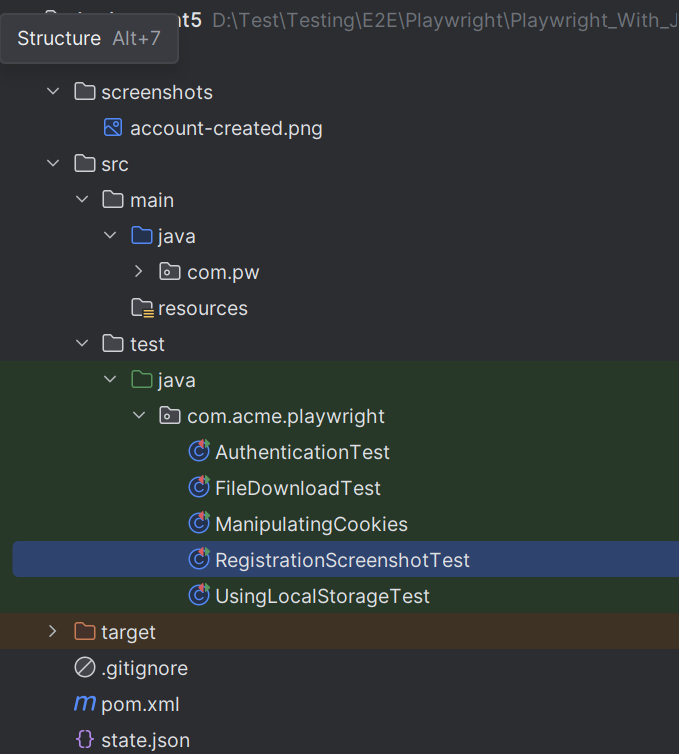
# Playwright for Java - Hands-On Assignment 5

# Project Structure



# Problem Statement 1: Managing Browser Cookies with Playwright Fixtures

# **Objective:** Practice handling cookies using Playwright’s JUnit 5 @UsePlaywright fixtures by adding, retrieving, and clearing cookies in a browser session.

## Steps:

1. Use the @UsePlaywright annotation to enable fixture-based injection of a Page object.
2. Write a test method to:
3. Navigate to the public site URL: <https://playwright.dev/>
4. Retrieve and print existing cookies from the BrowserContext.
5. Add a new cookie (cookie1=abc) for https://playwright.dev/.
6. Print the added cookie details (name, value).
7. Clear all cookies from the context.
8. Verify cookies are removed by printing the empty cookie list.

## Expected Output:

* The console log initially shows an empty or default cookie list.
* After adding, the cookie cookie1=abc is available in the context.
* After clearing, the cookie list is empty again.

# Problem Statement 2: Manipulating Local Storage with Playwright Fixtures

## Objective:

Practice using Playwright’s Java API and page.evaluate() to interact with the browser’s localStorage, ensuring that values can be added, retrieved, cleared, and persist correctly across page reloads.

## Steps:

1. Use the @UsePlaywright annotation to enable Playwright’s built-in JUnit 5 fixture support for automatic Page injection.
2. Write a test method to:
3. Navigate to the given public demo site: <https://www.saucedemo.com/>.
4. Locate the username input field (using getByPlaceholder("Username")) and type "standard\_user".
5. Use JavaScript evaluation (page.evaluate) to add a key-value pair into localStorage (testKey: PlaywrightRocks).
6. Retrieve and print the value from localStorage to confirm it was stored.
7. Clear all items from localStorage and reload the page.
8. Verify that the key testKey is no longer present.
9. Add a new value (testKey: HelloWorld) into localStorage, reload the page, and verify that it persists.
10. Assert that the username field value is reset (empty) after reload.

## Expected Output:

The test passes if:

* The value "PlaywrightRocks" is successfully stored and retrieved from localStorage.
* After clearing, testKey is no longer available.
* The new value "HelloWorld" persists after page reload.
* The username input field is empty after reload.

# Problem Statement 3: File Downloads with Playwright (Java, JUnit)

## Objective:

Modern web applications often allow users to download files such as documents, PDFs, or compressed archives (ZIP files). Playwright provides powerful APIs to capture and verify file downloads programmatically during automated tests.

