOD	WHAT IT DOES	EXAMPLE
IMPORTANT	<code>innerText()</code>	The visible text of the element. Returns a String.	<code>String msg = page.locator("#msg").innerText();</code>
IMPORTANT	<code>inputValue()</code>	What is typed inside an input or textarea.	
IMPORTANT	<code>getAttribute(String name)</code>	An attribute value, or null if missing.	<code>el.getAttribute("data-state")</code>
IMPORTANT	<code>isVisible()</code> / <code>isHidden()</code>	True or false right now. No waiting.	

IMPORTANT	<code>isEnabled()</code> / <code>isDisabled()</code> / <code>isChecked()</code> / <code>isEditable()</code>	State of the element right now. No waiting.
GOOD TO KNOW	<code>textContent()</code>	All text including hidden parts.
GOOD TO KNOW	<code>allInnerTexts()</code> / <code>allTextContents()</code>	Texts of ALL matching elements as <code>List<String></code> .
GOOD TO KNOW	<code>boundingBox()</code>	Position and size of the element.

IMPORTANT `is...()` vs assertions

`isVisible()` and the other `is...()` methods give an answer **right now** and do not wait. For tests, prefer the assertions in the next section, which wait and retry.

6. Assertions: checking results

Import this once: `import static`

`com.microsoft.playwright.assertions.PlaywrightAssertions.assertThat;` . Then write

`assertThat(locator).hasText("Hello");` . These assertions **retry** for up to 5 seconds until they become true. That is why Playwright tests are stable.

	METHOD	WHAT IT DOES	EXAMPLE
IMPORTANT	<code>assertThat(locator).isVisible()</code> <code>/ isHidden()</code>	Element is shown / not shown.	
IMPORTANT	<code>assertThat(locator).isEnabled()</code> <code>/ isDisabled()</code>	Element is clickable / disabled.	
IMPORTANT	<code>assertThat(locator).isChecked()</code> <code>)</code>	Checkbox or radio is ticked.	

IMPORTANT	<code>assertThat(locator).hasText("exact text")</code>	The WHOLE text equals this (spaces are ignored). Pass a <code>String[]</code> to check a list of elements, or a <code>Pattern</code> for a regular expression.	<code>assertThat(page.locator("#msg")).hasText("Saved");</code>
IMPORTANT	<code>assertThat(locator).containsText("part")</code>	The text contains this part.	
IMPORTANT	<code>assertThat(locator).hasValue("abc")</code>	An input contains this value.	
IMPORTANT	<code>assertThat(locator).hasAttribute("type", "password")</code>	An attribute has this value.	
IMPORTANT	<code>assertThat(locator).hasCount(3)</code>	Exactly this many elements match.	
IMPORTANT	<code>assertThat(page).hasURL(...) / hasTitle(...)</code>	Check the address or title. Accepts a <code>String</code> or a <code>Pattern</code> .	
IMPORTANT	<code>assertThat(locator).not().isChecked()</code>	<code>.not()</code> reverses any check.	
GOOD TO KNOW	<code>assertThat(locator).isEmpty() / isEditable() / isFocused()</code>	Other state checks.	
GOOD TO KNOW	<code>assertThat(locator).hasClass("active") / hasId / hasCSS</code>	Class, id or style checks.	

GOOD TO KNOW`setDefaultAssertionTimeout(ms)`

Change the 5 second default for all assertions.

```
PlaywrightAssertions.setDefaultAssertionTimeout(10000);
```

IMPORTANT Which assert to use

- Use `assertThat(locator)...` for anything on the page. It waits and retries.
- Use TestNG `Assert.assertEquals(actual, expected)` for plain Java values, such as a number you calculated.
- `hasText("")` with an empty string checks that the element is empty. It does not mean "anything".

7. Auto-waiting (the big difference from Selenium)

IMPORTANT What happens before every click

Before every action, Playwright waits until the element passes these checks. If they do not pass within 30 seconds, you get a `TimeoutError`.

ACTION	PLAYWRIGHT WAITS UNTIL THE ELEMENT IS
<code>click()</code> , <code>hover()</code> , <code>check()</code>	attached to the page, visible, stable (not moving), enabled, and not covered by another element
<code>fill()</code> , <code>clear()</code>	visible, enabled and editable
<code>selectOption()</code>	visible and enabled
<code>innerText()</code> , <code>getAttribute()</code>	attached to the page

In Selenium you write `WebDriverWait` and `ExpectedConditions` yourself. In Playwright this is built in. You only need explicit waiting for special cases, such as `locator.waitFor()` or `page.waitForResponse(...)`.

```

page.setDefaultTimeout(10000); // actions
wait up to 10 s
page.locator("#late").click(new Locator.ClickOptions().setTimeout(3000)); //
this one waits 3 s
page.locator("#spinner").waitFor(new Locator.WaitForOptions()
    .setState(WaitForSelectorState.HIDDEN)); // wait
for the spinner to go

```

What it does: Changing the default wait time, one custom wait, and waiting for an element to disappear.

8. Popups, frames, tabs, downloads

IMPORTANT Alert, confirm, prompt

Popups: by default Playwright closes (dismisses) alert, confirm and prompt popups by itself. To read one or press OK, register the handler BEFORE the click that opens it.

```

page.onceDialog(dialog -> {
    System.out.println(dialog.message()); // read the popup text
    dialog.accept(); // press OK (dialog.dismiss() =
Cancel)
});
page.locator("#submitBtn").click(); // this click opens the popup

```

What it does: `onceDialog` handles the next popup only. `onDialog` handles every popup. A handler that neither accepts nor dismisses blocks the page.

```

page.frameLocator("#payFrame").getByLabel("Card
number").fill("4111111111111111");

```

What it does: Frames: `frameLocator` goes into an iframe. No switching back is needed afterwards, unlike Selenium.

```

Page newTab = page.waitForPopup(() -> page.getByText("Open help").click());
newTab.waitForLoadState();
System.out.println(newTab.title());

```

What it does: New tab: `waitForPopup` runs the click and gives you the new tab as another `Page` object.

```
Download download = page.waitForDownload(() -> page.getByText("Export").click());
download.saveAs(Paths.get("report.csv"));
```

What it does: Download: waits for the file and saves it.

```
page.route("**/api/users", route -> route.fulfill(
    new Route.FulfillOptions().setStatus(200).setBody("[]")));
```

What it does: Network: replaces the answer of a request with your own. Useful to test an empty list or an error.

```
APIRequestContext request = playwright.request().newContext();
APIResponse response = request.get("https://reqres.in/api/users/2");
System.out.println(response.status());    // 200
System.out.println(response.text());
```

What it does: Playwright can also call APIs directly, without a browser, for API tests or test data setup.

9. Setup and cleanup with TestNG

Playwright only drives the browser. We use TestNG (as with Selenium) to run the tests, give annotations and `Assert`. The annotations `@Test`, `@BeforeMethod`, `@AfterMethod`, `@DataProvider` work exactly the same. Here is the base class used in the three cases below.

```
1 package playwrighttests;
2
3 import com.microsoft.playwright.Browser;
4 import com.microsoft.playwright.BrowserContext;
5 import com.microsoft.playwright.BrowserType;
6 import com.microsoft.playwright.Page;
7 import com.microsoft.playwright.Playwright;
8 import org.testng.annotations.AfterClass;
9 import org.testng.annotations.AfterMethod;
10 import org.testng.annotations.BeforeClass;
11 import org.testng.annotations.BeforeMethod;
12
13 public class BaseTest {
14
15     protected Playwright playwright;           // the main object: it starts
16     everything
17     protected Browser browser;                 // one browser (Chrome), shared by
18     all tests in a class
19     protected BrowserContext context;          // a fresh "profile" for each test
20     (like an incognito window)
21     protected Page page;                      // one tab inside the context
22
23     @BeforeClass
24     public void launchBrowser() {
25         playwright = Playwright.create();
26         browser = playwright.chromium().launch(new
27         BrowserType.LaunchOptions()
28             .setChannel("chrome")              // use
29             the Chrome installed on the machine
30             .setHeadless(Boolean.getBoolean("headless"))); // false
31         = you can watch the browser
32     }
33
34     @BeforeMethod
35     public void createPage() {
36         context = browser.newContext();        // new context = clean
37         cookies, storage, no leftovers
38         page = context.newPage();
39     }
40
41     @AfterMethod(alwaysRun = true)
42     public void closeContext() {
43         context.close();
44     }
45
46     @AfterClass(alwaysRun = true)
47     public void closeBrowser() {
48         browser.close();
49         playwright.close();
50     }
51 }
```

```

43     }
44 }

```

LINES	WHAT THIS PART DOES
1-12	Package and imports from Playwright and TestNG.
13-14	Parent class of the test classes. They <code>extend</code> it and get a ready <code>page</code> .
15-19	The four objects from the hierarchy: <code>Playwright</code> , <code>Browser</code> , <code>BrowserContext</code> , <code>Page</code> .
20-27	Runs once before the tests of a class. Starts Playwright and opens Chrome (one browser is shared, which is fast). <code>setChannel("chrome")</code> uses your installed Chrome. <code>-Dheadless=true</code> hides the window.
28-33	Runs before every test. A NEW context and page, so every test starts clean with no cookies.
34-38	Closes the context after each test, even if it failed.
39-44	Closes the browser and Playwright at the end of the class.

