

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра ИС

ОТЧЕТ
по практической работе № 6
по дисциплине «Автоматизация тестирования»
Тема: «Тестирование на основе паттерна Page Object»

Студент гр. 2372

Преподаватель

Соколовский В. Д.

Турнецкая Е.Л.

Санкт-Петербург

2024

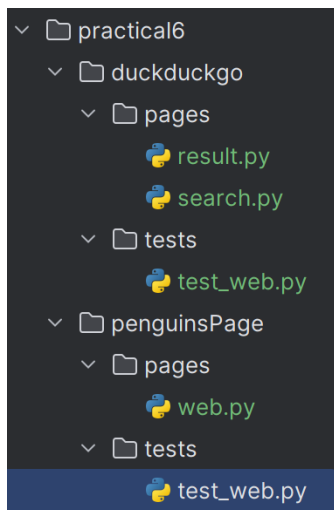
Цель работы: получение практических навыков по тестированию веб-элементов на основе паттерна Page Object.

Поставленные задачи:

1. Изучить особенности тестирования с использованием паттерна Page Object.
2. Реализовать проект по автоматизированному тестированию поисковой веб-системы по релевантности выполнения уникального запроса.
3. Реализовать индивидуальный проект по автоматизированному тестированию на учебном ресурсе с использованием модели Page Object.

Выполнение работы:

Структура проекта согласно рекомендациям разработчиками Pytest:



Скриншот 1 – Структура проекта в PyCharm

Проект по тестированию поисковой системы DuckDuckGo на основе паттерна Page Object по самостоятельно определенному запросу.

Листинг 1 – Код программы в search.py

```
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys

class DuckDuckGoSearchPage:
    URL = 'https://www.duckduckgo.com'
    SEARCH_INPUT = (By.ID, "searchbox_input")

    def __init__(self, browser):
        self.browser = browser

    def load(self):
        self.browser.get(self.URL)

    def search(self, phrase):
        search_input = self.browser.find_element(*self.SEARCH_INPUT)
        search_input.send_keys(phrase + Keys.RETURN)
```

Листинг 2 – код программы в result.py

```
from selenium.webdriver.common.by import By

class DuckDuckGoResultPage:
    SEARCH_RESULTS = (By.CSS_SELECTOR, "li[data-layout='organic']")
    SEARCH_INPUT = (By.ID, 'search_form_input')
    @classmethod
    def PHRASE_RESULTS(cls, phrase):
        xpath = f"//li[@data-layout='organic']//a[contains(@href, '{phrase}']]"
        return (By.XPATH, xpath)

    def __init__(self, browser):
        self.browser = browser

    def search_results_count(self):
        search_results = self.browser.find_elements(*self.SEARCH_RESULTS)
        return len(search_results)

    def phrase_result_count(self, phrase):
        phrase_results = self.browser.find_elements(*self.PHRASE_RESULTS(phrase))
        return len(phrase_results)
    def search_input_value(self):
        search_input = self.browser.find_element(*self.SEARCH_INPUT)
        return search_input.get_attribute('value')
```

