**What are Fixtures in Playwright?**

In Playwright, **Fixtures** are a way to provide pre-configured setups that can be reused across tests. They help you create consistent testing environments, set up browser contexts, pages, and other resources required for tests to run smoothly.

**Fixtures** manage things like:

* Launching the browser.
* Creating contexts and pages.
* Setting up the environment before tests start.
* Cleaning up resources after the tests are completed.

Think of **Fixtures** as predefined, reusable setups for your tests, allowing you to reduce redundant code and make your test setup more efficient and standardized.

**Playwright Built-in Fixtures**

Playwright comes with a built-in test runner, **Playwright Test** (@playwright/test), which provides commonly used fixtures like:

* **browser**: An instance of the browser (e.g., Chromium).
* **context**: A new browser context.
* **page**: A new page created in the given context.

These built-in fixtures simplify the process of setting up browser instances, contexts, and pages, as they are automatically created and managed by Playwright's test runner.

**Browser Fixture (browser, context, page)**

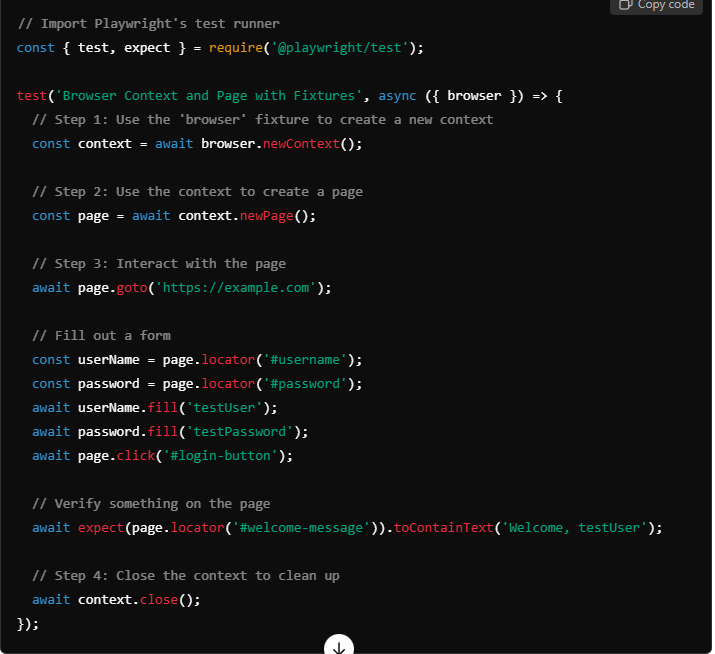
* **browser**: Represents a browser instance (e.g., Chromium, Firefox, WebKit).
  + Used to create browser contexts.
* **context**: Represents an isolated session. All **pages** within a **context** share the session, cookies, and local storage.
  + Used to create pages.
* **page**: Represents a browser tab or page where you interact with a website.
  + Used for navigating URLs, filling forms, clicking buttons, etc.

**Using Fixtures in Playwright Test**

When using Playwright's built-in test runner (@playwright/test), you can directly use these fixtures in your test functions. Let’s see how these fixtures work.

**Example: Using browser, context, and page Fixtures**

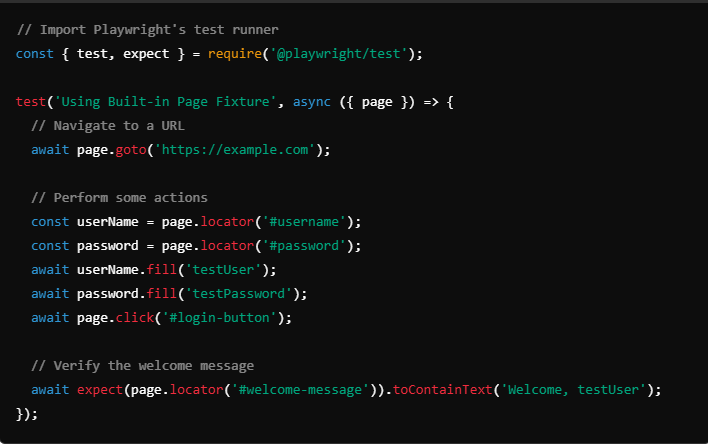
Here is how you can use the built-in fixtures in a Playwright test to set up and manage your browser context and pages:



**Simplified Fixture Example: Using the Built-in page Fixture**

Playwright Test also provides a convenient page fixture that you can use directly in your tests. This fixture is automatically managed, meaning you don't need to manually create or close contexts or pages—Playwright Test handles it for you.