10. Debugging and tools

	TOOL	WHAT IT DOES	HOW
IMPORTANT	Run with a visible browser	Use <code>setHeadless(false)</code> , and add <code>setSlowMo(500)</code> to watch every step.	<code>mvn test -Dtest=PasswordTest</code> (without <code>-Dheadless=true</code>)
IMPORTANT	<code>page.pause()</code>	Stops the test and opens the Inspector. You can step through and try locators.	<code>page.pause();</code>
IMPORTANT	Codegen	Opens a browser, records what you click, and writes the Java code.	<code>mvn exec:java -e -D exec.mainClass=com.microsoft.playwright.CLI -D exec.args="codegen https://example.com"</code>

GOOD TO KNOW	Trace viewer	A recording with a timeline, screenshots, DOM and network of each step. Great for failures in CI.	<pre>context.tracing().start(new Tracing.StartOptions().setScreenshots(true).setSnapshots(true)); ... context.tracing().stop(new Tracing.StopOptions().setPath(P aths.get("trace.zip")));</pre>
GOOD TO KNOW	Open a trace	Opens the saved trace in a viewer.	<pre>mvn exec:java -e -D exec.mainClass=com.microsoft.pl aywright.CLI -D exec.args="show-trace trace.zip"</pre>
GOOD TO KNOW	Video	Records a video of the test.	<pre>browser.newContext(new Browser.NewContextOptions().set RecordVideoDir(Paths.get("video s/")));</pre>
GOOD TO KNOW	Debug mode	Setting the environment variable <code>PWDEBUG=1</code> opens the Inspector for the whole run.	<pre>PWDEBUG=1 mvn test - Dtest=PasswordTest</pre>

11. Three real automations

Click a case below to open it. In each case you will see: the practice page, where every element is, the manual test cases, the full Java code with line numbers, and the code explained line by line. **Read the manual test cases first, then the code.** That is how real automation is done: manual thinking first, code second.

IMPORTANT Before you start

- Open each practice page in Chrome and test it by hand first. The page links are inside each case below.
- `PAGE_URL` in each test already points to the live practice page on GitHub Pages, so you only need internet. To use your own copy, open the file from `docs/demo-pages/` and put its `file:///` path in `PAGE_URL`.

- Run with Maven: `mvn test -Dtest="playwrighttests.PasswordTest"` . Remove `-Dheadless=true` to watch the browser work.
- All three cases were run and passed (66 checks) on the pages as given.

▼ Case 1: Password validation (a full validation spec)

Boundary values, equivalence partitions, negative and state testing on one password form.

STEP 1: THE PRACTICE PAGE

Open the page, click around like a manual tester first, then look at the locators below.

Open demo-password.html

The test already points to this live page (`PAGE_URL`), so it runs from any computer with internet. The same file is in the repository at `docs/demo-pages/demo-password.html` .

IMPORTANT What you will learn from this case

- Data-driven tests (one method runs many times with different data)
- Boundary value analysis and equivalence partitioning in real code
- Checking one rule at a time
- Reading attributes and the enabled / disabled state of a button

STEP 2: WHERE EACH ELEMENT IS (LOCATORS)

ELEMENT	HTML ON THE PAGE	PLAYWRIGHT LOCATOR (BEST)	CSS SELECTOR	XPATH
Username box	<code><input id="username" type="text"></code>	<code>page.getByLabel("Username")</code>	<code>#username</code>	<code>//input[@id='username']</code>
Password box	<code><input id="password" type="password"></code>	<code>page.getByLabel("Password", exact)</code>	<code>input[type='password']</code>	<code>//input[@id='password']</code>

Confirm password box	<code><input id="confirmPassword" type="password"></code>	<code>page.getByLabel("Confirm password")</code>	<code>#confirmPassword</code>	<code>//label[text()='Confirm password']/following::input[1]</code>
Show / Hide button	<code><button id="togglePwd">Show</button></code>	<code>page.getByRole(BUTTON, name="Show")</code>	<code>#togglePwd</code>	<code>//button[text()='Show']</code>
One rule line	<code><li id="rule-length" data-state="pending"></code>	<code>page.locator("#rule-length")</code>	<code>li[data-state='pass']</code>	<code>//li[@id='rule-length']</code>
Strength text	<code>Not entered</code>	<code>page.locator("#strength")</code>	<code>#strength</code>	<code>//span[@id='strength']</code>
Error message	<code><div id="pwdError" class="error"></code>	<code>page.locator("#pwdError")</code>	<code>#pwdError</code>	<code>//div[@id='pwdError']</code>
Create password button	<code><button type="submit" id="submitBtn" disabled></code>	<code>page.getByRole(BUTTON, name="Create password")</code>	<code>button[type='submit']</code>	<code>//button[@type='submit']</code>
Success message	<code><div id="successMsg" hidden></code>	<code>page.locator("#successMsg")</code>	<code>#successMsg</code>	<code>//div[contains(text(),'created successfully')]</code>

STEP 3: MANUAL TEST CASES FIRST

Automation follows manual testing. We first decide what to test, and only then write code.

ID	TECHNIQUE USED	INPUT	EXPECTED RESULT
TC01	State check	Nothing typed	Button disabled, strength "Not entered", rules pending
TC02-05	Positive + boundary	Abcd@1234, Abc@1234 (8 chars), Abcdefgh@1234567 (16 chars), Str0ng@Passw0rd	All rules pass, Medium or Strong, button enabled, success message
TC06-07	Boundary value	7 characters, 17 characters	Only the length rule fails
TC08-11	Equivalence partition	No uppercase, no lowercase, no digit, no special character	Only that one rule fails
TC12	Error guessing	A space inside the password	No-space rule fails
TC13	Business rule	Username ubed, password MyUBED@123	No-username rule fails
TC14-16	Error guessing (hostile input)	SQL text, script text, only spaces	Rejected, button disabled, no popup
TC17-19	Negative: confirm password	Different, different only in case, empty	Error message, button disabled
TC20-21	State transition	Valid then invalid then valid; username changed later	Button enabled or disabled to match the data
TC22	UI behaviour	Click Show / Hide	Field type switches between password and text

STEP 4: THE CODE

File: `playwrighttests/PasswordTest.java`

```

1  package playwrighttests;
2
3  import com.microsoft.playwright.Locator;
4  import com.microsoft.playwright.Page;
5  import com.microsoft.playwright.options.AriaRole;
6  import org.testng.Assert;
7  import org.testng.annotations.DataProvider;
8  import org.testng.annotations.Test;
9
10 import java.util.concurrent.atomic.AtomicBoolean;
11
12 import static
    com.microsoft.playwright.assertions.PlaywrightAssertions.assertThat;
13
14 public class PasswordTest extends BaseTest {
15
16     // Live practice page hosted on GitHub Pages (works from any
computer with internet)
17     private static final String PAGE_URL =
        "https://ubedzz49.github.io/qa-interview-prep/demo-pages/demo-
        password.html";
18
19     private static final String[] ALL_RULES =
20         {"length", "upper", "lower", "number", "special",
        "nospace", "nouser"};
21
22     // ----- Helper methods -----
23     private Locator passwordBox() {
24         return page.getByLabel("Password", new
        Page.GetByLabelOptions().setExact(true));
25     }
26
27     private void enter(String pwd, String confirm, String user) {
28         page.getByLabel("Username").fill(user);
29         passwordBox().fill(pwd);
30         page.getByLabel("Confirm password").fill(confirm);
31     }
32
33     private void enter(String pwd) {
34         enter(pwd, pwd, "");
35     }
36
37     private void expectRule(String rule, String state) {
38         assertThat(page.locator("#rule-" + rule)).hasAttribute("data-
        state", state);
39     }
40
41     private void open() {
42         page.navigate(PAGE_URL);

