header_navigation_test.py

```python
#
#  Course: COSC 4P02
#  Assignment: Group Project
#  Group: 9
#  Version: 1.0
#  Date: April 2024
#
from selenium import webdriver # Import the webdriver module.
from selenium.webdriver.chrome.options import Options as ChromeOptions # Import the ChromeOptions class.
from selenium.webdriver.chrome.service import Service as ChromeService # Import the ChromeService class.
from selenium.webdriver.common.by import By  # Import the By class for locating elements.
from webdriver_manager.chrome import ChromeDriverManager # Import the ChromeDriverManager for managing ChromeDriver binaries.
import pytest # Import the pytest module for testing.
import time # Import the time module for sleep functionality.

URL = "https://group9portal-eehbdxbhcgftezez.canadaeast-01.azurewebsites.net/index.php" # Our website URL to be tested.

@pytest.fixture(scope="module")
def browser():
    #
    # Fixture Browser:
    # This fixture provides a browser instance using Selenium WebDriver. It uses the Chrome browser in headless mode for testing (there is a line that can be commented out to view the GUI). This fixture is automatically invoked by
    # pytest when a test function includes it as an argument. It allows test functions to interact with the browser and perform actions like navigating to URLs, finding elements, and executing JavaScript.
    # The client object is available for interacting with the app's elements, and it will be cleaned up after the test.
    #
    options = ChromeOptions() # Create an instance of ChromeOptions to configure the Chrome browser.
    options.add_argument("--headless")  # Run Chrome in headless mode (without a GUI). If this line is commented out, the browser will open in a GUI mode. The test moves very fast in headless mode, but it does show the site.
    service = ChromeService(executable_path=ChromeDriverManager().install()) # Create an instance of ChromeService to manage the ChromeDriver executable.
    driver = webdriver.Chrome(service=service, options=options) # Create an instance of the Chrome WebDriver with the specified service and options.
    yield driver # Yield the driver instance to the test function.
    driver.quit() # Quit the driver after the test function completes.


def login(browser, username="testcase2", password="MNBVCXZ1234567890!@#$%^&*()"):
    #
```

```
    # A helper function to log in if the user is not already logged in. This function checks if the user is logged in by looking for the "Logout" button.
    # If the user is not logged in, it performs the login process. This method is used to avoid duplicating the login code across multiple test cases.
    #
    if len(browser.find_elements(By.LINK_TEXT, "Logout")) == 0: # Check if the "Logout" button is present to determine if the user is logged in.
        browser.find_element(By.LINK_TEXT, "Login").click() # Find the "Login" link element and click it to navigate to the login page.
        time.sleep(1) # Wait for the page to load after clicking the "Login" link.
        browser.find_element(By.NAME, "username").send_keys(username) # Enter the username in the username field. I've created a test user with both username and password as "testcase".
        browser.find_element(By.NAME, "password").send_keys(password) # Enter the password in the password field. I've created a test user with both username and password as "testcase".
        browser.find_element(By.CSS_SELECTOR, "button[type='submit']").click()  # Find the submit button using CSS selector and click it to log in.
        time.sleep(1) # Wait for the page to load after clicking the submit button.

def test_header_present(browser):
    #
    # Test Case 1: Testing the presence of the header. This will confirm that the header is present on the page for navigation.
    # Execution: python -m pytest header_navigation_test.py -k "test_header_present" -s -v # Only use -s to view the messages in the test.
    # This method will check if the header is present on the page. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL
    # and check for the presence of the buttons by the text of the buttons.
    # Expected Result: Pass. The header should be present on the page.
    #
    browser.get(URL) # Navigate to the specified URL in our browser instance.
    assert browser.find_element(By.LINK_TEXT, "Dashboard").is_displayed() # Check that the "Dashboard" link is displayed.
    assert browser.find_element(By.LINK_TEXT, "Generate").is_displayed() # Check that the "Generate" link is displayed.
    assert browser.find_element(By.LINK_TEXT, "About Us").is_displayed() # Check that the "About Us" link is displayed.
    assert browser.find_element(By.LINK_TEXT, "FAQ").is_displayed() # Check that the "FAQ" link is displayed.
    assert browser.find_element(By.LINK_TEXT, "Login").is_displayed() # Check that the "Login" link is displayed.
    assert browser.find_element(By.LINK_TEXT, "Register").is_displayed() # Check that the "Register" link is displayed.

def test_header_present_after_login(browser):
    #
    # Test Case 2: Testing the presence of the header after login. This will confirm that the header is present on the page for navigation after login, and the proper pages have appeared/disappeared.
    # Execution: python -m pytest header_navigation_test.py -k "test_header_present_after_login" -s -v # Only use -s to view the messages in the test.
