Отчёт по лабораторной работе №4
курс «Тестирование программного обеспечения Aquarius»

Выполнил студент группы ИП-211:
Вайберт Андреас Андреевич
Преподаватель курса:
Бочкарёв Борис Вячеславович

Новосибирск 2025

Цель работы

Освоить навыки разработки автотестов для веб-интерфейса с использованием Selenium на примере функционала авторизации в OpenBMC.

Задачи

● Изучить основы работы с Selenium.

● Разработать автотесты для проверки функционала авторизации в Web UI OpenBMC.

● Покрыть основные сценарии авторизации (успешная авторизация, неверные

данные, блокировка учетной записи).

● Запустить автотесты и проанализировать результаты.

Ход работы:

1. Подготовка окружения
    a. Установлен Python 3.12 и создано виртуальное окружение venv.
    b. С помощью pip установлены необходимые библиотеки: selenium, selenium-wire.
    c. Загружен и настроен ChromeDriver
    d. Выполнена команда
    “busctl set-property xyz.openbmc_project.User.Manager /xyz/openbmc_project/user xyz.openbmc_project.User.AccountPolicy MaxLoginAttemptBeforeLockout q 3”
    внутри запущенного OpenBMC в QEMU для установки блокировки учетной записи после 3 неудачных попыток

Реализация тестовых сценариев:

1. test_successful_login
    a. Проверяет возможность успешного входа с верными логином и паролем (root/0penBmc). После входа ожидается появление элемента с css классом “main-content” на странице.
2. test_invalid_credentials:
    a. Проверяет реакцию системы на ввод неверного пароля. Изначально возникла проблема с обнаружением сообщения об ошибке в UI, так как оно появлялось на очень короткое время. Для обхода этой проблемы был использован selenium-wire для перехвата сетевого запроса, отправляемого при попытке входа. Тест проверяет, что сервер возвращает HTTP статус-код 401 Unauthorized в ответ на попытку входа с неверными данными. Путь API для входа был определен как /redfish/v1/SessionService/Sessions.
3. test_account_lockout:
    a. Проверяет механизм блокировки учетной записи. Выполнена команда
    “busctl set-property xyz.openbmc_project.User.Manager /xyz/openbmc_project/user xyz.openbmc_project.User.AccountPolicy MaxLoginAttemptBeforeLockout q 3”
    внутри запущенного OpenBMC в QEMU для установки блокировки учетной записи после 3 неудачных попыток. Тест выполняет 3 неудачные попытки входа, проверяя после каждой получение статуса 401. Затем выполняется попытка входа с правильными учетными

данными. Ожидалось, что эта попытка также завершится ошибкой (например, статус 401 или 403) из-за блокировки.

Анализ результатов и выводы по тесту блокировки:

Тесты test_successful_login и test_invalid_credentials успешно пройдены. test_invalid_credentials корректно определяет неудачный вход по HTTP статус-коду 401, полученному через selenium-wire.

Тест test_account_lockout завершился с ошибкой AssertionError: 201 != 401. Это означает, что после трех последовательных неудачных попыток входа (каждая из которых вернула ожидаемый статус 401), последующая попытка входа с правильными учетными данными завершилась успехом (сервер вернул статус 201 Created). Ожидаемым поведением при работающем механизме блокировки был бы отказ в доступе (например, статус 401 или 403).

Несмотря на выполнение команды
“busctl set-property xyz.openbmc_project.User.Manager /xyz/openbmc_project/user xyz.openbmc_project.User.AccountPolicy MaxLoginAttemptBeforeLockout q 3”
механизм блокировки учетной записи не сработал должным образом протестированном окружении.

Вывод

В ходе лабораторной работы я освоил навыки разработки автотестов для веб-интерфейса с использованием Selenium. Был созданы тесты для проверки функционала авторизации в OpenBMC, которые покрыли основные сценарии и были проанализированы результаты. Эти навыки могут быть применены для тестирования других веб-приложений.