```

```

43     }
44
45     // ----- 1. Initial state -----
46     @Test
47     public void tc01_initialState() {
48         open();
49         assertThat(page.locator("#submitBtn")).isDisabled();
50         assertThat(page.locator("#strength")).hasText("Not entered");
51         for (String rule : ALL_RULES) {
52             expectRule(rule, "pending");
53         }
54         assertThat(page.locator("#pwdError")).isEmpty();
55         assertThat(page.locator("#successMsg")).isHidden();
56     }
57
58     // ----- 2. Positive cases -----
59     @DataProvider(name = "positive")
60     public Object[][] positive() {
61         return new Object[][]{
62             // id,          technique,          password,
63             expected strength
64             {"TC02", "Valid typical password", "Abcd@1234",
65             "Medium"},
66             {"TC03", "Boundary: 8 chars (min)", "Abc@1234",
67             "Medium"},
68             {"TC04", "Boundary: 16 chars (max)",
69             "Abcdefgh@1234567", "Strong"},
70             {"TC05", "Strong password (12+ chars)",
71             "Str0ng@Passw0rd", "Strong"},
72         };
73     }
74
75     @Test(dataProvider = "positive")
76     public void positiveCases(String id, String technique, String pwd,
77     String expectedStrength) {
78         open();
79         enter(pwd);
80
81         for (String rule : ALL_RULES) {
82             expectRule(rule, "pass");
83         }
84
85         assertThat(page.locator("#strength")).hasText(expectedStrength);
86         assertThat(page.locator("#pwdError")).isEmpty();
87         assertThat(page.locator("#submitBtn")).isEnabled();
88
89         page.getByRole(AriaRole.BUTTON, new
90         Page.GetByRoleOptions().setName("Create password")).click();

```

```

83         assertThat(page.locator("#successMsg")).hasText("Password
created successfully");
84     }
85
86     // ----- 3. Negative cases: break exactly ONE rule at a time -
-----
87     @DataProvider(name = "negative")
88     public Object[][] negative() {
89         return new Object[][]{
90             // id,         technique,
password,         username, broken rule
91             {"TC06", "Boundary: 7 chars (min - 1)",
"Abc@123", "", "length"},
92             {"TC07", "Boundary: 17 chars (max + 1)",
"Abcdefgh@12345678", "", "length"},
93             {"TC08", "Partition: no uppercase letter",
"abcd@1234", "", "upper"},
94             {"TC09", "Partition: no lowercase letter",
"ABCD@1234", "", "lower"},
95             {"TC10", "Partition: no number",
"Abcd@efgh", "", "number"},
96             {"TC11", "Partition: no special character",
"Abcd12345", "", "special"},
97             {"TC12", "Error guessing: space inside", "Abcd
@1234", "", "nospace"},
98             {"TC13", "Business rule: contains username",
"MyUBED@123", "ubed", "nouser"},
99         };
100     }
101
102     @Test(dataProvider = "negative")
103     public void negativeCases(String id, String technique, String pwd,
String user, String broken) {
104         open();
105         enter(pwd, pwd, user);
106
107         for (String rule : ALL_RULES) {
108             expectRule(rule, rule.equals(broken) ? "fail" : "pass");
109         }
110         assertThat(page.locator("#strength")).hasText("Weak");
111         assertThat(page.locator("#pwdError")).hasText("Password does
not meet all the rules");
112         assertThat(page.locator("#submitBtn")).isDisabled();
113     }
114
115     // ----- 4. Error guessing: hostile and strange input -----
--
116     @Test
117     public void tc14_sqlInjectionTextIsRejected() {

```

```

118         open();
119         enter("' OR '1'='1");
120         assertThat(page.locator("#submitBtn")).isDisabled();
121         assertThat(page.locator("#successMsg")).isHidden();
122     }
123
124     @Test
125     public void tc15_scriptTextShowsNoPopup() {
126         open();
127         AtomicBoolean popupSeen = new AtomicBoolean(false);
128         page.onDialog(dialog -> { // runs only if a popup
129             popupSeen.set(true);
130             dialog.dismiss();
131         });
132         enter("<script>alert(1)</script>");
133         assertThat(page.locator("#submitBtn")).isDisabled();
134         Assert.assertFalse(popupSeen.get(), "An unexpected popup
135         appeared");
136     }
137
138     @Test
139     public void tc16_onlySpacesAreRejected() {
140         open();
141         enter(" ");
142         expectRule("nospace", "fail");
143         assertThat(page.locator("#submitBtn")).isDisabled();
144     }
145
146     // ----- 5. Confirm password -----
147     @Test
148     public void tc17_mismatchShowsError() {
149         open();
150         enter("Abcd@1234", "Abcd@12345", "");
151         assertThat(page.locator("#confirmError")).hasText("Passwords do
152         not match");
153         assertThat(page.locator("#submitBtn")).isDisabled();
154     }
155
156     @Test
157     public void tc18_caseDifferenceIsMismatch() {
158         open();
159         enter("Abcd@1234", "abcd@1234", "");
160         assertThat(page.locator("#confirmError")).hasText("Passwords do
161         not match");
162     }
163
164     @Test
165     public void tc19_emptyConfirmKeepsButtonDisabled() {

```

```

163         open();
164         enter("Abcd@1234", "", "");
165         assertThat(page.locator("#submitBtn")).isDisabled();
166     }
167
168     // ----- 6. State transitions -----
169     @Test
170     public void tc20_validInvalidValidChangesButton() {
171         open();
172         Locator button = page.locator("#submitBtn");
173         enter("Abcd@1234");
174         assertThat(button).isEnabled();
175
176         passwordBox().fill("abcd@1234"); // breaks
177         the uppercase rule
178         assertThat(button).isDisabled();
179
180         passwordBox().fill("Abcd@1234");
181         assertThat(button).isEnabled();
182     }
183
184     @Test
185     public void tc21_changingUsernameCanBreakValidPassword() {
186         open();
187         enter("Abcd@1234");
188         assertThat(page.locator("#submitBtn")).isEnabled();
189         page.getByLabel("Username").fill("abcd");
190         expectRule("nouser", "fail");
191         assertThat(page.locator("#submitBtn")).isDisabled();
192     }
193
194     // ----- 7. UI behaviour -----
195     @Test
196     public void tc22_showHideTogglesFieldType() {
197         open();
198         Locator toggle = page.locator("#togglePwd");
199         assertThat(passwordBox()).hasAttribute("type", "password");
200         toggle.click();
201         assertThat(passwordBox()).hasAttribute("type", "text");
202         assertThat(toggle).hasText("Hide");
203         toggle.click();
204         assertThat(passwordBox()).hasAttribute("type", "password");
205     }

```

STEP 5: EXPLAINED LINE BY LINE

LINES	WHAT THIS PART DOES

1-13	Package and imports. <code>assertThat</code> is the retrying Playwright assertion. <code>AtomicBoolean</code> holds a flag that a popup listener can change.
14-16	Extends <code>BaseTest</code> , so <code>page</code> is ready. Every <code>@Test</code> method gets a fresh context and page.
17-18	Address of the practice page. Change <code>PATH/T0</code> to your folder.
19-21	The seven rule names. We loop over them instead of writing seven lines each time.
22	Small helper methods, so the tests below read clearly.
23-26	<code>getByLabel("Password")</code> would also match "Confirm password". <code>setExact(true)</code> makes it match only the label "Password".
27-36	<code>enter()</code> fills three boxes with <code>fill()</code> . <code>getByLabel</code> finds an input by its label, like a user would. The short <code>enter(pwd)</code> below is an overloaded shortcut.
37-44	Checks the <code>data-state</code> attribute of a rule line. The page sets it to <code>pass</code> , <code>fail</code> or <code>pending</code> . <code>assertThat(...).hasAttribute</code> retries until true.
45-57	TC01: before typing anything the button is disabled, strength says "Not entered", every rule is pending, and there is no error or success text.
58-69	Data provider with four valid passwords. Two are the boundaries 8 and 16 characters. One test method runs four times.
70-85	Types the password, checks all rules pass, the strength text, no error and an enabled button. Then it finds the button by its role and name, clicks it, and checks the success message.
86-101	Eight invalid passwords. Each breaks exactly ONE rule: boundary 7 and 17, no uppercase / lowercase / number / special, a space, and the username rule.
102-114	For each rule it expects <code>fail</code> only for the broken rule and <code>pass</code> for all the others. This proves the page flags the right rule and nothing else. It also checks "Weak", the error text, and a disabled button.
115-144	TC14: SQL text. TC15: script text; <code>onDialog</code> would set the flag if a popup ever appeared, and we assert that it did not. TC16: only spaces.
145-167	Different confirm password, a difference only in letter case, and an empty confirm password.

