edit_button_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.
from selenium.webdriver.support.ui import WebDriverWait # Import the WebDriverWait class for waiting for elements to be present.
from selenium.webdriver.support import expected_conditions as EC # Import expected_conditions for waiting conditions.
from selenium.webdriver.common.alert import Alert # Import the Alert class for handling JavaScript alerts.
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 test_editButtonsPresent(browser):
    #
    # Test Case 1: Testing the presence of the edit button(s). This will confirm that the button(s) are present on the page.
    # Execution: python -m pytest edit_button_test.py -k "test_editButtonsPresent" -s -v # Only use -s to view the messages in the test.
    # This method will check if the edit button(s) are 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 edit button(s) by their class names.
    # Expected Result: Pass. The edit button(s) should be present on the page.
    #
    browser.get(URL) # Navigate to the specified URL in our browser instance.
    edit_buttons = browser.find_elements(By.CLASS_NAME, "edit-btn") # Find all elements with the class name "edit-btn" (the edit button) on the page.
    assert len(edit_buttons) > 0, "Edit button(s) not found on the page." # Assert that at least one edit button is present. If not, raise an AssertionError with the message.
    print(f"Number of edit buttons found: {len(edit_buttons)}") # Print the number of edit buttons found on the page.

def test_editButtonsFunctionality(browser):
    #
    # Test Case 2: Testing the functionality of the edit button(s). This will confirm that the button(s) work as intended.
    # Execution: python -m pytest edit_button_test.py -k "test_editButtonsFunctionality" -s -v # Only use -s to view the messages in the test.
    # This method will check if the edit button(s) work properly. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL
    # and check that the edit button(s) function as expected when clicked. It will click the first edit button, edit the post content, and then save the changes.
    # Expected Result: Pass. The edit button(s) should work properly.
    #
    browser.get(URL)  # Navigate to the specified URL in our browser instance.
    edit_buttons = browser.find_elements(By.CLASS_NAME, "edit-btn")  # Find all edit buttons on the page.
    assert edit_buttons, "No edit buttons found" # Assert that at least one edit button is present.
    first_edit = edit_buttons[0]  # Select the first edit button from the list of edit buttons.
    post_card = first_edit.find_element(By.XPATH, "./ancestor::div[contains(@class, 'card')]")  # Find the parent card element of the edit button.
    original_content = post_card.find_element(By.XPATH, ".//p").text  # Get the original content of the post.
    first_edit.click()  # Click the first edit button to open the edit mode.
    textarea = post_card.find_element(By.TAG_NAME, "textarea")  # Find the textarea inside the card.
    assert textarea.is_displayed()  # Assert that the textarea is displayed.
    new_content = original_content + " (edited)"  # Create new content by appending " (edited)".
    textarea.clear()  # Clear existing content in the textarea.
    textarea.send_keys(new_content)  # Enter new content into the textarea.
    save_button = post_card.find_element(By.XPATH, ".//button[text()='save']")  # Find the save button inside the card.
    save_button.click()  # Click the save button to save the changes.
    WebDriverWait(browser, 10).until(EC.text_to_be_present_in_element((By.XPATH, ".//p"), new_content)) # Wait for the new content to be present in the post card.
    post_card2 = first_edit.find_element(By.XPATH, "./ancestor::div[contains(@class, 'card')]") # Find the parent card element of the edit button again.
    updated_content = post_card2.find_element(By.XPATH, ".//p").text  # Get the updated content of the post.
    assert updated_content == new_content # Assert that the updated content matches the new content.

def test_cancelButtonFunctionality(browser):
    #
    # Test Case 3: Testing the functionality of the cancel button. This will confirm that the cancel button works as intended.
    # Execution: python -m pytest edit_button_test.py -k "test_cancelButtonFunctionality" -s -v # Only use -s to view the messages in the test.
    # This method will check if the cancel button works properly. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL
    # and check that the cancel button functions as expected when clicked. It will click the first edit button, edit the post content, and then cancel the changes.
    # Expected Result: Pass. The cancel button should work properly. The content should not be changed after canceling.
    #
    browser.get(URL)  # Navigate to the page.
    edit_buttons = browser.find_elements(By.CLASS_NAME, "edit-btn") # Find all edit buttons on the page.
    assert edit_buttons, "No edit buttons found" # Assert that at least one edit button is present.
    first_edit = edit_buttons[0] # Select the first edit button from the list of edit buttons.
    post_card = first_edit.find_element(By.XPATH, "./ancestor::div[contains(@class, 'card')]") # Find the parent card element of the edit button.
    original_content = post_card.find_element(By.XPATH, ".//p").text # Get the original content of the post.
    first_edit.click() # Click the first edit button to open the edit mode.
    textarea = post_card.find_element(By.TAG_NAME, "textarea") # Find the textarea inside the card.
    assert textarea.is_displayed() # Assert that the textarea is displayed.
    new_content = original_content + " (edited)" # Create new content by appending " (edited)".
    textarea.clear()  # Clear any existing content in the textarea.
    textarea.send_keys(new_content)  # Enter the new content into the textarea.
    cancel_button = post_card.find_element(By.XPATH, ".//button[text()='cancel']") # Find the cancel button inside the card.
    cancel_button.click() # Click the cancel button to discard changes.
    post_card2 = first_edit.find_element(By.XPATH, "./ancestor::div[contains(@class, 'card')]") # Find the parent card element of the edit button again.
    updated_content = post_card2.find_element(By.XPATH, ".//p").text # Get the updated content of the post after canceling.
    assert updated_content == original_content # Assert that the updated content matches the original content after canceling.