Листинг 3 – код автотеста test_web.py

```
import pytest
from TestAutomation.practical6.duckduckgo.pages.result import DuckDuckGoResultPage
from TestAutomation.practical6.duckduckgo.pages.search import DuckDuckGoSearchPage
from selenium.webdriver import Chrome

@pytest.fixture
def browser():
    driver = Chrome()
    driver.implicitly_wait(10)
    yield driver
    driver.quit()

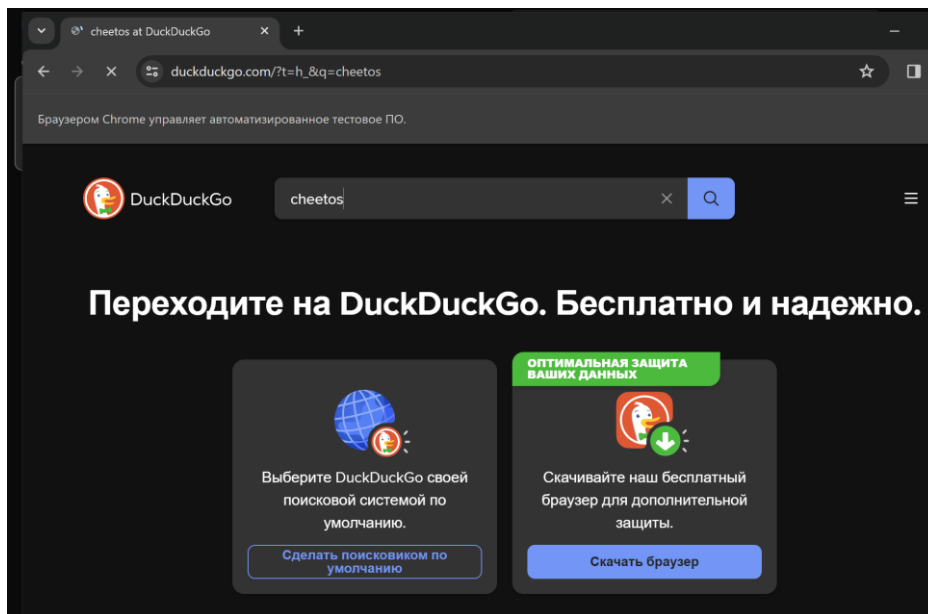
def test_basic_duckduckgo_search(browser):
    # Настройте данные для тест-кейса
    PHRASE = 'cheetos'
```

```
# Поиск фразы
search_page = DuckDuckGoSearchPage(browser)
search_page.load()
search_page.search(PHRASE)

# Проверка, что результаты появились
result_page = DuckDuckGoResultPage(browser)

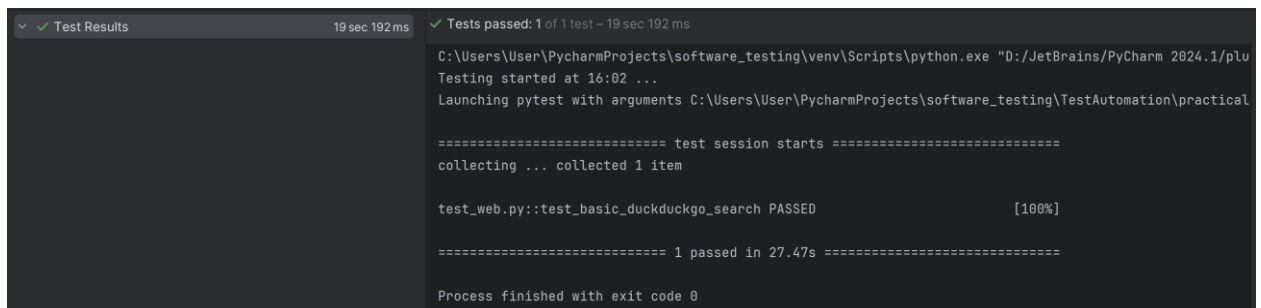
assert result_page.search_results_count() > 0
assert result_page.phrase_result_count(PHRASE) > 0
assert result_page.search_input_value() == PHRASE
```

Запуск кода:



Скриншот 2 – Запуск автотеста test_web.py

Завершение:



Скриншот 3 - Успешное завершение тестирования

Проект по тестированию наличия шести предметных карточек на веб-странице на основе паттерна Page Object.