- 168-192** The button must follow the data: valid, then invalid, then valid again. Changing the username later can also break a valid password.
- 193-205** The Show / Hide button changes the field `type` between `password` and `text`, and its own text between Show and Hide.

STEP 6: RUN IT

```
mvn test -Dtest="playwrighttests.PasswordTest" // run this
class only
mvn test -Dtest="playwrighttests.PasswordTest" -Dheadless=true // run
without a browser window
mvn test // run every test
```

▼ Case 2: Full page validation (every kind of input)

Text boxes, date, radio, checkbox, dropdowns, multi-select, slider, textarea, file upload and popups.

STEP 1: THE PRACTICE PAGE

Open the page, click around like a manual tester first, then look at the locators below.

Open demo-form.html

The test already points to this live page (`PAGE_URL`), so it runs from any computer with internet. The same file is in the repository at `docs/demo-pages/demo-form.html`.

IMPORTANT What you will learn from this case

- Every input type and how to handle it
- Dependent dropdowns and multi-select
- Alert and confirm popups
- File upload and boundary testing on file size

STEP 2: WHERE EACH ELEMENT IS (LOCATORS)

ELEMENT	HTML ON THE PAGE	PLAYWRIGHT LOCATOR (BEST)	CSS SELECTOR	XPATH

Full name box	<code><input id="fullName" type="text"></code>	<code>page.getByLabel ("Full name")</code>	<code>#fullName</code>	<code>//input[@id ='fullName']</code>
Date of birth	<code><input id="dob" type="date"></code>	<code>page.getByLabel ("Date of birth")</code>	<code>input[type=' date']</code>	<code>//input[@id ='dob']</code>
Male radio button	<code><input type="radio" name="gender" id="genderMale" value="Male"></code>	<code>page.getByLabel ("Male", exact)</code>	<code>input[name=' gender'] [value='Male']</code>	<code>//input[@na me='gender' and @value='Mal e']</code>
Country dropdown	<code><select id="country"></code>	<code>page.locator("# country")</code>	<code>select#count ry</code>	<code>//select[@i d='country']</code>
State dropdown (starts disabled)	<code><select id="state" disabled></code>	<code>page.locator("# state")</code>	<code>select#stat e</code>	<code>//select[@i d='state']</code>
Skills multi- select	<code><select id="skills" multiple></code>	<code>page.locator("# skills")</code>	<code>select[multi ple]</code>	<code>//select[@m ultiple]</code>
English checkbox	<code><input type="checkbox" name="lang" id="langEnglish" " value="English" ></code>	<code>page.getByLabel ("English")</code>	<code>input[name=' lang']:checke d</code>	<code>//input[@na me='lang' and @value='Eng lish']</code>
Experience slider	<code><input type="range" id="experience" min="0" max="20"></code>	<code>page.locator("# experience")</code>	<code>input[type=' range']</code>	<code>//input[@ty pe='range']</code>

Address textarea	<code><textarea id="address" maxlength="200" ></code>	<code>page.getByLabel ("Address")</code>	<code>textarea#add ress</code>	<code>//textarea[@id='addres s']</code>
Resume file input	<code><input type="file" id="resume"></code>	<code>page.getByLabel (/Resume/)</code>	<code>input[type=' file']</code>	<code>//input[@ty pe='file']</code>
Error under a field	<code><div class="error" id="err-email"></code>	<code>page.locator("# err-email")</code>	<code>#err-email</code>	<code>//div[@id=' err-email']</code>
Submit / Reset buttons	<code><button type="submit" id="submitBtn"> Submit</button></code>	<code>page.getByRole(BUTTON, name "Submit")</code>	<code>button[type= 'submit']</code>	<code>//button[te xt()='Reset<br ']<="" code=""/></code>

STEP 3: MANUAL TEST CASES FIRST

Automation follows manual testing. We first decide what to test, and only then write code.

ID	CONTROL	WHAT IS TESTED	TECHNIQUE
TC01-02	Whole form	Start state; submit empty form shows 11 error messages	State check, required fields
TC03	Name, email, phone (text)	Length 1, 2, 40, 41; letters only; email formats; phone length and first digit	Boundary value, equivalence partition
TC04	Date of birth (date)	Exactly 18 years, one day short, one day over, future date	Boundary value
TC05	Gender (radio)	Only one can be selected at a time	State check
TC06-07	Country and State (dependent dropdowns)	State list changes with country; disabled again when country is cleared	Dependency check
TC08	Skills (multi-select)	1 to 3 allowed; 4 and 0 give errors	Boundary value

TC09	Languages (checkboxes)	Tick and untick, count of ticked boxes	State check
TC10	Experience (slider)	Value display at 12 and at maximum 20	Boundary value
TC11-12	Address (textarea)	Counter, 200 character limit, minimum 10 characters	Boundary value
TC13	Resume (file upload)	PDF, PNG, exactly 2 MB, 2 MB + 1 byte, empty file, cv.pdf.exe	Boundary value, error guessing
TC14	Terms (checkbox)	Must be ticked	Required field
TC15	Submit	Full valid form, alert popup, result text	Positive end-to-end
TC16	Reset	Confirm popup: Cancel keeps data, OK clears it	Popup handling

STEP 4: THE CODE

File: playwrighttests/FormTest.java

```
1 package playwrighttests;
2
3 import com.microsoft.playwright.Locator;
4 import com.microsoft.playwright.Page;
5 import com.microsoft.playwright.options.AriaRole;
6 import com.microsoft.playwright.options.FilePayload;
7 import com.microsoft.playwright.options.SelectOption;
8 import org.testng.Assert;
9 import org.testng.annotations.DataProvider;
10 import org.testng.annotations.Test;
11
12 import java.time.LocalDate;
13 import java.util.concurrent.atomic.AtomicReference;
14 import java.util.regex.Pattern;
15
16 import static
17     com.microsoft.playwright.assertions.PlaywrightAssertions.assertThat;
18
19 public class FormTest extends BaseTest {
20     // Live practice page hosted on GitHub Pages (works from any
21     // computer with internet)
22     private static final String PAGE_URL =
23         "https://ubedzz49.github.io/qa-interview-prep/demo-pages/demo-
24         form.html";
25     private static final int MB = 1024 * 1024;
26
27     // ----- Helper methods -----
28     private void open() {
29         page.navigate(PAGE_URL);
30     }
31
32     private void clickSubmit() {
33         page.getByRole(AriaRole.BUTTON, new
34             Page.GetByRoleOptions().setName("Submit")).click();
35     }
36
37     private Locator error(String field) {
38         return page.locator("#err-" + field);
39     }
40
41     // ----- 1. Initial state -----
42     @Test
43     public void tc01_initialState() {
44         open();
45         assertThat(page.locator("#state")).isDisabled();
46         assertThat(page.locator("#expOut")).hasText("0");
47         assertThat(page.locator("#addrCount")).hasText("0");
48     }
49 }
```

```

44     assertThat(page.locator("input[name='gender']:checked")).hasCount(0);
45     assertThat(page.locator("#terms")).not().isChecked();
46     assertThat(page.locator("#result")).isHidden();
47 }
48
49 // ----- 2. Empty form: every required field shows an error --
-----
50 @Test
51 public void tc02_emptyFormShowsAllErrors() {
52     open();
53     clickSubmit();
54     String[][] expected = {
55         {"fullName", "Full name is required"},
56         {"email", "Email is required"},
57         {"phone", "Mobile number is required"},
58         {"dob", "Date of birth is required"},
59         {"gender", "Select a gender"},
60         {"country", "Select a country"},
61         {"skills", "Select at least one skill"},
62         {"lang", "Select at least one language"},
63         {"address", "Address is required"},
64         {"resume", "Upload your resume"},
65         {"terms", "You must accept the terms"},
66     };
67     for (String[] row : expected) {
68         assertThat(error(row[0])).hasText(row[1]);
69     }
70 }
71
72 // ----- 3. Text inputs: data-driven validation -----
73 @DataProvider(name = "textCases")
74 public Object[][] textCases() {
75     String phoneMsg = "Enter a 10 digit number starting with 6, 7,
8 or 9";
76     String nameLenMsg = "Name must be 2 to 40 characters";
77     return new Object[][]{
78         // field,          value,          expected message,
79 why
80         {"fullName", "A",          nameLenMsg,
81 "Boundary: 1 char (min - 1)"},
82         {"fullName", "A1",          "",
83 "Boundary: 2 chars (min)"},
84         {"fullName", "A".repeat(40), "",
85 "Boundary: 40 chars (max)"},
86         {"fullName", "A".repeat(41), nameLenMsg,
87 "Boundary: 41 chars (max + 1)"},
88         {"fullName", "John3",       "Name can have only
89 letters, space, dot, hyphen and apostrophe", "Invalid: digit in name"},

