EXPERIMENT-12

AIM: Develop test cases for the above containerized application using selenium.

Objective: To validate the functional, UI, authentication, and integration behavior of the web application running inside Docker by automating end-to-end tests using Selenium (remote WebDriver / Selenium container).

Software Requirements

- Code editor (VS Code or any)
- Docker Desktop
- Python (for running scripts locally if needed)
- Selenium (pip install selenium)
- Web browser (Chrome, Firefox, etc.)

Lab Procedure

Clone this repository to your local repository
https://github.com/Srivaishnavi08/tests

Or follow the below steps

Step-1:

Create a directory named selenium-test and navigate to the current directory path.

tests/

├— Dockerfile

├— index.html

├— SeleniumTest.py

├— docker-compose.yml

Step 2: Create the index.html file

This is your sample web page that Selenium will interact with.

index.html

<!DOCTYPE html>

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Selenium Login Example</title>
</head>
<body>
    <!-- Homepage -->
    <div id="homepage">
        <h1>Welcome!!!!!!</h1>
        <!-- Get Started Free Button -->
        <a href="#loginPage" id="get-started" onclick="navigateToLogin()">Get started</a>
    </div>

    <!-- Login Page -->
    <div id="loginPage" style="display: none;">
        <h2>Login to the Page</h2>
        <form onsubmit="return validateLogin()">
            <label for="user_email_login">Email:</label>
            <input type="email" id="user_email_login" name="user_email_login" required>
            <br><br>
            <label for="user_password">Password:</label>
            <input type="password" id="user_password" name="user_password" required>
            <br><br>
            <button type="submit" name="commit">Login</button>
        </form>
        <p id="error-message" style="color: red; display: none;">Invalid credentials, please try again.</p>
```

```
</div>

<!-- Dashboard Section (only shown after successful login) -->
<div id="dashboard" style="display: none;">
    <h2>Welcome to Your Dashboard!</h2>
    <p>This is the dashboard area you see after a successful login.</p>
</div>

<script>
    // Function to navigate to the login page
    function navigateToLogin() {
        document.getElementById('homepage').style.display = 'none';
        document.getElementById('loginPage').style.display = 'block';
    }

    // Function to validate login credentials
    function validateLogin() {
        const email = document.getElementById('user_email_login').value;
        const password = document.getElementById('user_password').value;

        if (email === "abc@gmail.com" && password === "password") {
            // Hide login page and display dashboard
            document.getElementById('loginPage').style.display = 'none';
            document.getElementById('dashboard').style.display = 'block';
            return false; // Prevent actual form submission
        } else {
            // Show error message if credentials are incorrect
            document.getElementById('error-message').style.display = 'block';
            return false; // Prevent actual form submission
        }
```

```
    }
  </script>
</body>
</html>
```

Step 3: Create the dockerfile

This dockerfile sets up a simple HTTP server to serve your index.html file.

Dockerfile

```
# Use the official Python image as the base image
FROM python:3.9

# Set the working directory in the container
WORKDIR /app

# Copy the index.html file to the container
COPY index.html .

# Expose port 8000 for the HTTP server
EXPOSE 8000

# Start a simple HTTP server to serve the index.html file
CMD ["python", "-m", "http.server", "8000"]
```

Step-4: Create the Selenium Test Script (seleniumTest.py)

This script will automate testing of your HTML page using Selenium.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time
```

```
print("Test Execution Started")
options = webdriver.ChromeOptions()
options.add_argument('--ignore-ssl-errors=yes')
options.add_argument('--ignore-certificate-errors')

# Start the Selenium WebDriver
driver = webdriver.Remote(
    command_executor='http://localhost:4444/wd/hub',
    options=options
)

# Maximize the window size
driver.maximize_window()
time.sleep(10)
driver.get("http://host.docker.internal:8000")  # Access the local server
time.sleep(10)

try:
    # Wait for the "Get started free" link to be clickable
    link = WebDriverWait(driver, 30).until(
        EC.element_to_be_clickable((By.LINK_TEXT, "Get started"))
    )
    link.click()  # Click the link
    time.sleep(10)  # Wait for any resulting page to load

    WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.ID, "user_email_login"))
    )
    WebDriverWait(driver, 10).until(
```

```
        EC.presence_of_element_located((By.ID, "user_password"))
    )

    # Enter login credentials
    username = driver.find_element(By.ID, "user_email_login")
    password = driver.find_element(By.ID, "user_password")
    login_button = driver.find_element(By.NAME, "commit")

    username.send_keys("abc@gmail.com")  # Replace with actual username
    password.send_keys("password")  # Replace with actual password
    login_button.click()

    # Check for a post-login element (adjust to your page's unique element for logged-in users)
    try:
        error_message = WebDriverWait(driver, 10).until(
            EC.visibility_of_element_located((By.ID, "error-message"))
        )
        time.sleep(10)
        print("Login failed: Incorrect credentials")
    except:
        # No error message found, proceed with checking for dashboard
        WebDriverWait(driver, 10).until(
            EC.visibility_of_element_located((By.ID, "dashboard"))  # Replace with actual post-login element ID
        )
        print("Login Successful!")

except Exception as e:
    print(f"An error occurred while trying to click the link: {e}")
```

```
finally:
    # Ensure the browser quits after execution
    driver.quit()
    print("Test Execution Completed!")
```

