

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

КУРСОВАЯ РАБОТА
по дисциплине «Автоматизация тестирования»
Тема: Автоматизация тестирования web-приложений

Студент гр. 2372

Соколовский В. Д.

Преподаватель

Турнецкая Е. Л.

Санкт-Петербург

2024

СОДЕРЖАНИЕ

	Введение	3
1.	Раздел 1	4
1.1.	Разработка проекта по автоматизации управления браузером с помощью Selenium Webdriver.	4
2.	Раздел 2	6
2.1.	Реализация автоматизированного тестирования релевантности результатов запроса в поисковой системе с помощью фреймворка PyTest.	6
3.	Раздел 3	9
3.1.	Автоматизированное тестирование веб-страницы на основе паттерна Page Object.	9
	Заключение	15
	Список использованных источников	17

ВВЕДЕНИЕ

Автоматизированное тестирование программного обеспечения используется для проверки работоспособности приложений и поиска ошибок в их работе. Оно помогает ускорить процесс разработки ПО и повысить его качество. Автоматизация тестов позволяет быстро обнаруживать ошибки и сбои в программе, что уменьшает затраты на исправление этих ошибок на поздних этапах разработки. Кроме того, автоматизированное тестирование повышает надежность продукта, так как автоматические тесты выполняются более точно и последовательно, чем ручные проверки.

Цель: получение практических навыков по автоматизации тестирования и контроля качества веб-приложений.

Основные задачи проекта включают в себя:

1. Изучить особенности тестирования веб-приложений с использованием Selenium Webdriver, языка программирования Python и фреймворка Pytest.
2. Реализовать проект по автоматизированному тестированию поисковой веб-системы по релевантности выполнения уникального запроса с помощью паттерна Page Object.
3. Реализовать проект по автоматизированному тестированию web-приложения с помощью Selenium Webdriver, языка программирования Python и фреймворка Pytest.

1. РАЗДЕЛ 1

1.1. Разработка проекта по автоматизации управления браузером с помощью Selenium Webdriver.

Листинг 1 – Код программы в YouTubeOpenClose.py

```
from selenium import webdriver
import time
from selenium.webdriver.chrome.service import Service as
ChromeService

# Путь к исполняемому файлу chromedriver.exe
chrome_driver_path = 'TestAutomation/coursework/chromedriver-
win64/chromedriver.exe'

# Создание сервиса Chrome
chrome_service = ChromeService(executable_path=chrome_driver_path)

# Создание экземпляра браузера
driver = webdriver.Chrome(service=chrome_service)

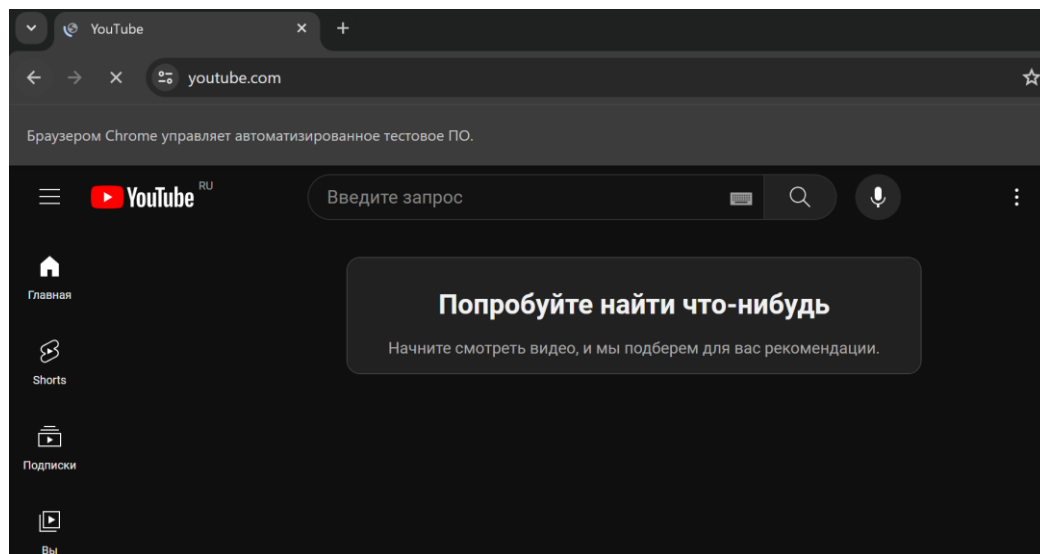
try:
    # Открытие веб-сайта
    driver.get('https://www.youtube.com/')

    # Ждем некоторое время (в данном случае, 5 секунд)
    time.sleep(5)

except Exception as e:
    print(f"Произошла ошибка: {e}")

finally:
    # Закрытие браузера
    driver.quit()
```

