ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки
09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 4

Дисциплина: Тестирование программного обеспечения

Тема: Автоматизация позитивных и негативных UI-сценариев

Выполнил(а): студент(ка) группы: 221-3711

До Дык Зунг

(Фамилия И.О.)

Дата, подпись_____ _____
(Дата) (Подпись)

Проверил:_____ _____
(Фамилия И.О., степень, звание) (Оценка)

Дата, подпись_____ _____
(Дата) (Подпись)

Замечания: _____

_____

_____

Москва

2025

Задание (Автоматическое тестирование):

1. **Реализуйте базовый класс `BasePage` и класс `ContactPage` по паттерну Page Object.**

BasePage.py

```
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

class BasePage:
    def __init__(self, driver):
        self.driver = driver
        self.wait = WebDriverWait(driver, 10)

    def find_element(self, by, value):
        return self.wait.until(EC.presence_of_element_located((by, value)))

    def send_keys(self, by, value, text):
        element = self.find_element(by, value)
        element.clear()
        element.send_keys(text)

    def click(self, by, value):
        element = self.find_element(by, value)
        element.click()

    def get_error_message(self, by, value):
        return self.find_element(by, value).text
```

ContactPage.py

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

class BasePage:
    def __init__(self, driver):
        self.driver = driver
        self.wait = WebDriverWait(driver, 10)

    def find_element(self, by, value):
        return self.wait.until(EC.presence_of_element_located((by, value)))

    def send_keys(self, by, value, text):
        element = self.find_element(by, value)
        element.clear()
        element.send_keys(text)

    def click(self, by, value):
        element = self.find_element(by, value)
        element.click()

    def get_error_message(self, by, value):
        return self.find_element(by, value).text

class ContactPage(BasePage):
    def __init__(self, driver):
        super().__init__(driver)
        self.name_field = (By.ID, "name")
```

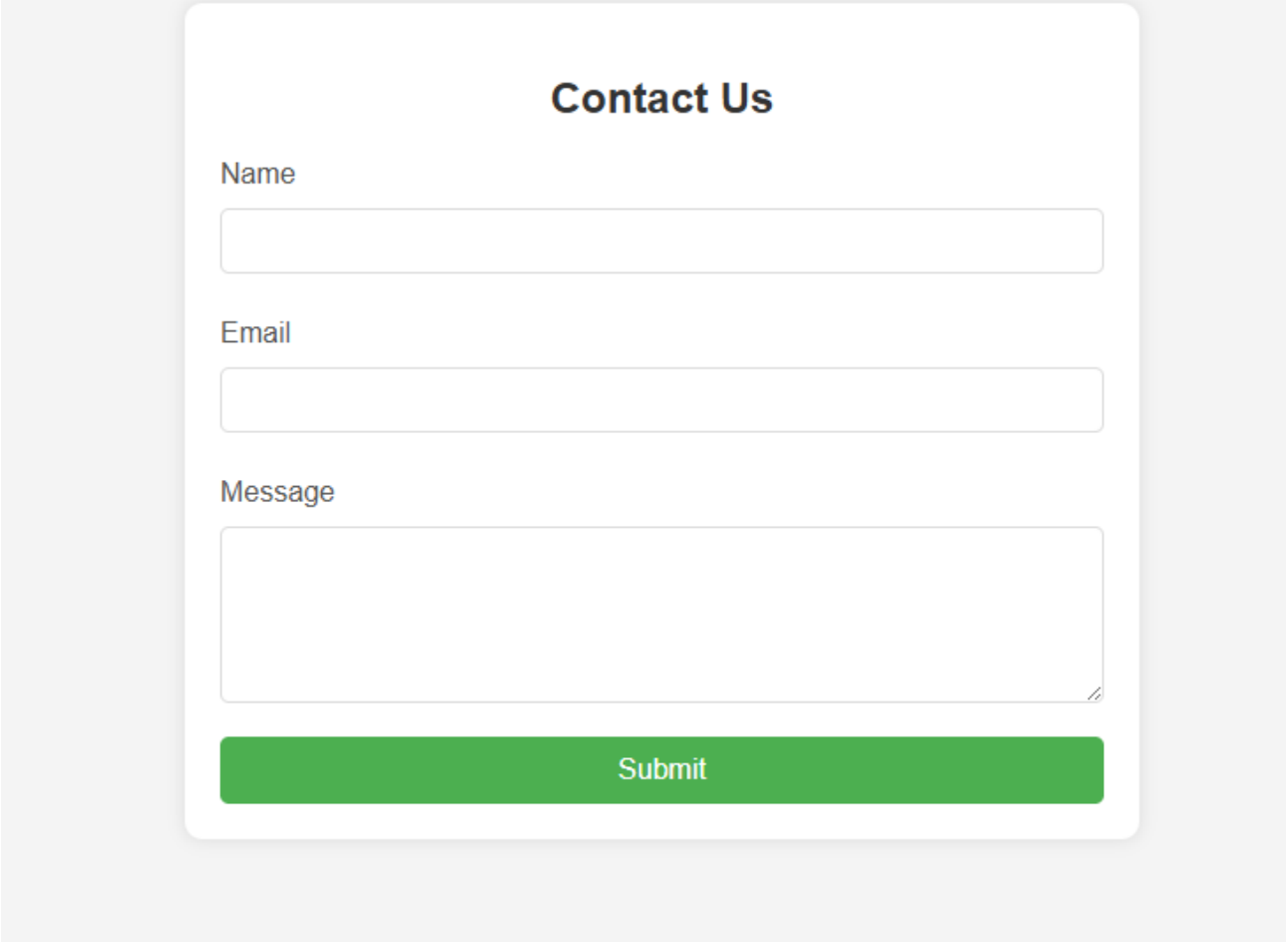
```
    self.email_field = (By.ID, "email")
    self.message_field = (By.ID, "message")
    self.submit_button = (By.CSS_SELECTOR, "button[type='submit']")
    self.name_error = (By.ID, "nameError")
    self.email_error = (By.ID, "emailError")
    self.message_error = (By.ID, "messageError")