Step 5: Create the docker-compose.yml File

This file defines two services: your HTML server and the Selenium Chrome container.

(docker-compose.yml)

```
services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: html-server
    ports:
      - "8000:8000"

  selenium:
    image: selenium/standalone-chrome
    container_name: selenium-chrome
    ports:
      - "4444:4444"
    depends_on:
      - app
```

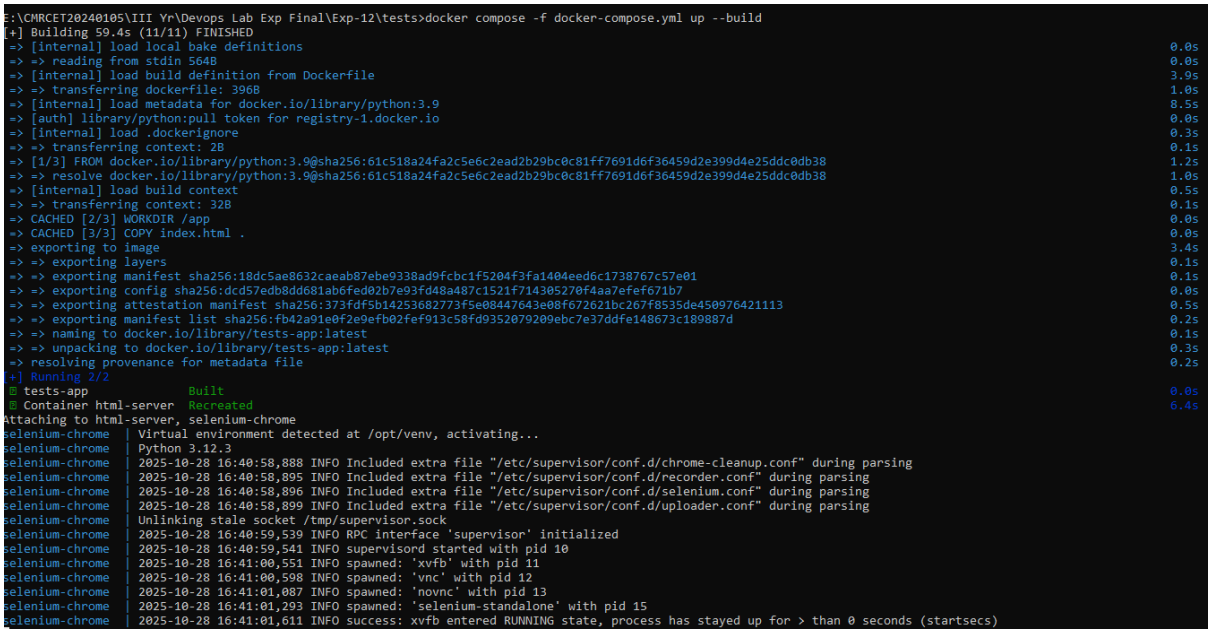
Step-6:

- Build and Run Your Docker Containers
- In terminal use the command

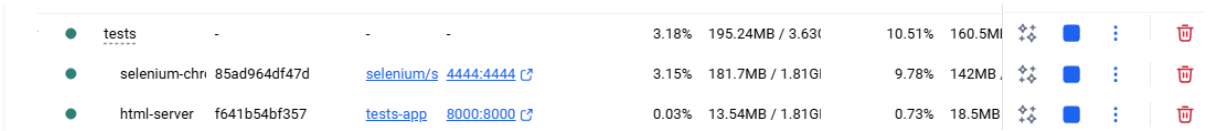
docker compose –f docker-compose up --build

This command will:

- Build the Docker image for your HTML server.
- Pull the Selenium standalone Chrome image.



- Start both services.
- Press **v** to navigate to docker.