Запуск кода:



Скриншот 1 – Запуск программного кода YouTubeOpenClose.py

2. РАЗДЕЛ 2

1.2. Реализация автоматизированного тестирования релевантности результатов запроса в поисковой системе с помощью фреймворка PyTest.

Листинг 1 – Программный код в test_YouTube.py

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.service import Service as
ChromeService
import time

# Путь к исполняемому файлу chromedriver.exe
chrome_driver_path = 'TestAutomation/coursework/chromedriver-
win64/chromedriver.exe'

# Создание сервиса Chrome
chrome_service = ChromeService(executable_path=chrome_driver_path)

# Создание экземпляра браузера
driver = webdriver.Chrome(service=chrome_service)

try:
    # Открытие веб-сайта YouTube
    driver.get('https://www.youtube.com/')

    # Нахождение элемента поля поиска
    search_box = driver.find_element(By.CSS_SELECTOR,
'input#search')

    # Ввод запроса в поле поиска
    search_query = 'Python programming tutorial'
    search_box.send_keys(search_query)

    # Отправка формы поиска
    search_box.send_keys(Keys.RETURN)

    # Ожидание загрузки результатов поиска
    time.sleep(8) # Подождем несколько секунд для загрузки
результатов
```

```

# Проверка релевантности результатов поиска
results = driver.find_elements(By.CSS_SELECTOR, 'ytd-video-renderer') # Находим все элементы результатов поиска

# Вывод количества найденных результатов
print(f'Найдено результатов: {len(results)}')

# Проверка первых нескольких результатов на релевантность
for i, result in enumerate(results[:5]):
    title = result.find_element(By.CSS_SELECTOR, 'h3.title-and-badge.style-scope.ytd-video-renderer').text
    print(f'Результат {i + 1}: {title}')
    assert 'Python' in title or 'programming' in title.lower(), f'Нерелевантный результат: {title}'

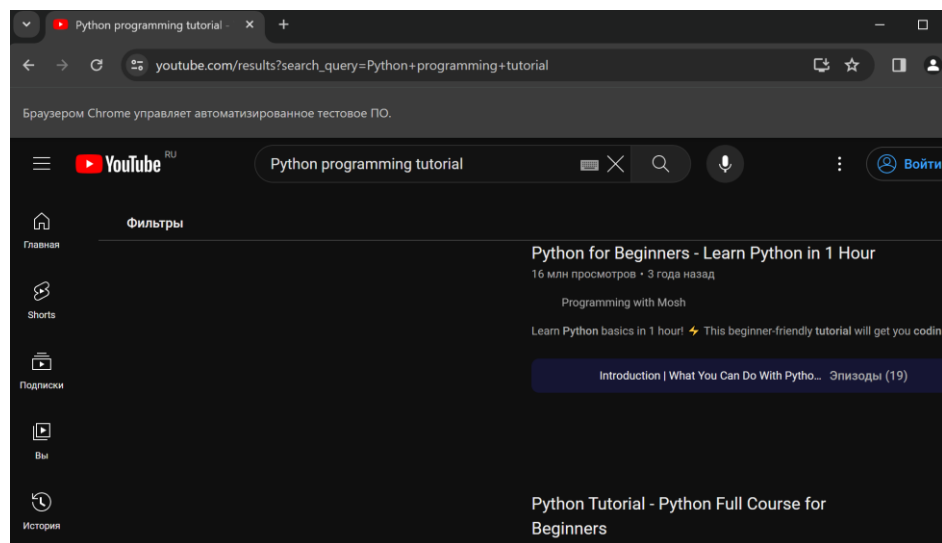
finally:
    # Закрытие браузера
    driver.quit()

```

Цель автотеста:

Проверить, соответствуют ли первые несколько запросов поиска на YouTube введенному поисковому запросу, содержащему ключевые слова «Python programming tutorial»

Запуск кода:



Скриншот 1 – Запуск автотеста test_YouTube.py

Результат и завершение работы:

```
Найдено результатов: 13
Результат 1: Python for Beginners - Learn Python in 1 Hour
Результат 2: Python Tutorial - Python Full Course for Beginners
Результат 3: Learn Python - Full Course for Beginners [Tutorial]
Результат 4: Python Full Course for free 🐍
Результат 5: Python — полный курс для начинающих. Этот навык изменит твою жизнь.

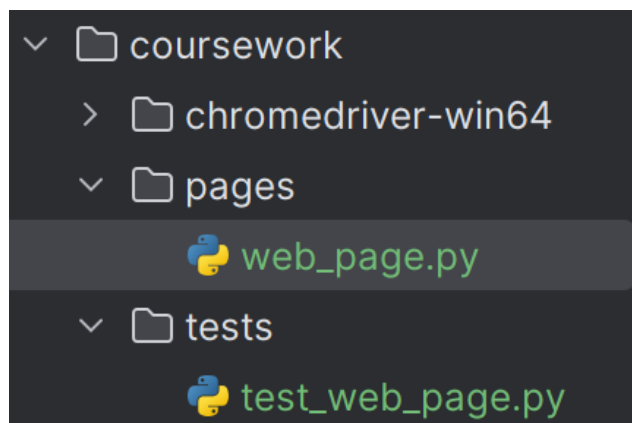
Process finished with exit code 0
```

Скриншот 2 – Результат кода

3. РАЗДЕЛ 3

3.1. Автоматизированное тестирование веб-страницы на основе паттерна Page Object. Тестирование самостоятельно выбранного веб-приложения.

Структура проекта:



Скриншот 1 - Структура проекта в PyCharm

Листинг 1 – Программный код файла web_page.py

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

class YouTubePage:
    def __init__(self, driver):
        self.driver = driver
        self.url = "https://www.youtube.com" # URL страницы YouTube
        self.search_box_selector = 'input#search' # CSS-селектор
        для поля поиска
        self.video_container_selector = 'ytd-video-renderer' # CSS-
        селектор для контейнера видео
        self.video_title_selector = 'h3.title-and-badge.style-
        scope.ytd-video-renderer' # для заголовка видео

    def load(self):
        self.driver.get(self.url)

    def search(self, phrase):
        search_box = self.driver.find_element(By.CSS_SELECTOR,
```

```

self.search_box_selector)
    search_box.send_keys(phrase) # Вводим фразу в поле поиска
    search_box.send_keys(Keys.RETURN)

def get_count_of_video_containers_with_phrase(self, phrase):
    time.sleep(5)
    # Находим все контейнеры с видео
    video_containers =
self.driver.find_elements(By.CSS_SELECTOR,
self.video_container_selector)
    count = 0
    for container in video_containers:
        # Получаем текст заголовка видео
        title = container.find_element(By.CSS_SELECTOR,
self.video_title_selector).text
        if phrase.lower() in title.lower(): # Проверяем,
содержится ли искомая фраза в заголовке
            count += 1
    return count # Возвращаем количество видео, содержащих
искомую фразу в заголовке

def get_count_of_video_images(self):
    time.sleep(3)
    video_images = self.driver.find_elements(By.CSS_SELECTOR,
f'{self.video_container_selector} img')
    return len(video_images) # Возвращаем количество найденных
изображений

def get_count_of_play_buttons(self):
    time.sleep(3)
    play_buttons = self.driver.find_elements(
        By.CSS_SELECTOR,
        f'{self.video_container_selector} .yt-simple-
endpoint.style-scope.ytd-thumbnail'
    ) # Находим все кнопки воспроизведения
    return len(play_buttons) # Возвращаем количество найденных
кнопок

def open_first_video(self, phrase):
    time.sleep(3)
    video_containers =
self.driver.find_elements(By.CSS_SELECTOR,