Here’s an example of using the built-in page fixture:



In this example:

* The **page fixture** is automatically created by the Playwright test runner.
* You don’t need to explicitly create a **browser context** or **page**—the page fixture provides a clean **page** instance for each test.
* Playwright Test takes care of the setup and teardown of the page to ensure that every test runs in a clean environment.

**Key Advantages of Using Fixtures:**

1. **Simplified Setup**:
   * With fixtures like page, context, and browser, you don’t have to manually write boilerplate code to set up these components every time you write a test.
2. **Isolation**:
   * Fixtures like context ensure that each test runs in isolation, meaning the session (cookies, storage, etc.) is unique for each test, reducing flakiness.
3. **Automatic Management**:
   * Playwright’s built-in fixtures automatically handle the **setup** and **teardown** process, ensuring that each test has a clean slate to work with.
4. **Reusable Testing Setup**:
   * Fixtures enable you to reuse common test setup configurations, making your test code more modular and easier to maintain.

**Doubt** - Automatic Management: Playwright’s built-in fixtures automatically handle the setup and teardown process, ensuring that each test has a clean slate to work with. - **so we don't need to close page or browser or context seperately?**

That's correct! When you use **Playwright's built-in fixtures**, you do **not** need to manually handle the **setup** or **teardown** (i.e., creation or closure) of the **page**, **context**, or **browser**.

**Automatic Management with Built-in Fixtures**

Playwright Test provides the following fixtures:

* **browser**: A new browser instance.
* **context**: A new browser context.
* **page**: A new page.

When you use these fixtures in your test, Playwright Test takes care of:

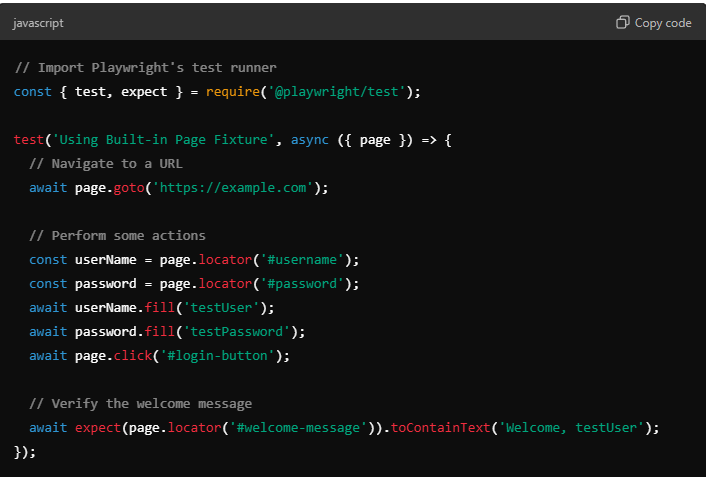
1. **Setting up**: It automatically **creates** the browser, context, and page before the test runs.
2. **Tearing down**: After the test finishes, it automatically **closes** these resources—**page**, **context**, and **browser**.

This automatic management helps in:

* Ensuring **isolation** between tests.
* Preventing resource leaks, like keeping unnecessary browsers open.
* Reducing the code that you need to write for setup and cleanup.

**Example: Built-in Fixture Usage without Manual Closure**

Here's how it works in Playwright Test with the page fixture:



**No Need to Manually Close**

In the example above:

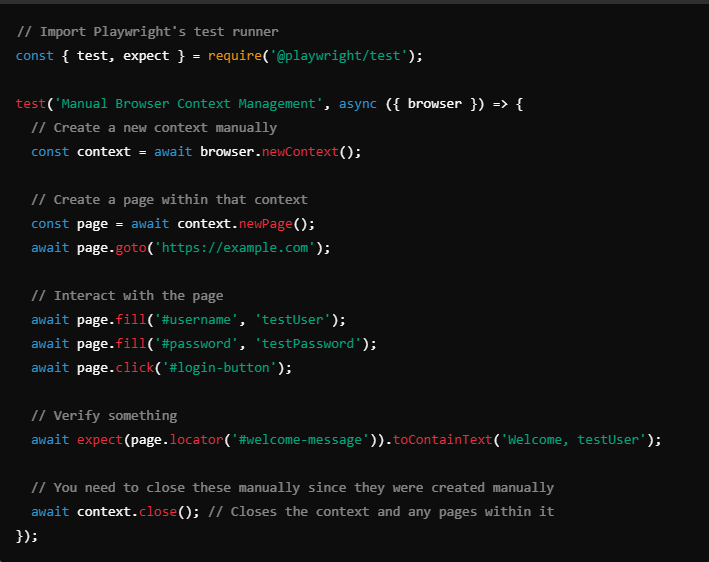
* **page** is a fixture automatically provided by Playwright Test.
* **Playwright Test**:
  + Automatically **creates** a new **browser** and **context** and then **provides** a clean **page** instance at the start of the test.
  + After the test completes, it **automatically closes** the **page**, **context**, and **browser** for you.