    # This method will check if the header is present on the page after login, and check that the proper pages have appeared/disappeared. It uses the Selenium WebDriver (predefined as a
```

```
fixture above)
    # to navigate to our URL and check for the presence of the buttons by the text of the buttons.
    # Expected Result: Pass. The header should be present on the page after login with the correct pages and buttons.
    #
    browser.get(URL) # Navigate to the specified URL in our browser instance.
    login(browser) # Call the login function to log in if not already logged in.

    assert browser.find_element(By.LINK_TEXT, "Dashboard").is_displayed() # Check that the "Dashboard" link is displayed.
    assert browser.find_element(By.LINK_TEXT, "Generate").is_displayed() # Check that the "Generate" link is displayed.
    assert browser.find_element(By.LINK_TEXT, "About Us").is_displayed() # Check that the "About Us" link is displayed.
    assert browser.find_element(By.LINK_TEXT, "FAQ").is_displayed() # Check that the "FAQ" link is displayed.
    assert browser.find_element(By.LINK_TEXT, "Profile").is_displayed() # Check that the "Profile" link is displayed.
    assert browser.find_element(By.LINK_TEXT, "Logout").is_displayed() # Check that the "Logout" link is displayed.
    register_button = browser.find_elements(By.LINK_TEXT, "Register") # Find the "Register" link element. There should be none after login.
    assert len(register_button) == 0 # Check that the "Register" link is not displayed after login.
    login_button1 = browser.find_elements(By.LINK_TEXT, "Login") # Find the "Login" link element. There should be none after login.
    assert len(login_button1) == 0 # Check that the "Login" link is not displayed after login.

def test_header_present_after_logout(browser):
    #
    # Test Case 3: Testing the presence of the header after logout. This will confirm that the header is present on the page for navigation after logout, and the proper pages have appeared/disappeared.
    # Execution: python -m pytest header_navigation_test.py -k "test_header_present_-after_logout" -s -v # Only use -s to view the messages in the test.
    # This method will check if the header is present on the page after logout, and check that the proper pages have appeared/disappeared. It uses the Selenium WebDriver (predefined as a fixture above)
    # to navigate to our URL and check for the presence of the buttons by the text of the buttons.
    # Expected Result: Pass. The header should be present on the page after logout with the correct pages and buttons.
    #
    browser.get(URL)  # Navigate to the home page
    logged_in = len(browser.find_elements(By.LINK_TEXT, "Logout")) > 0 # Check if the "Logout" button is present to determine if the user is logged in.
    if not logged_in: # If the user is not logged in, log in first.
        login(browser) # Call the login function.
    logout_button = browser.find_element(By.LINK_TEXT, "Logout") # Find the "Logout" link element.
```

```
    logout_button.click() # Click the "Logout" link to log out.
    time.sleep(1) # Wait for the page to load after logout.

    assert browser.find_element(By.LINK_TEXT, "Dashboard").is_displayed() # Check that the "Dashboard" link is displayed.
    assert browser.find_element(By.LINK_TEXT, "Generate").is_displayed() # Check that the "Generate" link is displayed.
    assert browser.find_element(By.LINK_TEXT, "About Us").is_displayed() # Check that the "About Us" link is displayed.
    assert browser.find_element(By.LINK_TEXT, "FAQ").is_displayed() # Check that the "FAQ" link is displayed.
    assert browser.find_element(By.LINK_TEXT, "Login").is_displayed() # Check that the "Login" link is displayed.
    assert browser.find_element(By.LINK_TEXT, "Register").is_displayed() # Check that the "Register" link is displayed.
    assert len(browser.find_elements(By.LINK_TEXT, "Profile")) == 0 # Check that the "Profile" link is not displayed after logout.
    assert len(browser.find_elements(By.LINK_TEXT, "Logout")) == 0 # Check that the "Logout" link is not displayed after logout.

def test_header_functionality(browser):
    #
    # Test Case 4: Testing the functionality of the header buttons. This will confirm that the header buttons are functional and navigate to the correct pages.
    # Execution: python -m pytest header_navigation_test.py -k "test_header_functionality" -s -v # Only use -s to view the messages in the test.
    # This method will check if the header buttons are functional and navigate to the correct pages. It uses the Selenium WebDriver (predefined as a fixture above)
    # to navigate to our URL and check for the presence of elements relevant to the different pages of our site.
    # Expected Result: Pass. The header buttons should be functional and navigate to the correct pages.
    #
    browser.get(URL) # Navigate to the specified URL in our browser instance.
    browser.find_element(By.LINK_TEXT, "Dashboard").click() # Check that the "Dashboard" link is displayed and click it to navigate to the dashboard page.
    time.sleep(1) # Wait for the page to load after clicking the "Dashboard" link.
    assert "index.php" in browser.current_url # Check that the current URL contains "index.php" (the dashboard page).