```

```

self.video_container_selector)
    for container in video_containers:
        title = container.find_element(By.CSS_SELECTOR,
self.video_title_selector).text
        if phrase.lower() in title.lower(): # Проверяем,
содержится ли искомая фраза в заголовке
            container.click() # Кликаем на первое найденное
видео
        return 1
    return 0

```

Листинг 2 - Программный код файла test_web_page.py

```

import pytest
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.common.exceptions import WebDriverException
import os

from TestAutomation.coursework.pages.web_page import YouTubePage

chrome_driver_path = os.path.abspath('../chromedriver-
win64/chromedriver.exe')

if not os.path.exists(chrome_driver_path):
    raise FileNotFoundError(f"Chromedriver not found at path:
{chrome_driver_path}")

# Создание объекта сервиса с указанием пути до chromedriver
service = Service(executable_path=chrome_driver_path)

@pytest.fixture
def browser():
    # Инициализация драйвера Chrome с использованием созданного
объекта сервиса
    driver = webdriver.Chrome(service=service)
    driver.implicitly_wait(10)
    yield driver
    driver.quit()

```

```
def test_page(browser):
    PHRASE = 'Тестировщик' # Фраза для тестов
    page = YouTubePage(browser)
    page.load() # Загрузка страницы
    page.search(PHRASE) # Поиск фразы
    # Проверки
    assert page.get_count_of_video_containers_with_phrase(PHRASE) > 0

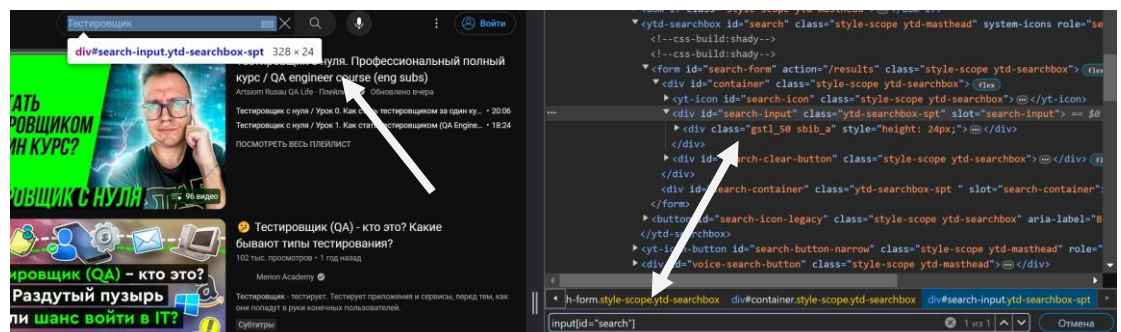
    assert page.get_count_of_video_images() > 0
    assert page.get_count_of_play_buttons() > 0
    assert page.open_first_video(PHRASE) == 1
```

Цель автотеста:

- Проверка корректности результатов поиска видео по заданной фразе.
- Подсчет количества видео, заголовки которых содержат определенную фразу.
- Проверка наличия изображений и кнопок воспроизведения на странице результатов поиска.
- Открытие первого найденного видео для дальнейшего анализа его содержания или поведения.