- Click on the links

8000:8000

4444:4444

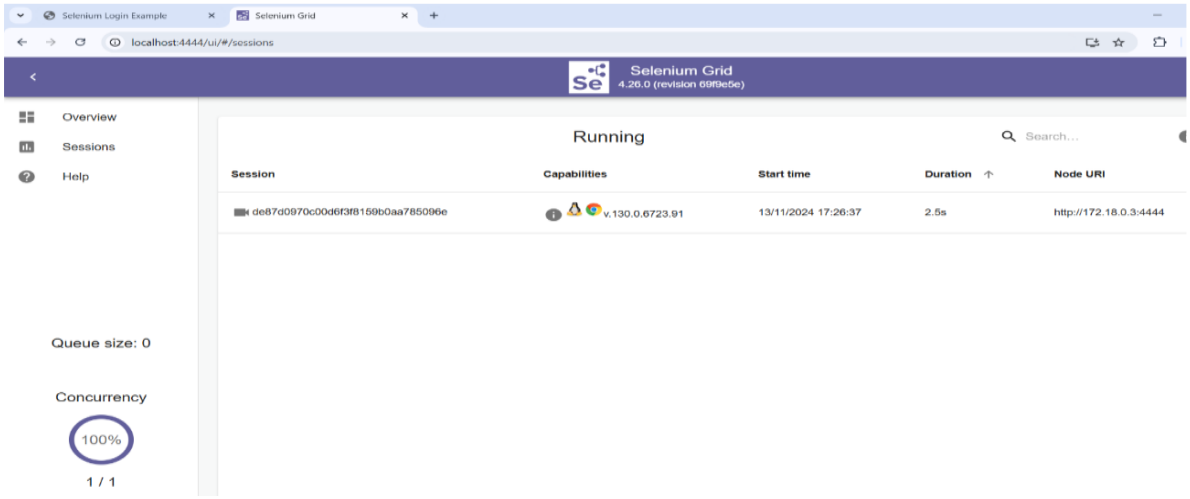
Step 7:

Run the Selenium Test Script

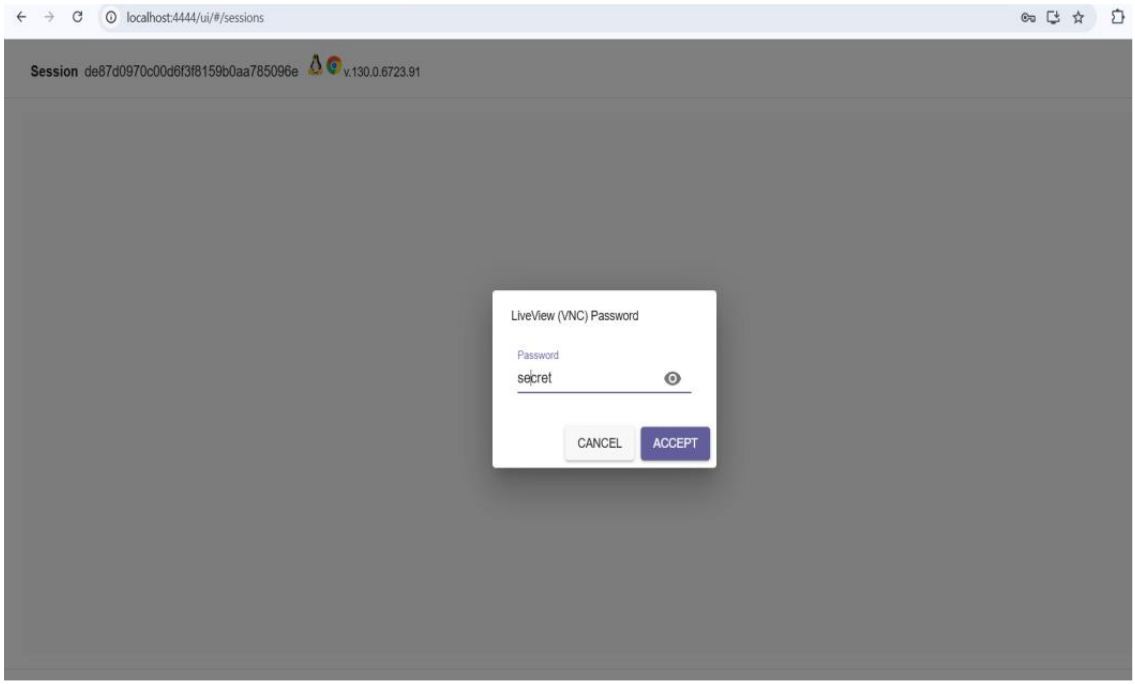
While your containers are running, open a new terminal and run:

python seleniumTest.py

Go to sessions, click on video icon.



Enter the password “secret” to view the video of automatic testing.



Step 8:

Stop the Docker Containers Once you are done testing.

Stop the containers using: **docker compose –f docker-compose down**

This command stops and removes the containers, networks, and any volumes associated with theservices.

After running the test and clicking the video icon, the video will show how the test interacts with the web page (clicking buttons, entering passwords, etc.)