openbmc_auth_tests.py:

```
import unittest

import time
from seleniumwire import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException, NoSuchElementException,
StaleElementReferenceException

OPENBMC_URL = 'https://localhost:2443'
VALID_USERNAME = 'root'
VALID_PASSWORD = '0penBmc'
INVALID_PASSWORD = 'wrongpassword'
LOCKOUT_ATTEMPTS = 3
WAIT_TIMEOUT = 15

LOGIN_API_PATH_PART = '/redfish/v1/SessionService/Sessions'
```

```
EXPECTED_INVALID_LOGIN_STATUS = 401
EXPECTED_LOCKED_OUT_STATUS = 401


class OpenBMCAuthTests(unittest.TestCase):

    def setUp(self):
        chrome_options = Options()
        chrome_options.add_argument('--ignore-certificate-errors')
        # chrome_options.add_argument('--headless')
        chrome_options.add_argument('--no-sandbox')
        chrome_options.add_argument('--disable-dev-shm-usage')
        chrome_options.add_argument("--start-maximized")

        sw_options = {'verify_ssl': False}
        self.driver = webdriver.Chrome(options=chrome_options,
seleniumwire_options=sw_options)

        self.driver.get(OPENBMC_URL)
        try:
            WebDriverWait(self.driver, WAIT_TIMEOUT).until(
                EC.presence_of_element_located((By.ID, 'username'))
            )
        except TimeoutException:
            self.driver.quit()
            self.fail(f"Failed to load login page (initial load)")
        del self.driver.requests
        self.driver.delete_all_cookies()
        self.driver.refresh()
        try:
            WebDriverWait(self.driver, WAIT_TIMEOUT).until(
                EC.presence_of_element_located((By.ID, 'username'))
            )
        except TimeoutException:
            self.driver.quit()
            self.fail(f"Failed to reload login page")
        del self.driver.requests


    def _perform_login(self, username, password):
        try:
            user_field = WebDriverWait(self.driver,
WAIT_TIMEOUT).until(EC.element_to_be_clickable((By.ID, 'username')))
            user_field.clear()
            user_field.send_keys(username)

            pass_field = WebDriverWait(self.driver,
WAIT_TIMEOUT).until(EC.element_to_be_clickable((By.ID, 'password')))
            pass_field.clear()
            pass_field.send_keys(password)
```

```
            login_button = WebDriverWait(self.driver,
WAIT_TIMEOUT).until(EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-test-id="login-
button-submit"]')))
            login_button.click()
        except Exception as e:
             self.fail(f"Login interaction failed: {type(e).__name__} - {e}")


    def test_successful_login(self):
        del self.driver.requests
        self._perform_login(VALID_USERNAME, VALID_PASSWORD)
        try:
            dash_locator = (By.ID, 'main-content')
            WebDriverWait(self.driver, WAIT_TIMEOUT).until(
                EC.visibility_of_element_located(dash_locator)
            )
            self.assertTrue(self.driver.find_element(*dash_locator).is_displayed())
        except Exception as e:
            self.fail(f"Successful login failed validation: {type(e).__name__}")


    def test_invalid_credentials(self):
        del self.driver.requests
        self._perform_login(VALID_USERNAME, INVALID_PASSWORD)
        try:
            login_request = self.driver.wait_for_request(LOGIN_API_PATH_PART, timeout=10)
            self.assertIsNotNone(login_request.response)
            self.assertEqual(login_request.response.status_code,
EXPECTED_INVALID_LOGIN_STATUS)
        except TimeoutException:
            self.fail(f"Login request not detected")
        except Exception as e:
             self.fail(f"Network check failed: {type(e).__name__} - {e}")
        try:
            WebDriverWait(self.driver, 2).until(EC.presence_of_element_located((By.ID,
'username')))
        except TimeoutException:
             self.fail("Not on login page after invalid attempt")


    def test_account_lockout(self):
        for i in range(LOCKOUT_ATTEMPTS):
            del self.driver.requests
            self._perform_login(VALID_USERNAME, INVALID_PASSWORD)
            try:
                login_request = self.driver.wait_for_request(LOGIN_API_PATH_PART, timeout=10)
                self.assertIsNotNone(login_request.response)
                self.assertEqual(login_request.response.status_code,
EXPECTED_INVALID_LOGIN_STATUS)
```

```
                WebDriverWait(self.driver,
WAIT_TIMEOUT).until(EC.presence_of_element_located((By.ID, 'username')))
                time.sleep(0.2)
            except Exception as e:
                 self.fail(f"Failure during lockout attempt {i + 1}: {type(e).__name__} -
{e}")

        del self.driver.requests
        self._perform_login(VALID_USERNAME, VALID_PASSWORD)
        try:
            lockout_check_request = self.driver.wait_for_request(LOGIN_API_PATH_PART,
timeout=10)
            self.assertIsNotNone(lockout_check_request.response)
            self.assertEqual(lockout_check_request.response.status_code,
EXPECTED_LOCKED_OUT_STATUS,
                             f"Observed status {lockout_check_request.response.status_code},
expected {EXPECTED_LOCKED_OUT_STATUS} (lockout)")
        except Exception as e:
             self.fail(f"Final lockout check failed: {type(e).__name__} - {e}")


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

if __name__ == '__main__':
    import sys
    unittest.main(verbosity=2)
```