def fill_form(self, name, email, message):

    self.send_keys(*self.name_field, name)
    self.send_keys(*self.email_field, email)
    self.send_keys(*self.message_field, message)

def submit_form(self):
    self.click(*self.submit_button)

def get_name_error(self):
    return self.get_error_message(*self.name_error)

def get_email_error(self):
    return self.get_error_message(*self.email_error)

def get_message_error(self):
    return self.get_error_message(*self.message_error)
```

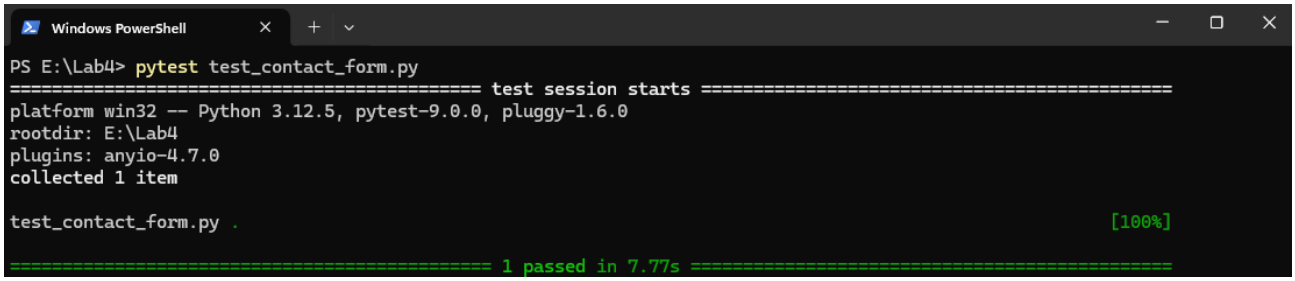
2. **Напишите позитивный автотест: заполнение всех полей формы валидными данными, отправка и проверка успешного сообщения.**

test_contact_form.py

```
import unittest
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.common.alert import Alert
from ContactPage import ContactPage

class TestContactForm(unittest.TestCase):
    def setUp(self):
        service = Service(ChromeDriverManager().install())
        self.driver = webdriver.Chrome(service=service)
        self.driver.get("file:///E:/Lab4/web.html")

    def tearDown(self):
        self.driver.quit()

    def close_alert_if_present(self):
        try:
            alert = Alert(self.driver)
            alert.accept()
        except:
            pass

    def test_valid_form_submission(self):
        contact_page = ContactPage(self.driver)
        contact_page.fill_form("Do Duc Dung", "doducdung1701@gmail.com", "This
is a valid message.")
        contact_page.submit_form()
```

```
        self.close_alert_if_present()
        self.assertEqual(contact_page.get_name_error(), "")
        self.assertEqual(contact_page.get_email_error(), "")
        self.assertEqual(contact_page.get_message_error(), "")
        print("Test valid form submission passed.")
if __name__ == "__main__":
    unittest.main()
```





3. **Напишите негативный автотест: попытка отправить форму с пустым обязательным полем и проверка текста ошибки.**

