**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: студент группы 221-371**

**Евдокимов Артемий Русланович**

**(Фамилия И.О )**

**Дата, подпись \_\_16.09.2025\_\_ *\_\_\_\_\_\_\_\_\_\_\_***

**(Дата) (Подпись)**

**Проверил: \_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

**(Фамилия И.О., степень, звание)**

**Дата, подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *\_\_\_\_\_\_\_\_\_\_\_***

**(Дата) (Подпись)**

**Замечания: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Москва**

**2025**

**Цель работы**

Научиться структурировать код автотестов с использованием Page Object Pattern и обрабатывать различные UI-сценарии для веб-форм.

**Объект тестирования**

Веб-форма оформления заказа с следующими полями:

* Имя (обязательное поле)
* Email (обязательное поле)
* Телефон (обязательное поле)
* Адрес (обязательное поле)
* Комментарий (необязательное поле)

**Задание**

**Автоматическое тестирование:**

1. Реализовать базовый класс BasePage и класс ContactPage по паттерну Page Object
2. Написать позитивный автотест: заполнение всех полей формы валидными данными, отправка и проверка успешного сообщения
3. Написать негативный автотест: попытка отправить форму с пустым обязательным полем и проверка текста ошибки

**Ручное тестирование:**

Составить матрицу принятия решений для полей формы, чтобы определить все возможные валидные/невалидные комбинации

**Реализация**

**1. Приложение для тестирования (Flask)**

**app\_order.py**

from flask import Flask, render\_template, request, jsonify

import re

app = Flask(\_\_name\_\_)

@app.route('/order')

def order\_form():

    return render\_template('order\_form.html')

@app.route('/submit-order', methods=['POST'])

def submit\_order():

, '').strip()

    email = request.form.get('email', '').strip()

    phone = request.form.get('phone', '').strip()

    address = request.form.get('address', '').strip()

    comment = request.form.get('comment', '').strip()

# Получение данных из формы

    name = request.form.get('name', '').strip()

    email = request.form.get('email', '').strip()

    phone = request.form.get('phone', '').strip()

    address = request.form.get('address', '').strip()

    comment = request.form.get('comment', '').strip()

    # Валидация данных

    errors = {}

    if not name:

        errors['name'] = 'Поле "Имя" обязательно для заполнения'

    if not email:

        errors['email'] = 'Поле "Email" обязательно для заполнения'

    elif not re.match(r'^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$', email):

        errors['email'] = 'Введите корректный email адрес'

    if not phone:

        errors['phone'] = 'Поле "Телефон" обязательно для заполнения'

    elif not re.match(r'^\+?[1-9]\d{1,14}$', phone):

        errors['phone'] = 'Введите корректный номер телефона'

    if not address:

        errors['address'] = 'Поле "Адрес" обязательно для заполнения'

    if errors:

        return jsonify({'success': False, 'errors': errors})

    # Если все данные валидны, возвращаем успешный ответ

    return jsonify({

        'success': True,

        'message': 'Заказ успешно оформлен! Мы свяжемся с вами в ближайшее время.'

    })

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True, host='0.0.0.0', port=5000)

**2. Реализация Page Object Pattern**

**base\_page.py**

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.common.exceptions import TimeoutException

class BasePage:

    def \_\_init\_\_(self, driver):

        self.driver = driver

        self.wait = WebDriverWait(driver, 10)

    def find\_element(self, locator):

        return self.wait.until(EC.presence\_of\_element\_located(locator))

    def find\_clickable\_element(self, locator):

        return self.wait.until(EC.element\_to\_be\_clickable(locator))

    def is\_element\_visible(self, locator):

        try:

            return self.wait.until(EC.visibility\_of\_element\_located(locator))

        except TimeoutException:

            return False

    def get\_element\_text(self, locator):

        return self.find\_element(locator).text

**contact\_page.py**

from selenium.webdriver.common.by import By

from base\_page import BasePage

class ContactPage(BasePage):

    # Локаторы элементов формы

    NAME\_INPUT = (By.ID, "name")

    EMAIL\_INPUT = (By.ID, "email")

    PHONE\_INPUT = (By.ID, "phone")

    ADDRESS\_INPUT = (By.ID, "address")

    COMMENT\_INPUT = (By.ID, "comment")

    SUBMIT\_BUTTON = (By.ID, "submit-btn")

    SUCCESS\_MESSAGE = (By.CLASS\_NAME, "success-message")

    ERROR\_MESSAGE = (By.CLASS\_NAME, "error-message")