This means:

* You **don’t** have to write code like page.close(), context.close(), or browser.close().
* It ensures that all resources are properly closed, avoiding issues with browser instances remaining open.

**Custom Context and Browser Management**

If you create your **own** **context** or **browser** manually (instead of using Playwright’s fixtures), then you will need to close those resources manually to avoid leaks. For example:



Above we are using browser context and so we are closing context manually.

**Why we are not closing page manually in above example?**

In the **manual** setup example, we are indeed using the **browser** fixture, manually creating a **context**, and also manually closing it. Let me clarify the implications and why **closing only the context** works well.

**Playwright's Handling of Pages and Contexts**

* When you manually create a **browser context** (browser.newContext()), the **context** becomes responsible for managing its **pages**.
* If you close a **context**, all **pages** that belong to that context are automatically closed as well.

This means:

* You **do not need to explicitly close each page** within the context. Closing the **context** will close all the **pages** associated with that context.

**Example Explained**

In the provided example:

Here is what happens:

1. **context = await browser.newContext()**:
   * A new **browser context** is created.
   * This context is **independent** and has its own isolated session.
2. **page = await context.newPage()**:
   * A new **page** (i.e., a browser tab) is created inside the context.
   * The **context** is responsible for managing the lifecycle of this page.
3. **await context.close()**:
   * When we close the **context**, it **automatically** closes all the **pages** created in that context.
   * This means there is no need to explicitly call page.close().

**Why Close Only the Context?**

* **Automatic Cleanup**: Closing a **context** will automatically clean up all associated **pages**. This helps reduce code repetition and ensures that nothing is left open accidentally.
* **Cleaner Code**: By closing the **context**, the code remains simple and readable. Explicitly closing every page is unnecessary and could lead to more complex code, especially when there are multiple pages.

**Custom Fixtures**

Besides using the built-in fixtures, you can also create your **custom fixtures** if your tests have specific needs. For instance, you might need a fixture to:

* Set up database connections.
* Load specific data before a test.
* Create reusable login functions.

Here's an example of creating a custom fixture:

|  |
| --- |
| const { test as base, expect } = require('@playwright/test');  // Extend Playwright's base test with a custom fixture  const test = base.extend({  loggedInPage: async ({ browser }, use) => {  // Create a new context and page  const context = await browser.newContext();  const page = await context.newPage();    // Navigate and log in  await page.goto('https://example.com/login');  await page.fill('#username', 'testUser');  await page.fill('#password', 'testPassword');  await page.click('#login-button');    // Use the page in the test  await use(page);    // Close the context after the test  await context.close();  },  });  test('Test using custom loggedInPage fixture', async ({ loggedInPage }) => {  // loggedInPage is already logged in at this point  await expect(loggedInPage.locator('#welcome-message')).toContainText('Welcome, testUser');  }); |

**Summary**

* **Fixtures** in Playwright are pre-configured environments or setups that can be reused across tests.
* Playwright provides built-in fixtures such as browser, context, and page to simplify the setup process.
  + **browser**: Represents the browser instance.
  + **context**: Represents an isolated environment/session for tests.
  + **page**: Represents a page or tab in the browser.
* You can directly use the built-in fixtures to avoid repetitive setup code, making the tests cleaner and more readable.
* Fixtures help in ensuring test **isolation**, **simplicity**, and **consistency** in test execution.
* You can also create **custom fixtures** for more specific testing requirements, which extend the power of Playwright’s test runner.

Using fixtures allows you to write more maintainable, readable, and efficient test scripts. They provide a powerful way to ensure your test setup is always consistent and reliable.