```

```

84         {"fullName", "Mary-Ann O'Neil", "", "Valid
special characters"},
85         {"email", "abc", "Enter a valid email
address", "Invalid: no @"},
86         {"email", "abc@mail", "Enter a valid email
address", "Invalid: no domain extension"},
87         {"email", "a b@mail.com", "Enter a valid email
address", "Invalid: space"},
88         {"email", "user@mail.com", "", "Valid
email"},
89         {"phone", "12345", phoneMsg, "Invalid:
too short"},
90         {"phone", "5876543210", phoneMsg, "Invalid:
starts with 5"},
91         {"phone", "98765432101", phoneMsg,
"Boundary: 11 digits"},
92         {"phone", "abcdefghij", phoneMsg, "Invalid:
letters"},
93         {"phone", "9876543210", "", "Valid
mobile number"},
94     };
95 }
96
97 @Test(dataProvider = "textCases")
98 public void tc03_textFieldValidation(String field, String value,
String message, String why) {
99     open();
100     page.locator("#" + field).fill(value);
101     clickSubmit();
102     assertThat(error(field)).hasText(message);
103 }
104
105 // ----- 4. Date of birth: boundary around 18 years -----
106 @DataProvider(name = "dobCases")
107 public Object[][] dobCases() {
108     LocalDate today = LocalDate.now();
109     return new Object[][]{
110         {"exactly 18 years today (valid)",
today.minusYears(18).toString(), ""},
111         {"turns 18 tomorrow (invalid)",
today.minusYears(18).plusDays(1).toString(), "You must be at least 18
years old"},
112         {"18 years and 1 day (valid)",
today.minusYears(18).minusDays(1).toString(), ""},
113         {"tomorrow (future date)",
today.plusDays(1).toString(), "Date of birth cannot be in
the future"},
114     };
115 }

```

```

116
117     @Test(dataProvider = "dobCases")
118     public void tc04_dateOfBirth(String why, String date, String
message) {
119         open();
120         page.locator("#dob").fill(date);                                // fill()
works directly on date inputs
121         clickSubmit();
122         assertThat(error("dob")).hasText(message);
123     }
124
125     // ----- 5. Radio buttons -----
126     @Test
127     public void tc05_onlyOneRadioCanBeSelected() {
128         open();
129         page.getByLabel("Female").check();
130         assertThat(page.getByLabel("Female")).isChecked();
131         page.getByLabel("Other").check();
132         assertThat(page.getByLabel("Other")).isChecked();
133         assertThat(page.getByLabel("Female")).not().isChecked();
134
assertThat(page.locator("input[name='gender']:checked")).hasCount(1);
135     }
136
137     // ----- 6. Dropdowns: single and dependent -----
138     @Test
139     public void tc06_countryChangesStateList() {
140         open();
141         Locator state = page.locator("#state");
142
143         page.locator("#country").selectOption("India");
144         assertThat(state).isEnabled();
145         assertThat(state.locator("option")).hasText(
146             new String[]{"-- Select state --", "Karnataka",
"Maharashtra", "Delhi"});
147
148         page.locator("#country").selectOption(new
SelectOption().setLabel("USA"));    // by visible text
149         assertThat(state.locator("option")).hasText(
150             new String[]{"-- Select state --", "California",
"Texas"});
151
152         page.locator("#country").selectOption(new
SelectOption().setIndex(0));        // back to first option
153         assertThat(state).isDisabled();
154     }
155
156     @Test
157     public void tc07_stateIsRequiredAfterChoosingCountry() {

```

```

158         open();
159         page.locator("#country").selectOption("UK");
160         clickSubmit();
161         assertThat(error("state")).hasText("Select a state");
162     }
163
164     // ----- 7. Multi-select -----
165     @Test
166     public void tc08_multiSelectAllowsOneToThreeSkills() {
167         open();
168         Locator skills = page.locator("#skills");
169
170         skills.selectOption(new String[]{"Selenium", "Git"});
171         clickSubmit();
172         assertThat(error("skills")).hasText("");
173
174         skills.selectOption(new String[]{"Selenium", "Playwright",
175 "SQL", "Git"}); // 4 = max + 1
176         clickSubmit();
177         assertThat(error("skills")).hasText("Select at most 3 skills");
178
179         skills.selectOption(new String[]{});
180         // clear all
181         clickSubmit();
182         assertThat(error("skills")).hasText("Select at least one
skill");
183     }
184
185     // ----- 8. Checkboxes -----
186     @Test
187     public void tc09_languageCheckboxes() {
188         open();
189         page.getByLabel("English").check();
190         page.getByLabel("Telugu").check();
191
192         assertThat(page.locator("input[name='lang']:checked")).hasCount(2);
193         page.getByLabel("English").uncheck();
194
195         assertThat(page.locator("input[name='lang']:checked")).hasCount(1);
196     }
197
198     // ----- 9. Slider -----
199     @Test
200     public void tc10_sliderShowsItsValue() {
201         open();
202         page.locator("#experience").fill("12");
203         assertThat(page.locator("#expOut")).hasText("12");
204         page.locator("#experience").fill("20"); // max
205         assertThat(page.locator("#expOut")).hasText("20");

```

```

202     }
203
204     // ----- 10. Textarea -----
205     @Test
206     public void tc11_textareaCounterAndLimit() {
207         open();
208         Locator box = page.locator("#address");
209         box.pressSequentially("Hello");
210         assertThat(page.locator("#addrCount")).hasText("5");
211
212         box.fill("");
213         box.pressSequentially("x".repeat(205)); // we
type 205, the browser stops at 200
214         assertThat(page.locator("#addrCount")).hasText("200");
215     }
216
217     @Test
218     public void tc12_shortAddressIsRejected() {
219         open();
220         page.locator("#address").fill("Short");
221         clickSubmit();
222         assertThat(error("address")).hasText("Address must be at least
10 characters");
223     }
224
225     // ----- 11. File upload -----
226     @DataProvider(name = "fileCases")
227     public Object[][] fileCases() {
228         return new Object[][]{
229             // why, file name, size,
expected message
230             {"valid PDF", "resume.pdf", 1000,
""},
231             {"wrong type (png)", "photo.png", 1000,
"Only PDF, DOC or DOCX files are allowed"},
232             {"exactly 2 MB (max)", "big.pdf", 2 * MB,
""},
233             {"2 MB + 1 byte (max + 1)", "bigger.pdf", 2 * MB + 1,
"File must be 2 MB or smaller"},
234             {"empty file", "empty.pdf", 0,
"File is empty"},
235             {"double extension", "cv.pdf.exe", 1000,
"Only PDF, DOC or DOCX files are allowed"},
236         };
237     }
238
239     @Test(dataProvider = "fileCases")
240     public void tc13_resumeUpload(String why, String name, int size,
String message) {