```
def test_invalid_name(self):
    """Kiểm tra trường hợp tên không hợp lệ"""
    contact_page = ContactPage(self.driver)
    contact_page.fill_form("", "john.doe@example.com", "This is a message.")
    contact_page.submit_form()
    self.close_alert_if_present()  # Đóng alert nếu có
    # Kiểm tra lỗi của trường name
    self.assertNotEqual(contact_page.get_name_error(), "")
    print("Test invalid name passed.")
```

```
PS E:\Lab4> pytest test_contact_form.py
========================================= test session starts =========================================
platform win32 -- Python 3.12.5, pytest-9.0.0, pluggy-1.6.0
rootdir: E:\Lab4
plugins: anyio-4.7.0
collected 1 item

test_contact_form.py F                                                                        [100%]

============================================== FAILURES ===============================================
_____________________________________ TestContactForm.test_invalid_name _____________________________________

self = <test_contact_form.TestContactForm testMethod=test_invalid_name>

    def test_invalid_name(self):
        contact_page = ContactPage(self.driver)
        contact_page.fill_form("", "doducdung1701@gmail.com", "This is a message.")
        contact_page.submit_form()
        self.close_alert_if_present()
>       self.assertNotEqual(contact_page.get_name_error(), "")
E       AssertionError: '' == ''

test_contact_form.py:29: AssertionError
======================================= short test summary info =======================================
FAILED test_contact_form.py::TestContactForm::test_invalid_name - AssertionError: '' == ''
========================================= 1 failed in 7.51s =========================================
```

Задание (Ручное тестирование): Составьте матрицу принятия решений для полей формы, чтобы определить все возможные валидные/невалидные комбинации.

Предположим, что у нас есть форма с тремя полями:

1. **Name** (Имя) — должно быть заполнено (не пустое).
2. **Email** (Электронная почта) — должен быть в правильном формате (например, email@example.com).
3. **Message** (Сообщение) — не должно быть пустым.

Мы определим валидные и невалидные состояния для каждого поля и составим матрицу принятия решений, которая будет включать все возможные комбинации этих состояний.

Возможные состояния полей:

- **Name**:
  - **Валидно**: Заполнено (например, "Иван Иванов").
  - **Невалидно**: Пустое поле.
- **Email**:
  - **Валидно**: Правильный формат электронной почты (например, email@example.com).
  - **Невалидно**: Неправильный формат электронной почты (например, email@com или пустое поле).
- **Message**:
  - **Валидно**: Заполнено (например, "Это сообщение").
  - **Невалидно**: Пустое поле.

Матрица принятия решений

Name	Email	Message	Ожидаемый результат
Валидно	Валидно	Валидно	Успешная отправка формы
Валидно	Валидно	Невалидно	Ошибка: пустое сообщение
Валидно	Невалидно	Валидно	Ошибка: неправильный email
Валидно	Невалидно	Невалидно	Ошибка: неправильный email и пустое сообщение
Невалидно	Валидно	Валидно	Ошибка: пустое имя
Невалидно	Валидно	Невалидно	Ошибка: пустое имя и пустое сообщение
Невалидно	Невалидно	Валидно	Ошибка: пустое имя и неправильный email
Невалидно	Невалидно	Невалидно	Ошибка: все поля неверны

Объяснение:

- Каждая строка матрицы представляет собой возможное сочетание значений полей в форме.
- **Ожидаемый результат** для каждой комбинации зависит от того, какие поля заполнены корректно:
    - Если все поля заполнены корректно, форма отправляется успешно.
    - Если хотя бы одно поле неверно, появляется ошибка.

Примечания:

1. В некоторых случаях, например, при **невалидном email** или **невалидном имени**, можно ожидать, что форма не будет отправлена, и пользователю будет показана ошибка.
2. Если **Message** пустое, то при этом должно быть показано сообщение об ошибке, даже если другие поля валидны.

Использование матрицы в тестировании:

Эту матрицу можно использовать при тестировании формы, чтобы проверить все возможные варианты ввода и убедиться, что система правильно обрабатывает все ошибки и успешные отправки формы.

Пример для тестирования:

- **Тест 1**: Вводим правильное имя, правильный email и сообщение, проверяем, что форма отправляется успешно.
- **Тест 2**: Оставляем поле **Message** пустым и проверяем, что появляется ошибка.
- **Тест 3**: Вводим неправильный формат email и проверяем, что появляется ошибка.