## Steps:

**Automate a PDF download**

1. Navigate to a public website containing PDF samples like <https://www.princexml.com/samples/>
2. Trigger a PDF download by clicking on a link.

Note: Since PDF files are previewable in browsers, downloads can only be detected in **headless mode**.

**Automate a ZIP download**

1. Navigate to a public website containing ZIP sample files like <https://www.sample-videos.com/download-sample-zip.php>
2. Trigger a ZIP file download by clicking a link or button.

ZIP downloads work in both **headless and headed modes**, as browsers treat them as non-previewable files.

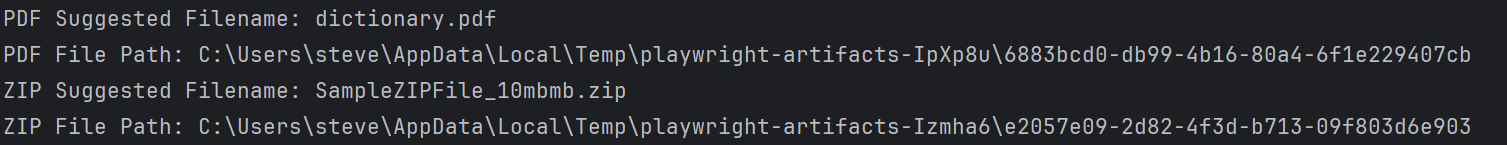
**Validate the downloaded file**

1. Use **JUnit assertions** to ensure the Download object is not null.
2. Verify that the suggested filename ends with the correct extension (.pdf or .zip).
3. Print the file path and suggested filename for confirmation.

**Implementation details**

* Use page.waitForDownload() to wait for the download event.
* Use Playwright locators (locator or getByRole) to click the correct download link.
* Close the browser at the end of each test.

## Expected Output:



# Problem Statement 4: Automating Screenshots with Masking on a Public Registration Form

## Objective:

Use Playwright JUnit 5 fixtures to capture screenshots—including element masking—and validate form submission on a public registration page.

## Steps:

1. Use @Playwright to enable Page injection.

2. Navigate to the LambdaTest registration page:

<https://ecommerce-playground.lambdatest.io/index.php?route=account/register>

3. Fill in the "First Name" field with the value "John" and submit the form.

4. Capture two types of screenshots:

• A basic screenshot of the resulting page.

• An advanced full-page screenshot with all input fields masked in blue.

5. Assert that a success heading (such as "**Your Account Has Been Created!**”) is visible after submission.

## Expected Outcome:

• Screenshots are saved in the folder with filenames:

• The masked screenshot visually hides all form fields in blue.

• A success message confirms that the form submission was successful.

# Problem Statement 5: Persisting Authentication with Playwright Storage State

## Objective

Demonstrate how to persist GitHub login sessions using Playwright’s storage state feature. Automate the login flow, verify successful authentication by checking the visibility of the user's avatar, save the session to a JSON file, and reuse it in a separate test to confirm that the user remains logged in without re-entering credentials.

## Steps

1. Use the annotation to enable Playwright’s JUnit 5 fixture support for automatic injection of Playwright and Browser instances.

2. Launch a Chromium browser in non-headless mode to observe the login flow.

3. Navigate to the GitHub login page.

4. Fill in the username and password fields using the provided credentials.

5. Locate and click the "Sign in" button using a role-based locator with an exact match.

6. After login, verify that the first matching avatar image is visible. This confirms that the user is authenticated and the profile avatar is rendered in the top-right corner.

7. Save the browser context’s storage state to a file named state.json .

8. In a separate test method, create a new browser context using the saved storage state.

9. Navigate to GitHub’s homepage.

10. Confirm that the user remains logged in by asserting the visibility of the first matching avatar image.

## Expected Output

• The first test logs into GitHub and saves the session to a file.

• The second test reuses the saved session and opens GitHub in a logged-in state.

• The avatar is visible in both tests, confirming successful authentication persistence without re-entering credentials.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*