    FIELD\_ERROR = (By.CLASS\_NAME, "field-error")

    def \_\_init\_\_(self, driver):

        super().\_\_init\_\_(driver)

        self.driver = driver

    def open(self, url):

        self.driver.get(url)

    def set\_name(self, name):

        self.find\_clickable\_element(self.NAME\_INPUT).clear()

        self.find\_clickable\_element(self.NAME\_INPUT).send\_keys(name)

    def set\_email(self, email):

        self.find\_clickable\_element(self.EMAIL\_INPUT).clear()

        self.find\_clickable\_element(self.EMAIL\_INPUT).send\_keys(email)

    def set\_phone(self, phone):

        self.find\_clickable\_element(self.PHONE\_INPUT).clear()

        self.find\_clickable\_element(self.PHONE\_INPUT).send\_keys(phone)

    def set\_address(self, address):

        self.find\_clickable\_element(self.ADDRESS\_INPUT).clear()

        self.find\_clickable\_element(self.ADDRESS\_INPUT).send\_keys(address)

    def set\_comment(self, comment):

        self.find\_clickable\_element(self.COMMENT\_INPUT).clear()

        self.find\_clickable\_element(self.COMMENT\_INPUT).send\_keys(comment)

    def submit\_form(self):

        self.find\_clickable\_element(self.SUBMIT\_BUTTON).click()

    def get\_success\_message(self):

        return self.get\_element\_text(self.SUCCESS\_MESSAGE)

    def get\_error\_message(self):

        return self.get\_element\_text(self.ERROR\_MESSAGE)

    def get\_field\_error(self, field\_name):

        return self.get\_element\_text((By.ID, f"{field\_name}-error"))

    def is\_success\_message\_displayed(self):

        return self.is\_element\_visible(self.SUCCESS\_MESSAGE)

    def is\_error\_message\_displayed(self):

        return self.is\_element\_visible(self.ERROR\_MESSAGE)

**3. Автотесты**

**test\_order\_form.py**

import pytest

import os

import sys

from selenium import webdriver

from selenium.webdriver.chrome.service import Service

from selenium.webdriver.chrome.options import Options

from selenium.common.exceptions import WebDriverException

from contact\_page import ContactPage

class TestOrderForm:

    @pytest.fixture(autouse=True)

    def setup(self):

        #Пути к браузеру и драйверу

        yandex\_browser\_path = "C:\\Users\\Евдик\\AppData\\Local\\Yandex\\YandexBrowser\\Application\\browser.exe"

        yandex\_driver\_path = "C:\\Users\\Евдик\\AppData\\Local\\Yandex\\YandexBrowser\\Application\\yandexdriver.exe"

        #Проверяем существование файлов

        if not os.path.exists(yandex\_browser\_path):

            self.\_print\_result("Яндекс.Браузер не найден", "ERROR")

            pytest.skip("Яндекс.Браузер не найден")

        if not os.path.exists(yandex\_driver\_path):

            self.\_print\_result("YandexDriver не найден", "ERROR")

            pytest.skip("YandexDriver не найден")

        #Настройка опций

        options = Options()

        options.binary\_location = yandex\_browser\_path

        options.add\_argument("--disable-extensions")

        options.add\_argument("--disable-gpu")

        options.add\_argument("--no-sandbox")

        #Инициализация драйвера

        service = Service(executable\_path=yandex\_driver\_path)

        self.driver = webdriver.Chrome(service=service, options=options)

        #Установка размера окна

        try:

            self.driver.maximize\_window()

        except WebDriverException:

            self.driver.set\_window\_size(1920, 1080)

        #Инициализация Page Object

        self.contact\_page = ContactPage(self.driver)

        self.contact\_page.open("http://localhost:5000/order")

        yield

        self.driver.quit()

def \_print\_result(self, message, status="INFO"):

        print(f"\n{status}: {message}", file=sys.stderr)

        sys.stderr.flush()

    def test\_positive\_scenario\_valid\_data(self):

        """Позитивный тест: заполнение всех полей формы валидными данными"""

        #Заполнение формы

        self.contact\_page.set\_name("Иван Иванов")

        self.contact\_page.set\_email("ivan@example.com")

        self.contact\_page.set\_phone("+79161234567")

        self.contact\_page.set\_address("ул. Примерная, д. 1, кв. 1")

        self.contact\_page.set\_comment("Доставка после 18:00")

        #Отправка формы

        self.contact\_page.submit\_form()

        #Проверка успешного сообщения

        assert self.contact\_page.is\_success\_message\_displayed()

        assert "успешно" in self.contact\_page.get\_success\_message().lower()

        self.\_print\_result("Позитивный сценарий: УСПЕШНО", "SUCCESS")

    def test\_negative\_scenario\_empty\_required\_field(self):

        """Негативный тест: попытка отправить форму с пустым обязательным полем"""

        #Заполнение всех полей, кроме имени

        self.contact\_page.set\_email("ivan@example.com")

        self.contact\_page.set\_phone("+79161234567")

        self.contact\_page.set\_address("ул. Примерная, д. 1, кв. 1")