def test_editCancelMultiplePosts(browser):
    #
    # Test Case 4: Testing the functionality of editing multiple posts. This will confirm that only the edited post is updated, while the other remains unchanged.
    # Execution: python -m pytest edit_button_test.py -k "test_editCancelMultiplePosts" -s -v # Only use -s to view the messages in the test.
    # This method will check if the edit button(s) work properly when editing multiple posts. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL
    # and check that the edit button(s) function as expected when clicked. It will click the first edit button, edit the post content, and then save the changes. It will also click the second edit button,
    # but not make any changes to the post content.
    # Expected Result: Pass. The edited post should be updated, while the other post should remain unchanged.
    #
    browser.get(URL) # Navigate to the specified URL in our browser instance.
    cards = browser.find_elements(By.CLASS_NAME, "card") # Find all post cards on the page.
    assert len(cards) >= 2, "Not enough post cards found" # Assert that at least two post cards are present.
    post_card = cards[0] # Select the first post card from the list of cards.
    post_card2 = cards[1] # Select the second post card from the list of cards.
    first_edit = post_card.find_element(By.CLASS_NAME, "edit-btn") # Find the edit button inside the first post card.
    second_edit = post_card2.find_element(By.CLASS_NAME, "edit-btn") # Find the edit button inside the second post card.
    original_content = post_card.find_element(By.XPATH, ".//p").text # Get the original content of the first post.
    original_content2 = post_card2.find_element(By.XPATH, ".//p").text # Get the original content of the second post.
    assert original_content != original_content2 # Assert that the contents of the two posts are different.
    first_edit.click() # Click the edit button of the first post to open the edit mode.
    second_edit.click() # Click the edit button of the second post to open the edit mode.
    textarea = post_card.find_element(By.TAG_NAME, "textarea") # Find the textarea inside the first post card.
    textarea2 = post_card2.find_element(By.TAG_NAME, "textarea") # Find the textarea inside the second post card.
    assert textarea.is_displayed() # Assert that the textarea of the first post is displayed.
    assert textarea2.is_displayed() # Assert that the textarea of the second post is displayed.
    new_content = original_content + " (edited)" # Create new content for the first post by appending "(edited)".
    textarea.clear() # Clear any existing content in the textarea of the first post.
    textarea.send_keys(new_content) # Enter the new content into the textarea of the first post.
    post_card.find_element(By.XPATH, ".//button[text()='save']").click() # Click the save button of the first post to save the changes.
    WebDriverWait(browser, 10).until(EC.text_to_be_present_in_element((By.XPATH, f"(//div[contains(@class, 'card')])[1]//p"), new_content)) # Wait for the new content to be
```

```
present in the first post card.
    post_card2.find_element(By.XPATH, ".//button[text()='cancel']").click() # Click the cancel button of the second post to discard changes.
    updated_content = post_card.find_element(By.XPATH, ".//p").text # Get the updated content of the first post after saving changes.
    updated_content2 = post_card2.find_element(By.XPATH, ".//p").text # Get the updated content of the second post after canceling changes.
    assert updated_content == new_content # Assert that the updated content of the first post matches the new content.
    assert updated_content2 == original_content2 # Assert that the updated content of the second post matches the original content after canceling changes. Nothing has changed.

def test_blankEdit(browser):
    #
    # Test Case 5: Testing the functionality of the save button when the textarea is blank. This will confirm that the button does not save when the textarea is blank.
    # Execution: python -m pytest edit_button_test.py -k "test_blankEdit" -s -v # Only use -s to view the messages in the test.
    # This method will check if the save button works properly when the textarea is blank. It uses the Selenium WebDriver (predefined as a fixture above) to navigate to our URL
    # and check that the save button functions as expected when clicked. It will click the first edit button, clear the post content, and then save the changes.
    # Expected Result: Pass. The save button should not save the changes when the textarea is blank.
    #
    browser.get(URL)  # Navigate to the specified URL in our browser instance.
    edit_buttons = browser.find_elements(By.CLASS_NAME, "edit-btn")  # Find all edit buttons on the page.
    assert edit_buttons, "No edit buttons found" # Assert that at least one edit button is present.
    first_edit = edit_buttons[0]  # Select the first edit button from the list of edit buttons.
    post_card = first_edit.find_element(By.XPATH, "./ancestor::div[contains(@class, 'card')]")  # Find the parent card element of the edit button.
    original_content = post_card.find_element(By.XPATH, ".//p").text  # Get the original content of the post.
    first_edit.click()  # Click the first edit button to open the edit mode.
    textarea = post_card.find_element(By.TAG_NAME, "textarea")  # Find the textarea inside the card.
    assert textarea.is_displayed()  # Assert that the textarea is displayed.
    textarea.clear()  # Clear existing content in the textarea.
    save_button = post_card.find_element(By.XPATH, ".//button[text()='save']")  # Find the save button inside the card.
    save_button.click()  # Click the save button to save the changes.
    WebDriverWait(browser, 5).until(EC.alert_is_present()) # Wait for the alert to be present.
    alert = Alert(browser) # Create an Alert object to handle the alert.
    assert alert.text == "Edit cannot be empty." # Assert that the alert message is as expected.
    alert.accept() # Accept the alert to close it.
    cancel_button = post_card.find_element(By.XPATH, ".//button[text()='cancel']") # Find the cancel button inside the card.
```

```python
    cancel_button.click() # Click the cancel button to discard changes.
    post_card2 = first_edit.find_element(By.XPATH, "./ancestor::div[contains(@class, 'card')]") # Find the parent card element of the edit button again.
    updated_content = post_card2.find_element(By.XPATH, ".//p").text # Get the updated content of the post after canceling. It should be the same as the original content.
    assert updated_content == original_content # Assert that the updated content matches the original content after canceling.
```