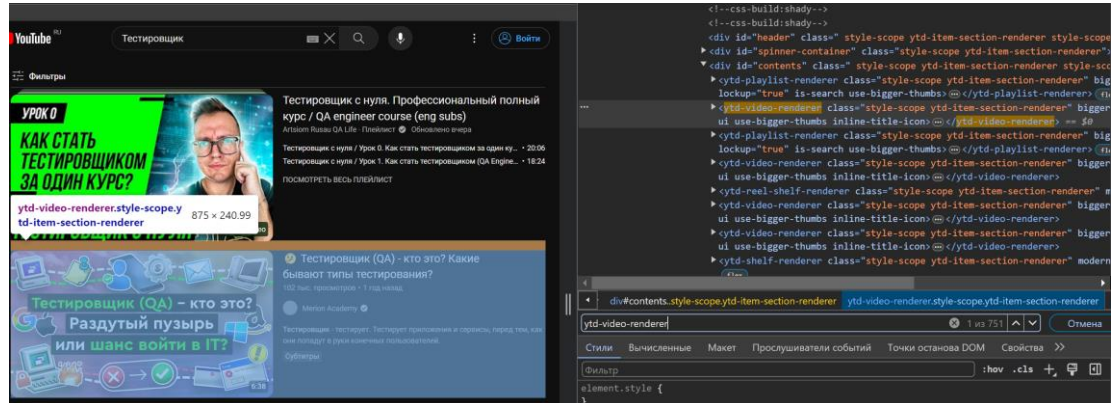
Нахождение и проверка CSS-селекторов через DevTools:

CSS-селектор поля поиска: `input[id="search"]`

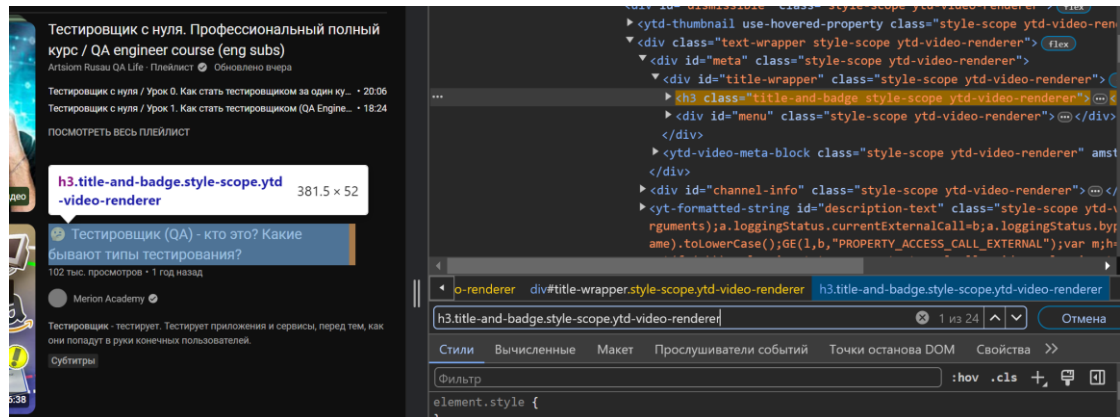


Скриншот 2 - Определение локаторов для строки поиска

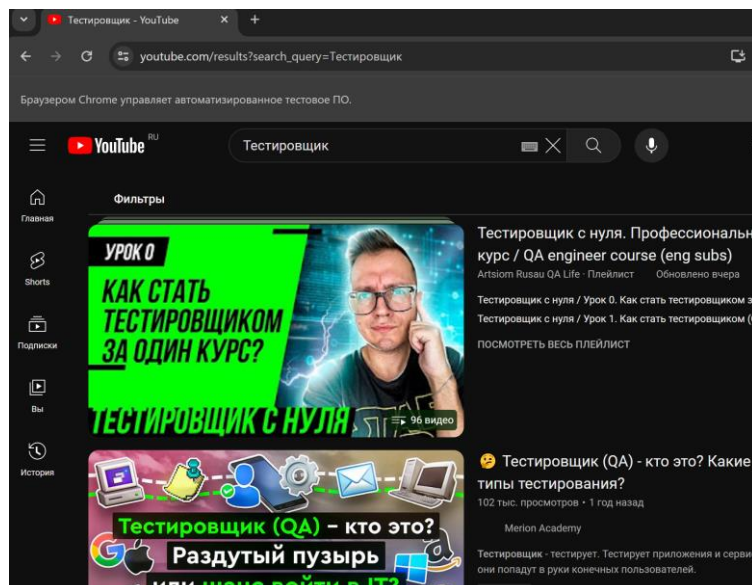
CSS-селектор для контейнера видео: *ytd-video-renderer*