    browser.find_element(By.LINK_TEXT, "Generate").click() # Check that the "Generate" link is displayed and click it to navigate to the generate page.
    time.sleep(1) # Wait for the page to load after clicking the "Generate" link.
    assert "generate_page.php" in browser.current_url # Check that the current URL contains "generate_page.php" (the generate page).
    assert "Generate Post" in browser.page_source # Check that the page contains "Generate Post" (the title of the generate page).
    assert browser.find_element(By.XPATH, "//button[contains(text(), 'Generate')]").is_displayed() # Check that the "Generate" button is displayed on the generate page.
    assert browser.find_element(By.XPATH, "//button[contains(text(), 'Generate')]").is_enabled() # Check that the "Generate" button is enabled on the generate page.
```

```
129
130     browser.find_element(By.LINK_TEXT, "About Us").click() # Check that the "About Us" link is displayed and click it to navigate to the about us page.
131     time.sleep(1) # Wait for the page to load after clicking the "About Us" link.
132     assert "about_us.php" in browser.current_url # Check that the current URL contains "about_us.php" (the about us page).
133     assert "Project Inspiration &amp;amp; Background" in browser.page_source # Check that the page contains "Project Inspiration & Background" (the section title on the about us page).
134     assert "Objectives & Goals" in browser.page_source # Check that the page contains "Objectives & Goals" (the section title on the about us page).
135     assert "Technologies & Methodologies" in browser.page_source # Check that the page contains "Technologies & Methodologies" (the section title on the about us page).
136     assert "Future Enhancements" in browser.page_source # Check that the page contains "Future Enhancements" (the section title on the about us page).
137
138     browser.find_element(By.LINK_TEXT, "FAQ").click() # Check that the "FAQ" link is displayed and click it to navigate to the FAQ page.
139     time.sleep(1) # Wait for the page to load after clicking the "FAQ" link.
140     assert "faq.php" in browser.current_url # Check that the current URL contains "faq.php" (the FAQ page).
141     assert "Frequently Asked Questions" in browser.page_source # Check that the page contains "Frequently Asked Questions" (the title of the FAQ page).
142     assert "Below you'll find answers to some of the most common questions we receive. If you need further assistance, feel free to reach out to us." in browser.page_source # Check that the page contains the introductory text on the FAQ page.
143
144     logged_in = len(browser.find_elements(By.LINK_TEXT, "Logout")) > 0 # Check if the "Logout" button is present to determine if the user is logged in.
145     if not logged_in: # If the user is not logged in:
146         browser.find_element(By.LINK_TEXT, "Login").click() # Find the "Login" link element and click it to navigate to the login page.
147         time.sleep(1) # Wait for the page to load after clicking the "Login" link.
148         assert "login_pageNew.php" in browser.current_url # Check that the current URL contains "login_pageNew.php" (the login page).
149         browser.find_element(By.NAME, "username") # Check that the username field is present on the login page.
150         browser.find_element(By.NAME, "password") # Check that the password field is present on the login page.
151         browser.find_element(By.CSS_SELECTOR, "button[type='submit']") # Check that the submit button is present on the login page.
152
153         browser.find_element(By.LINK_TEXT, "Register").click() # Find the "Register" link element and click it to navigate to the register page.
154         time.sleep(1) # Wait for the page to load after clicking the "Register" link.
155         assert "register_pageNew.php" in browser.current_url # Check that the current URL contains "register_pageNew.php" (the register page).
156         browser.find_element(By.NAME, "email") # Check that the email field is present on the register page.
157         browser.find_element(By.NAME, "username") # Check that the username field is present on the register page.
158         browser.find_element(By.NAME, "password") # Check that the password field is present on the register page.
```

```
159         browser.find_element(By.CSS_SELECTOR, "button[type='submit']") # Check that the
submit button is present on the register page.
160
161         login(browser) # Call the login function to log in.
162         browser.find_element(By.LINK_TEXT, "Profile").click() # Find the "Profile" link
element and click it to navigate to the profile page.
163         time.sleep(2) # Wait for the page to load after clicking the "Profile" link.
164         assert "profile_page.php" in browser.current_url # Check that the current URL
contains "profile_page.php" (the profile page).
165         assert "First Name:" in browser.page_source # Check that the page contains "First
Name:" (the label for the first name field on the profile page).
166         assert "Last Name:" in browser.page_source  # Check that the page contains "Last
Name:" (the label for the last name field on the profile page).
167         assert "Username:" in browser.page_source # Check that the page contains "Username:"
(the label for the username field on the profile page).
168         assert "Email:" in browser.page_source # Check that the page contains "Email:" (the
label for the email field on the profile page).
169         assert "Content Generation Frequency:" in browser.page_source # Check that the page
contains "Content Generation Frequency:" (the label for the content generation frequency
field on the profile page).
170         assert "Generation Time:" in browser.page_source # Check that the page contains
"Generation Time:" (the label for the generation time field on the profile page).
171
```