        #Отправка формы

        self.contact\_page.submit\_form()

        #Проверка сообщения об ошибке

        assert self.contact\_page.is\_error\_message\_displayed() or self.contact\_page.get\_field\_error("name") != ""

        error\_text = self.contact\_page.get\_error\_message() if self.contact\_page.is\_error\_message\_displayed() else self.contact\_page.get\_field\_error("name")

        assert "обязательно" in error\_text.lower() or "заполните" in error\_text.lower()

        self.\_print\_result("Негативный сценарий (пустое поле): ОЖИДАЕМАЯ ОШИБКА", "INFO")

    def test\_negative\_scenario\_invalid\_email(self):

        """Негативный тест: невалидный email"""

        #Заполнение формы с невалидным email

        self.contact\_page.set\_name("Евдокимов Артемий")

        self.contact\_page.set\_email("evdikkommail.ru")

        self.contact\_page.set\_phone("+79161234567")

        self.contact\_page.set\_address("ул. Первомайская, д. 1, кв. 1")

#Отправка формы

        self.contact\_page.submit\_form()

        #Проверка сообщения об ошибке

        assert self.contact\_page.is\_error\_message\_displayed() or self.contact\_page.get\_field\_error("email") != ""

        error\_text = self.contact\_page.get\_error\_message() if self.contact\_page.is\_error\_message\_displayed() else self.contact\_page.get\_field\_error("email")

        assert "email" in error\_text.lower() or "электронная почта" in error\_text.lower()

        self.\_print\_result("Негативный сценарий (невалидный email): ОЖИДАЕМАЯ ОШИБКА", "INFO")

if \_\_name\_\_ == "\_\_main\_\_":

    pytest.main([\_\_file\_\_, "-v", "-s"])

**4. Матрица принятия решений для полей формы**

| Поле | Обязательное | Валидные значения | Невалидные значения | Ожидаемый результат |
| --- | --- | --- | --- | --- |
| Имя | Да | Непустая строка (2+ символа) | Пустая строка | Ошибка: "Поле обязательно для заполнения" |
| Email | Да | Корректный email ([user@example.com](https://mailto:user@example.com/)) | Некорректный email (user, user@, user@example) | Ошибка: "Введите корректный email адрес" |
| Телефон | Да | Корректный телефон (+79161234567) | Некорректный телефон (123, abcdef) | Ошибка: "Введите корректный номер телефона" |
| Адрес | Да | Непустая строка (5+ символов) | Пустая строка | Ошибка: "Поле обязательно для заполнения" |
| Комментарий | Нет | Любая строка (включая пустую) | Нет невалидных значений | Всегда принимается |

**Результаты тестирования**

**Успешно выполненные тесты:**

1. **Позитивный сценарий** - заполнение всех полей формы валидными данными
   * Заполнение всех полей корректными данными
   * Отправка формы
   * Проверка успешного сообщения
   * Результат: "Позитивный сценарий: УСПЕШНО"
2. **Негативный сценарий (пустое поле)** - попытка отправки формы с пустым обязательным полем
   * Заполнение всех полей, кроме одного обязательного
   * Отправка формы
   * Проверка сообщения об ошибке
   * Результат: "Негативный сценарий (пустое поле): ОЖИДАЕМАЯ ОШИБКА"
3. **Негативный сценарий (невалидный email)** - попытка отправки формы с невалидным email
   * Заполнение формы с некорректным email
   * Отправка формы
   * Проверка сообщения об ошибке
   * Результат: "Негативный сценарий (невалидный email): ОЖИДАЕМАЯ ОШИБКА"

**Статистика тестирования:**

* Всего тестов: 3
* Успешно пройдено: 3
* Провалено: 0
* Пропущено: 0

**Выводы**

В ходе лабораторной работы были успешно выполнены все поставленные задачи:

1. Реализована структура Page Object Pattern с базовым классом BasePage и классом ContactPage для работы с формой заказа
2. Написан позитивный автотест для проверки успешного заполнения и отправки формы
3. Написаны негативные автотесты для проверки обработки ошибок при невалидных данных
4. Составлена матрица принятия решений для всех полей формы
5. Разработано Flask-приложение с формой заказа и валидацией данных

Основные преимущества реализованного подхода:

* Четкое разделение кода на логические компоненты (Page Objects)
* Упрощение поддержки и модификации тестов
* Возможность повторного использования кода
* Улучшенная читаемость и структурированность тестов

Разработанные автотесты охватывают основные сценарии работы с веб-формой и могут быть расширены для проверки дополнительных кейсов. Применение Page Object Pattern значительно повысило поддерживаемость тестов по сравнению с предыдущей лабораторной работой.