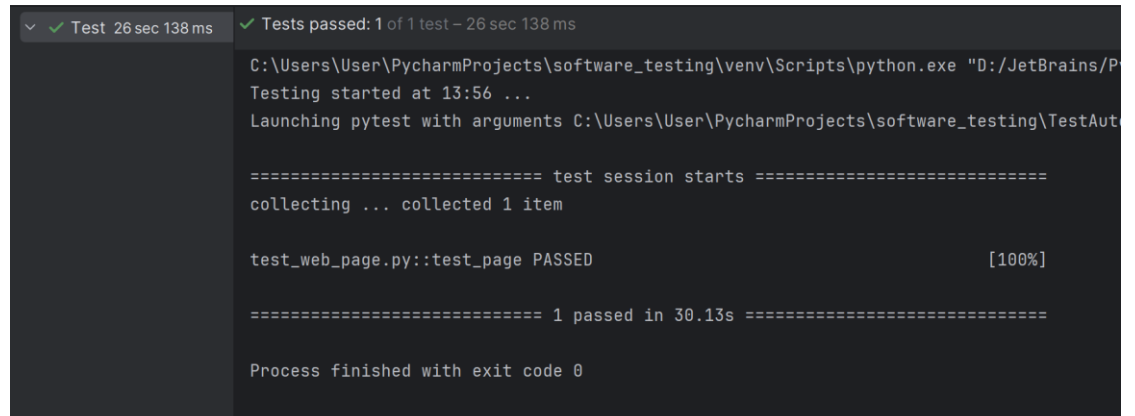
CSS-селектор для заголовка видео: *h3.title-and-badge.style-scope.ytd-video-renderer*



Запуск автотеста:



Результат успешно проведенного:



```
✓ Test 26 sec 138 ms  ✓ Tests passed: 1 of 1 test - 26 sec 138 ms

C:\Users\User\PycharmProjects\software_testing\venv\Scripts\python.exe "D:/JetBrains/P
Testing started at 13:56 ...
Launching pytest with arguments C:\Users\User\PycharmProjects\software_testing\TestAut

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

test_web_page.py::test_page PASSED [100%]

===== 1 passed in 30.13s =====

Process finished with exit code 0
```

Скриншот 6 - Результат теста

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы были разработаны и реализованы три основных раздела, связанных с автоматизацией тестирования веб-приложений с использованием инструментов Selenium WebDriver, Python и Pytest.

В первом разделе, "Разработка проекта по автоматизации управления браузером с помощью Selenium WebDriver", позволил еще раз поработать с основными принципами работы с Selenium WebDriver, его настройкой и использованием для управления браузером. В рамках данного раздела был разработан и протестирован программный код автотеста, демонстрирующий открытие веб-приложения в браузере с использованием Selenium WebDriver.

Во втором разделе, "Реализация автоматизированного тестирования релевантности результатов запроса в поисковой системе с помощью фреймворка PyTest", были выполнены тесты на релевантность результатов поисковой выдачи на основе фреймворка PyTest. Был разработан программный код автотеста с комментариями, а также представлены скриншоты, подтверждающие успешное проведение тестовых мероприятий.

В третьем разделе, "Автоматизированное тестирование веб-страницы на основе паттерна Page Object", были использованы DevTools для проведения тестовых мероприятий, а также разработан программный код автотестов с комментариями на основе паттерна Page Object.

В результате выполнения всех разделов данной работы были лучше усвоены и отработаны практические навыки по автоматизации тестирования веб-приложений с использованием различных инструментов и подходов. Эти навыки будут ценным активом для дальнейшей карьеры в области разработки и тестирования программного обеспечения, а также позволят повысить эффективность и качество разрабатываемых продуктов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация по Selenium Webdriver. URL: <https://www.selenium.dev/documentation/webdriver/>
2. Материалы курсы «Автоматизация тестирования с помощью selenium и python». URL: <https://stepik.org/course/575/>
3. Документация по PyTest. URL: <https://www.lambdatest.com/learninghub/pytest-tutorial>
4. Реализация паттерна Page Object на Python + Pytest. URL: <https://habr.com/ru/articles/472156/>