```

```

241         open();
242         // Playwright can upload a file made in memory. No real file is
needed on disk.
243         page.locator("#resume").setInputFiles(new FilePayload(name,
"application/octet-stream", new byte[size]));
244         clickSubmit();
245         assertThat(error("resume")).hasText(message);
246     }
247
248     // ----- 12. Terms checkbox -----
249     @Test
250     public void tc14_termsMustBeAccepted() {
251         open();
252         clickSubmit();
253         assertThat(error("terms")).hasText("You must accept the
terms");
254         page.locator("#terms").check();
255         clickSubmit();
256         assertThat(error("terms")).hasText("");
257     }
258
259     // ----- 13. Happy path: whole form, plus the alert popup ----
-----
260     @Test
261     public void tc15_validFormSubmitsAndShowsResult() {
262         open();
263         page.getByLabel("Full name").fill("Ravi Kumar");
264         page.getByLabel("Email").fill("ravi@example.com");
265         page.getByLabel("Mobile number").fill("9876543210");
266         page.getByLabel("Date of birth").fill("1998-05-20");
267         page.getByLabel("Male", new
Page.GetByLabelOptions().setExact(true)).check();
268         page.getByLabel("Country").selectOption("India");
269         page.getByLabel("State").selectOption("Karnataka");
270         page.getByLabel(Pattern.compile("Skills")).selectOption(new
String[]{"Selenium", "Playwright"});
271         page.getByLabel("English").check();
272         page.getByLabel("Hindi").check();
273         page.getByLabel(Pattern.compile("Experience in
years")).fill("2");
274         page.getByLabel("Address").fill("12 MG Road, Bengaluru");
275         page.getByLabel(Pattern.compile("Resume")).setInputFiles(
276             new FilePayload("ravi.pdf", "application/pdf", "dummy
pdf".getBytes()));
277         page.getByLabel("I agree to the terms and conditions").check();
278
279         // The page shows an alert popup after a good submit. Listen
BEFORE clicking.
280         AtomicReference<String> popupText = new AtomicReference<>("");

```

```

281         page.onceDialog(dialog -> {
282             popupText.set(dialog.message());
283             dialog.accept();
284         });
285         clickSubmit();
286
287         Assert.assertEquals(popupText.get(), "Application submitted
successfully");
288         Locator result = page.locator("#result");
289         assertThat(result).containsText("\"fullName\": \"Ravi
Kumar\"");
290         assertThat(result).containsText("\"phone\": \"9876543210\"");
291         assertThat(result).containsText("\"state\": \"Karnataka\"");
292         assertThat(result).containsText("\"experience\": 2");
293         assertThat(result).containsText("\"resume\": \"ravi.pdf\"");
294     }
295
296     // ----- 14. Reset button and the confirm popup -----
297     @Test
298     public void tc16_resetAsksForConfirmation() {
299         open();
300         page.getByLabel("Full name").fill("Ravi Kumar");
301         page.locator("#country").selectOption("USA");
302
303         page.onceDialog(dialog -> dialog.dismiss());           //
304         press Cancel
305         page.getByRole(AriaRole.BUTTON, new
Page.GetByRoleOptions().setName("Reset")).click();
306         assertThat(page.getByLabel("Full name")).hasValue("Ravi
Kumar");
307
308         page.onceDialog(dialog -> dialog.accept());           //
309         press OK
310         page.getByRole(AriaRole.BUTTON, new
Page.GetByRoleOptions().setName("Reset")).click();
311         assertThat(page.getByLabel("Full name")).hasValue("");
312         assertThat(page.locator("#state")).isDisabled();
313     }
314 }

```

STEP 5: EXPLAINED LINE BY LINE

LINES	WHAT THIS PART DOES
1-17	Imports. <code>SelectOption</code> for dropdowns, <code>FilePayload</code> for in-memory file upload, <code>AtomicReference</code> to carry a popup message out of a listener.

18-23	Extends <code>BaseTest</code> . <code>PAGE_URL</code> is the page address. <code>MB</code> is the number of bytes in 1 MB, used for the file size boundaries.
24-36	<code>open()</code> , <code>clickSubmit()</code> (button found by role and name) and <code>error(field)</code> , which returns the red message under a field (<code>err-<field></code>).
37-48	TC01: the State dropdown is disabled, counters show 0, no radio is selected, terms are not ticked, the result is hidden.
49-71	TC02: click Submit on an empty form. A 2D array holds each field and its expected message, and a loop checks all 11.
72-96	Data provider with 15 rows: boundaries on name length (1, 2, 40, 41), invalid characters, email formats, phone length and first digit. An empty expected message means "no error expected".
97-104	One method for all 15 rows: fill the field, click Submit, compare the error text.
105-124	Dates come from <code>LocalDate.now()</code> , so the test never goes out of date. <code>fill()</code> works directly on a date input. Boundary: exactly 18 years, one day too young, one day older, and a future date.
125-136	<code>check()</code> on a radio. Selecting another unselects the first. <code>hasCount(1)</code> on the <code>:checked</code> selector proves only one is selected.
137-163	This is a dependent dropdown. <code>selectOption("India")</code> picks by value; <code>setLabel("USA")</code> by visible text; <code>setIndex(0)</code> by position. <code>hasText(String[])</code> checks the whole list of options in order.
164-182	<code>selectOption(new String[]{...})</code> selects many options at once. 2 is valid, 4 is max + 1 (error), and an empty array clears all (error).
183-193	<code>check()</code> and <code>uncheck()</code> are safe because they look at the current state first. A count of <code>:checked</code> boxes proves the result.
194-203	<code>fill("12")</code> works directly on a range input. The shown number is checked, including the maximum 20.
204-224	<code>pressSequentially</code> types key by key. We type 205 characters and the browser keeps only 200 (<code>maxlength</code>). <code>fill()</code> would ignore that limit, so it is not used here.
225-247	<code>setInputFiles(new FilePayload(...))</code> uploads a file made in memory, so no real file is needed. Boundary: exactly 2 MB passes, 2 MB + 1 byte fails. Also a wrong type, an empty file, and a double extension.

248-258 Without ticking the terms an error shows. After `check()` it clears.

259-295 `getByLabel` fills every control. The popup handler is registered BEFORE the click (`onceDialog`), it saves the message and accepts. After the click we check the message with TestNG `Assert`, and the result text with `containsText`.

296-312 Reset opens a confirm popup. `dismiss()` (Cancel) keeps the data. `accept()` (OK) clears the form and disables the State box.

STEP 6: RUN IT

```
mvn test -Dtest="playwrighttests.FormTest"           // run this class only
mvn test -Dtest="playwrighttests.FormTest" -Dheadless=true // run without a browser window
mvn test                                             // run every test
```

▼ Case 3: Complete flow (login to order confirmation)

One user journey: login, search, cart, coupon, checkout, payment and logout.

STEP 1: THE PRACTICE PAGE

Open the page, click around like a manual tester first, then look at the locators below.

Open demo-shop.html

The test already points to this live page (`PAGE_URL`), so it runs from any computer with internet. The same file is in the repository at `docs/demo-pages/demo-shop.html`.