Листинг 1 – код программы web.py

```
from selenium.webdriver.common.by import By

class PenguinsPage:
    URL = 'https://qa-test-selectors.netlify.app'
    VARIANT = 17
    HEADING = "Крутая челочка"
    TITLE_TEXT = "Златовласка"

    def __init__(self, browser):
        self.browser = browser

    def load(self):
        self.browser.get(self.URL)

    def find_variant(self):
        button = self.browser.find_element(By.CSS_SELECTOR, f'.variant__btn:nth-child({self.VARIANT})')

        # Нажатие на кнопку с вариантом
        button.click()

    def penguins_elements_count(self):
        # Поиск элементов с data-type="penguins"
        penguins_elements = self.browser.find_elements(By.XPATH, '//*[@data-type="penguins"]')
        return len(penguins_elements)

    def bang_elements_count(self):
        # Поиск элементов с id="bang"
        bang_elements = self.browser.find_elements(By.ID, 'bang')
        return len(bang_elements)

    def coolBang_elements_count(self):
        # Поиск элементов с class="coolBang"
        coolBang_elements = self.browser.find_elements(By.CLASS_NAME, 'coolBang')
        return len(coolBang_elements)

    def goldy_elements_count(self):
        # Поиск элементов с name="goldy-hair"
        goldy_elements = self.browser.find_elements(By.NAME, 'goldy-hair')
        return len(goldy_elements)

    def heading_images_count(self):
        # Поиск изображений с heading="Крутая челочка"
```

```
heading_images = self.browser.find_elements(By.XPATH, f'//img[@heading="{self.HEADING}"]')
return len(heading_images)

def title_elements_count(self):
    # Поиск элементов с name="goldy-hair"
    title_elements = self.browser.find_elements(By.XPATH, f'//h1[text()="{self.TITLE_TEXT}"]')
    return len(title_elements)
```

Листинг 2 – код автотеста test_web.py

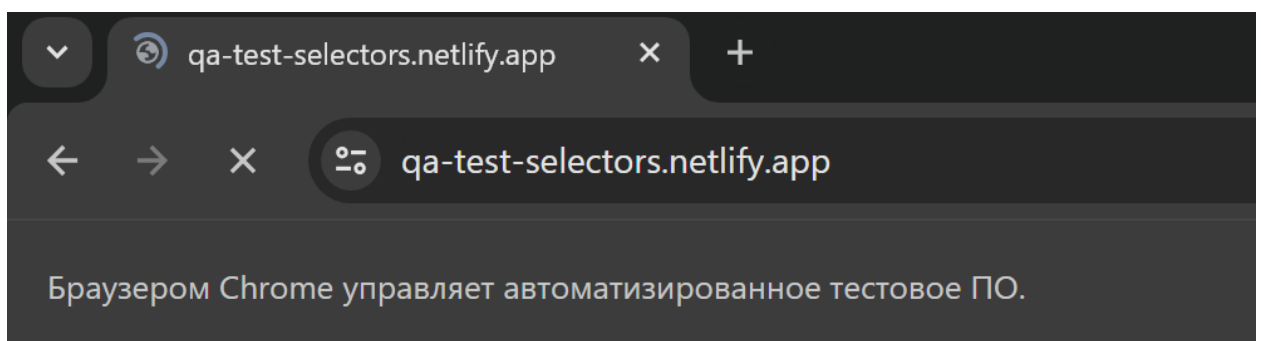
```
import pytest
from TestAutomation.practical6.penguinsPage.pages.web import PenguinsPage
from selenium.webdriver import Chrome

# Реализация фикстуры
@pytest.fixture
def browser():
    driver = Chrome()
    driver.implicitly_wait(30)
    yield driver
    driver.quit()

#Функция по проверке осуществления перехода на страницу с вариантом
def test_penguin_page(browser):
    penguin_page = PenguinsPage(browser)
    penguin_page.load()
    penguin_page.find_variant()

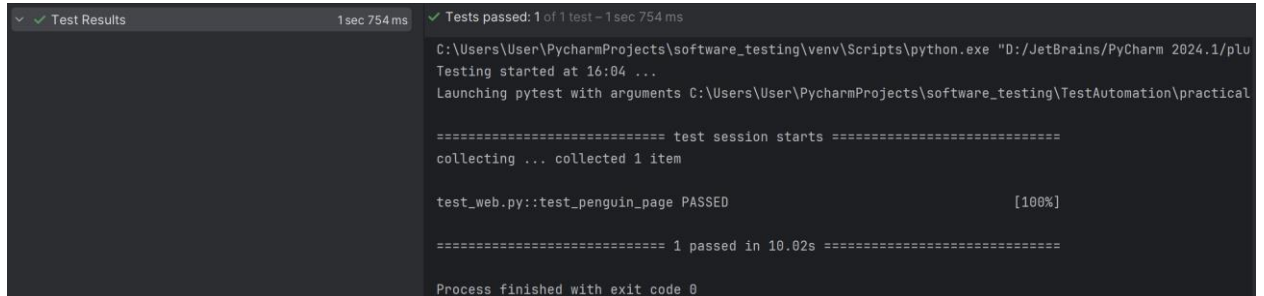
    # Реализация проверок с помощью PyTest
    assert penguin_page.penguins_elements_count() > 0
    assert penguin_page.bang_elements_count() > 0
    assert penguin_page.coolBang_elements_count() > 0
    assert penguin_page.goldy_elements_count() > 0
    assert penguin_page.heading_images_count() > 0
    assert penguin_page.title_elements_count() > 0
```