IMPORTANT What you will learn from this case

- An end-to-end test with many steps
- Checking calculations (subtotal, discount, total)
- Negative cases inside a flow
- Finding elements inside a card or table row

STEP 2: WHERE EACH ELEMENT IS (LOCATORS)

ELEMENT	HTML ON THE PAGE	PLAYWRIGHT LOCATOR (BEST)	CSS SELECTOR	XPATH
Login email box	<code><input id="email" type="email"></code>	<code>page.getByLabel("Email")</code>	<code>#email</code>	<code>//input[@id='email']</code>
Login button	<code><button type="button" id="loginBtn">Login</button></code>	<code>page.getByRole(BUTTON, name="Login")</code>	<code>#loginBtn</code>	<code>//button[text()='Login']</code>
Search box	<code><input id="search" placeholder="Search products"></code>	<code>page.getByPlaceholder("Search products")</code>	<code>input[placeholder='Search products']</code>	<code>//input[@placeholder='Search products']</code>
A product card	<code><div class="product" id="product-2"></code>	<code>page.locator("#product-2")</code>	<code>.product</code>	<code>//div[@id='product-2']</code>
Add button inside a card	<code><button id="add-3">Add to cart</button></code>	<code>page.locator("#product-3").getByRole(BUTTON)</code>	<code>#product-3 button</code>	<code>//div[@id='product-3']//button</code>
Cart link	<code><button id="cartLink">Cart (0</button></code>	<code>page.getByRole(BUTTON, name="/Cart/")</code>	<code>#cartLink</code>	<code>//button[contains(text(),'Cart')]</code>
Quantity of an item	<code>2</code>	<code>page.locator("#qty-1")</code>	<code>#cart-row-1 span</code>	<code>//tr[@id='cart-row-1']//span</code>
Remove button	<code><button id="rm-3">Remove</button></code>	<code>page.locator("#rm-3")</code>	<code>#rm-3</code>	<code>//tr[@id='cart-row-3']//button[text()='Remove']</code>

Coupon box	<code><input id="couponInput" aria-label="Coupon code"></code>	<code>page.getByLabel("Coupon code")</code>	<code>#couponInput</code>	<code>//input[@id='couponInput']</code>
Total amount	<code><strong id="total"></code>	<code>page.locator("#total")</code>	<code>#total</code>	<code>//strong[@id='total']</code>
UPI radio button	<code><input type="radio" name="pay" id="payUpi" value="UPI"></code>	<code>page.getByLabel("UPI")</code>	<code>input[name='pay'][value='UPI']</code>	<code>//input[@name='pay' and @value='UPI']</code>
Order id	<code><strong id="orderId"></code>	<code>page.locator("#orderId")</code>	<code>#orderId</code>	<code>//strong[@id='orderId']</code>

STEP 3: MANUAL TEST CASES FIRST

Automation follows manual testing. We first decide what to test, and only then write code.

STEP	ACTION	EXPECTED RESULT	IDEA
Login (4 cases)	Empty fields, wrong password, unknown email, password in wrong case	Correct error message, shop page stays hidden	Negative testing
1	Login with qa@test.com and Test@1234	Welcome text, 3 products shown	Positive
2	Search "pen", "zzz"; filter by category	1 product, "No products found", counts correct	Search and filter
3	Add Notebook twice and Backpack once	Cart count is 3	State change
4	Open the cart	Notebook qty 2, line total 500, subtotal 1700	Calculation check
5	Click + then -	Quantity 3 then back to 2	State change
6	Coupon ABC, then SAVE10	Invalid message; then discount 170, total 1530	Negative then positive

7	Remove Backpack	Subtotal 500, discount 50, total 450	Recalculation
8	Checkout with empty fields, PIN 123, no payment	Three different error messages	Validation
9	Pay with UPI	Order placed, id like ORD-12345, total 450, cart emptied	End result
10	Logout	Back to login page	Flow end
Extra	Open cart with no items	Checkout button disabled	Rule check

STEP 4: THE CODE

File: playwrighttests/ShopFlowTest.java

```

1 package playwrighttests;
2
3 import com.microsoft.playwright.Locator;
4 import com.microsoft.playwright.Page;
5 import com.microsoft.playwright.options.AriaRole;
6 import com.microsoft.playwright.options.SelectOption;
7 import org.testng.Assert;
8 import org.testng.annotations.DataProvider;
9 import org.testng.annotations.Test;
10
11 import java.util.regex.Pattern;
12
13 import static
14     com.microsoft.playwright.assertions.PlaywrightAssertions.assertThat;
15
16 public class ShopFlowTest extends BaseTest {
17     // Live practice page hosted on GitHub Pages (works from any
18     // computer with internet)
19     private static final String PAGE_URL =
20         "https://ubedzz49.github.io/qa-interview-prep/demo-pages/demo-
21         shop.html";
22     private static final String EMAIL = "qa@test.com";
23     private static final String PASSWORD = "Test@1234";
24
25     // ----- Helper methods -----
26     private void login(String email, String password) {
27         page.getByLabel("Email").fill(email);
28         page.getByLabel("Password").fill(password);
29         page.getByRole(AriaRole.BUTTON, new
30             Page.GetByRoleOptions().setName("Login")).click();
31     }
32
33     // "₹1530" -> 1530
34     private int money(String css) {
35         return
36             Integer.parseInt(page.locator(css).innerText().replace("₹", ""));
37     }
38
39     // ----- Login: negative cases -----
40     @DataProvider(name = "badLogins")
41     public Object[][] badLogins() {
42         return new Object[][]{
43             {"both empty", "", ""},
44             {"Email and password are required"},
45             {"wrong password", EMAIL, "wrong"},
46             {"Invalid email or password"},
47             {"unknown email", "nobody@test.com",
48                 PASSWORD, "Invalid email or password"}},

```

```

41         {"password in wrong case", EMAIL,
"test@1234", "Invalid email or password"}},
42     };
43 }
44
45 @Test(dataProvider = "badLogins")
46 public void loginFails(String why, String email, String password,
String message) {
47     page.navigate(PAGE_URL);
48     login(email, password);
49     assertThat(page.locator("#loginError")).hasText(message);
50     assertThat(page.locator("#shopView")).isHidden();
51 }
52
53 // ----- Complete end-to-end flow -----
54 @Test
55 public void endToEndPurchase() {
56     page.navigate(PAGE_URL);
57
58     // 1. Login with valid details
59     login(EMAIL, PASSWORD);
60     assertThat(page.locator("#welcome")).hasText("Welcome,
qa@test.com");
61     assertThat(page.locator(".product")).hasCount(3);
62
63     // 2. Search and filter products
64     Locator search = page.getByPlaceholder("Search products");
65     search.fill("pen");
66     assertThat(page.locator(".product")).hasCount(1);
67     assertThat(page.locator("#product-2 .name")).hasText("Pen
Pack");
68
69     search.fill("zzz");
70     assertThat(page.locator("#noResults")).isVisible();
71
72     search.fill("");
73     page.getByLabel("Category").selectOption("Bags");
74     assertThat(page.locator(".product")).hasCount(1);
75     page.getByLabel("Category").selectOption("All");
76     assertThat(page.locator(".product")).hasCount(3);
77
78     // 3. Add products to the cart
79     page.locator("#add-1").click(); // Notebook
80     page.locator("#add-1").click(); // Notebook
again, so quantity = 2
81     page.locator("#product-3").getByRole(AriaRole.BUTTON).click();
// Backpack: button inside its card
82     assertThat(page.locator("#cartCount")).hasText("3");
83

```

```

84      // 4. Check the cart and the maths
85      page.getByRole(AriaRole.BUTTON, new
Page.GetByRoleOptions().setName(Pattern.compile("Cart"))).click();
86      assertThat(page.locator("#qty-1")).hasText("2");
87      assertThat(page.locator("#cart-row-1 .line-
total"))).hasText("₹500");
88      Assert.assertEquals(money("#subtotal"), 500 + 1200);
89      Assert.assertEquals(money("#discount"), 0);
90      Assert.assertEquals(money("#total"), 1700);
91
92      // 5. Change quantity
93      page.locator("#inc-1").click();
94      assertThat(page.locator("#qty-1")).hasText("3");
95      page.locator("#dec-1").click();
96      assertThat(page.locator("#qty-1")).hasText("2");
97
98      // 6. Coupon: invalid first, then valid
99      page.getByLabel("Coupon code").fill("ABC");
100     page.locator("#applyCoupon").click();
101     assertThat(page.locator("#couponMsg")).hasText("Invalid coupon
code");
102     Assert.assertEquals(money("#discount"), 0);
103
104     page.getByLabel("Coupon code").fill("SAVE10");
105     page.locator("#applyCoupon").click();
106     assertThat(page.locator("#couponMsg")).hasText("Coupon SAVE10
applied: 10% off");
107     Assert.assertEquals(money("#discount"), 170);
108     Assert.assertEquals(money("#total"), 1530);
109
110     // 7. Remove an item and the total updates
111     page.locator("#rm-3").click();           // remove
Backpack
112     assertThat(page.locator("#cart-row-3")).hasCount(0);
113     Assert.assertEquals(money("#subtotal"), 500);
114     Assert.assertEquals(money("#discount"), 50);
115     Assert.assertEquals(money("#total"), 450);
116
117     // 8. Checkout: validation first
118     page.locator("#checkoutBtn").click();
119     page.locator("#placeOrder").click();
120     assertThat(page.locator("#checkoutError")).hasText("Name and
address are required");
121
122     page.getByLabel("Full name").fill("Ravi Kumar");
123     page.getByLabel("Address").fill("12 MG Road, Bengaluru");
124     page.getByLabel("PIN code").fill("123");
125     page.locator("#placeOrder").click();

```

```

126         assertThat(page.locator("#checkoutError")).hasText("Enter a
valid 6 digit PIN code");
127
128         page.getByLabel("PIN code").fill("560001");
129         page.locator("#placeOrder").click();
130         assertThat(page.locator("#checkoutError")).hasText("Select a
payment method");
131
132         // 9. Pay and confirm the order
133         page.getByLabel("UPI").check();
134         page.locator("#placeOrder").click();
135         assertThat(page.locator("#orderStatus")).hasText("Order
placed");
136
137         assertThat(page.locator("#orderId")).hasText(Pattern.compile("^ORD-
\\d{5}$"));
138         assertThat(page.locator("#orderTotal")).hasText("₹450");
139         assertThat(page.locator("#orderPay")).hasText("UPI");
140         assertThat(page.locator("#cartCount")).hasText("0");
141         // cart is emptied
142
143         // 10. Logout returns to the login page
144         page.getByRole(AriaRole.BUTTON, new
Page.GetByRoleOptions().setName("Logout")).click();
145         assertThat(page.locator("#loginView")).isVisible();
146         assertThat(page.locator("#topNav")).isHidden();
147     }
148
149     // ----- Empty cart rule -----
150     @Test
151     public void checkoutIsDisabledForEmptyCart() {
152         page.navigate(PAGE_URL);
153         login(EMAIL, PASSWORD);
154         page.getByRole(AriaRole.BUTTON, new
Page.GetByRoleOptions().setName(Pattern.compile("Cart"))).click();
155         assertThat(page.locator("#emptyCart")).isVisible();
156         assertThat(page.locator("#checkoutBtn")).isDisabled();
157     }
158 }

```

STEP 5: EXPLAINED LINE BY LINE

LINES	WHAT THIS PART DOES
1-14	Imports. <code>Pattern</code> is used for regular expressions.
15-21	Extends <code>BaseTest</code> . The page address and the valid login are constants.

22-29	<code>login()</code> fills the two boxes using labels and clicks the Login button found by role and name.
30-33	<code>money("#total")</code> turns the text "₹1530" into the number 1530 so we can compare numbers with TestNG <code>Assert</code> .
34-44	Four wrong logins: both empty, wrong password, unknown email, and the password in the wrong letter case.
45-52	Each row logs in with bad data, then checks the exact error text and that the shop page stays hidden.
53-57	One long test is one full user journey. The numbered comments are the steps.
58-62	Log in and check the welcome text and 3 products. No explicit wait is needed, <code>assertThat</code> retries.
63-77	<code>getByPlaceholder</code> finds the search box. "pen" shows 1 product; "zzz" shows "No products found". <code>selectOption</code> filters by category.
78-83	Add the Notebook twice. The Backpack button is found INSIDE its card: <code>locator("#product-3").getByRole(BUTTON)</code> (chaining).
84-91	Open the cart (a regular expression matches "Cart (3)"), check quantity 2, the line total, and that subtotal $1700 = 500 + 1200$.
92-97	Plus makes quantity 3, minus brings it back to 2.
98-109	A wrong coupon gives an error and no discount. SAVE10 gives 10% off: discount 170, total 1530.
110-116	Remove the Backpack. <code>hasCount(0)</code> proves the row is gone, and the totals recalculate: 500, 50, 450.
117-131	Three validation messages in order: empty fields, wrong PIN 123, no payment method.
132-140	Choose UPI and place the order. Check the status, the order id with a regular expression (<code>ORD-</code> and 5 digits), the amount, and that the cart is empty.
141-146	Logout returns to the login page and hides the top menu.
147-156	A separate small test: with an empty cart the Checkout button must be disabled.

STEP 6: RUN IT

```

mvn test -Dtest="playwrighttests.ShopFlowTest" // run this
class only
mvn test -Dtest="playwrighttests.ShopFlowTest" -Dheadless=true // run
without a browser window
mvn test // run every test

```

12. Selenium vs Playwright (a favourite interview question)

	SELENIUM	PLAYWRIGHT
Waiting	You write explicit waits yourself	Auto-wait built in
Assertions	TestNG or JUnit, no retry	<code>assertThat(locator)</code> retries until true
Speed	Slower, goes through a driver per browser	Faster, one connection per browser
Isolation	New driver per test (slow)	New context per test (fast, cheap)
Browsers	Chrome, Firefox, Edge, Safari	Chromium, Firefox, WebKit
Languages	Java, Python, C#, JS, Ruby and more	Java, JavaScript / TypeScript, Python, C#
Tools	Needs extra libraries for reports	Trace viewer, video, screenshots, codegen built in
Network control	Limited	Easy: intercept, fake, wait for responses
Mobile apps	With Appium	Only mobile web emulation
Community	Oldest and biggest, most jobs	Newer, growing very fast

13. Quick interview answers

- IMPORTANT** What is Playwright?

An open-source library from Microsoft to automate Chromium, Firefox and WebKit. It supports Java, JavaScript / TypeScript, Python and C#.

- **IMPORTANT Explain Playwright, Browser, BrowserContext and Page.**

Playwright starts everything. Browser is the browser program. BrowserContext is an isolated profile like an incognito window. Page is one tab.

- **IMPORTANT Why is Playwright less flaky than Selenium?**

Auto-waiting before each action, and web-first assertions that retry until true.

- **IMPORTANT What is auto-waiting?**

Before an action, Playwright waits until the element is attached, visible, stable, enabled and not covered.

- **IMPORTANT What is a Locator? Why is it better than a WebElement?**

A Locator is a recipe to find an element, and it searches again on every use, so it never goes stale.

- **IMPORTANT Which locators do you prefer?**

getByRole, getByLabel, getByText, getByTestId first, because they match what users see. CSS and XPath when needed.

- **IMPORTANT What is strict mode?**

If a locator matches more than one element, an action throws an error instead of picking one. Fix with a better locator, first(), or nth().

- **IMPORTANT fill() vs pressSequentially()?**

fill() sets the whole value at once. pressSequentially() types key by key like a user, and respects maxlength.

- **IMPORTANT How do you handle alerts?**

Register `page.onceDialog(...)` BEFORE the action. Without a handler, Playwright dismisses dialogs automatically.

- **IMPORTANT How do you handle frames and new tabs?**

`page.frameLocator("#id")` for frames. `page.waitForPopup(() → click)` for a new tab.

- **IMPORTANT Why a new BrowserContext per test?**

Clean cookies and storage for every test, so tests are independent. Creating a context is very fast.

- **IMPORTANT Selenium vs Playwright?**

See the table above: auto-wait, retrying assertions, speed, built-in trace and video, contexts.

- **GOOD TO KNOW How do you debug a failing test?**

Run headed with `slowMo`, use `page.pause()` and the Inspector, and open the trace

viewer for failures in CI.

- **GOOD TO KNOW** **How do you test the same code on many browsers?**

Change `chromium()` to `firefox()` or `webkit()`, or loop over the three.

- **GOOD TO KNOW** **How do you reuse a login?**

Log in once, save `context.storageState(...)` to a file, and start other contexts from that file.

- **GOOD TO KNOW** **How do you mock an API response?**

```
page.route("**/api/users", route → route.fulfill(...)).
```

- **GOOD TO KNOW** **Can Playwright test APIs?**

Yes, with `playwright.request().newContext()` you can send GET, POST and so on, and check `response.status()`.

- **GOOD TO KNOW** **Does Playwright support parallel tests?**

Yes. With TestNG you can run classes in parallel, and because each test uses its own context they do not disturb each other.