Запуск:



Скриншот 4 – Запуск кода автотеста test_web.py

Завершение:



The screenshot shows the 'Test Results' window in PyCharm. The left pane shows a green checkmark and 'Test Results' with a duration of '1 sec 754 ms'. The right pane shows the test output for 'test_web.py::test_penguin_page', which passed. The output text is as follows:

```
✓ Tests passed: 1 of 1 test - 1 sec 754 ms
C:\Users\User\PycharmProjects\software_testing\venv\Scripts\python.exe "D:/JetBrains/PyCharm 2024.1/plu
Testing started at 16:04 ...
Launching pytest with arguments C:\Users\User\PycharmProjects\software_testing\TestAutomation\practical

===== test session starts =====
collecting ... collected 1 item

test_web.py::test_penguin_page PASSED [100%]

===== 1 passed in 10.02s =====

Process finished with exit code 0
```

Скриншот 5 - Успешное завершение тестирования

Выводы:

В ходе выполнения данной работы имело своей целью получение практических навыков по тестированию веб-элементов с использованием паттерна Page Object. Этот паттерн позволяет структурировать тесты таким образом, что они становятся более читаемыми и поддерживаемыми, поскольку логика взаимодействия с веб-страницами отделяется от самих тестов. Для достижения этой цели было поставлено несколько задач, которые позволили глубже разобраться в теории и практике автоматизированного тестирования.

Изучение особенностей тестирования с использованием паттерна Page Object дало понимание, как правильно организовывать код тестов и взаимодействовать с веб-элементами.

Реализация проекта по автоматизированному тестированию поисковой веб-системы по релевантности выполнения уникального запроса позволила применить полученные знания на практике. Мы создали тест, который проверяет, насколько эффективно система справляется с запросами, и убедились, что наша тестовая структура поддерживает легкую модификацию и расширение тестов.

Индивидуальный проект по автоматизированному тестированию на учебном ресурсе с использованием модели Page Object позволил закрепить знания и навыки, полученные на предыдущих этапах. В этом проекте мы смогли проанализировать и реализовать более сложные сценарии тестирования, что позволило нам глубже понять преимущества использования Page Object в реальных условиях.

В результате выполнения работы были приобретены важные навыки по автоматизированному тестированию с использованием паттерна Page Object. Мы научились не только эффективно писать тесты, но и организовывать их таким образом, чтобы они были легки в поддержке и расширении. Эти знания и умения, несомненно, пригодятся в дальнейшей профессиональной деятельности.

Список использованных источников:

1. Документация по Page Object. URL:
https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/
2. Реализация паттерна Page Object на Python + Pytest.
URL: <https://habr.com/ru/articles/472156/>