Web Browser Automation - организация среды тестирования на основе SeleniumBase

В данном документе предлагается решение вопроса об организации автоматического тестирования с использованием фреймворка Selenium в стеке Python с применением библиотеки SeleniumBase и сопутствующих.

Выбор технических методов и инструментов обоснован характеристикой, которая составлена в соответствии с требованиями.

ПРЕДМЕТ ТЕСТИРОВАНИЯ

Для понимания предмета тестирования приведём здесь примеры функциональностей, к которым возможно применить автотесты:

- Обновление ПО;
- Смена IP;
- Изменение сетевых настроек;
- Работа журнала;
- Доступность SNMP ресурсов;
- Коды ответов на HTTP API.

ТРЕБОВАНИЯ К ТЕСТИРОВАНИЮ

Основываются на общих требованиях, определённых в документе:
[PRJ:28056035] Автоматическое тестирование новых прошивок для устройтсв NetPing

N	Требование	Соответствие	Примечания
1	Возможность запускать удалённо вручную тесты для прошивок, выложенных в любых разделах репозитория	Да	Наиболее сложные виды тестов, с технической точки зрения - тесты Drag'n Drop файла прошивки с рабочего стола ПК в веб-интерфейс - осуществляются при помощи библиотеки Selenium-Python-Helium. Краткая инструкция даётся в разделе How-To

2	Для начала теста выполнить автоматическую загрузку прошивки на соответствующее устройство тестового стенда и проверить что обновление прошло успешно	Да	
3	Зафиксировать отчёт о тестировании прошивки в журнале тестирования. Зафиксировать возникающие ошибки	Ожидается опытное подтверждение	
4	Возможность просмотра журнала тестирования и сообщения о возникающих ошибках	Ожидается опытное подтверждение	
5	Возможность сохранения журнала тестирования вместе с прошивкой, для подтверждения факта успешного прохождения тестов прошивкой	Ожидается опытное подтверждение	

ХАРАКТЕРИСТИКА ИНСТРУМЕНТОВ И МЕТОДОВ

В соответствии с общими Требованиями (из предыдущего раздела), инструменты и методы тестирования представлены несколькими аспектами:

Порог вхождения

Порог вхождения в технологический стек определяется уровнем сложности написания тестов рядовым сотрудником NetPing. Условимся считать рядовым сотрудником начинающего программиста, способного составить короткую программу на языке Python.

Доступность использования

Доступность использования автотестов для различных групп пользователей, определяется возможностью запуска процесса тестирования с минимальным набором знаний и инструментов, таких как:

1. Доступ по SSH к машине тестирования;
2. Запуск программы из командной строки;
3. Проверка отчётов о тестировании.

Object-Oriented Design

Данный аспект характеризует процесс создания и использования тестов с точки зрения управления предметами программирования, подразумевающей следующие аспекты:

Определение класса

В программном коде каждый тест или группа тестов должны быть представлены отдельным классом; название класса должно отражать название тестируемой функциональности и сопровождаться коротким комментарием, например:

Пример определения класса с комментарием

```
4
5   class ApplyFirmware(BaseCase):
6       '''
7       Drag'n drop firmware file. Check if the firmware is applied successfully.
8       '''
```

Определение констант

Константы для тестирования должны задаваться статическими атрибутами класса и следовать сразу после объявления класса, например:

Пример определения констант

```
4
5   class ApplyFirmware(BaseCase):
6       '''
7       Drag'n drop firmware file. Check if the firmware is applied successfully.
8       '''
9       firmware_valid   =  True
10      web_page_address =  "http://visor40:ping40@tst.alentis.ru:8040/update.html"
11      firmware_path    =  "/home/anton/python_projects/np-autotest/data/"
12      fw_fake_filename =  "[Pub] DKSF 70.7.4.R.npu" # required becase of bug in the drag-n-drop feature
13      fw_real_filename =  "test_firmware4.npu"
14
```

Названия тестов

Названия тестов должны как можно точнее отражать выполняемое действие и записываться заглавными буквами с тем, чтобы они были хорошо различимы в потоке данных, например:

Пример наименования теста и вывод результатов в консоли

```
17      def test_1_DRAGN_DROP_FIRMWARE_VALID(self):
18          # ================================== #
19          hm.start_firefox(self.web_page_address, headless=True)
20          browser = hm.get_driver()
```

```
dragn_drop_firmware.py::ApplyFirmware::test_1_DRAGN_DROP_FIRMWARE_VALID PASSED     [ 20%]
dragn_drop_firmware.py::ApplyFirmware::test_2_FIRMWARE_LOADED PASSED               [ 40%]
dragn_drop_firmware.py::ApplyFirmware::test_3_FIRMWARE_CODEBASE_REPLACED PASSED    [ 60%]
dragn_drop_firmware.py::ApplyFirmware::test_4_NEW_WEBINTERFACE_PAGES_LOADED PASSED [ 80%]
dragn_drop_firmware.py::ApplyFirmware::test_5_FIRMWARE_APPLIED PASSED              [100%]

============================== 5 passed in 104.96s (0:01:44) ===============================
```

Многошаговые тесты

Много шаговые сценарии тестирования следует разбивать на отдельные тесты, задавая их последовательное выполнение при помощи префиксов вида “test_1_”, test_2_” и т.д. добавленных к названию теста;

Пример кода

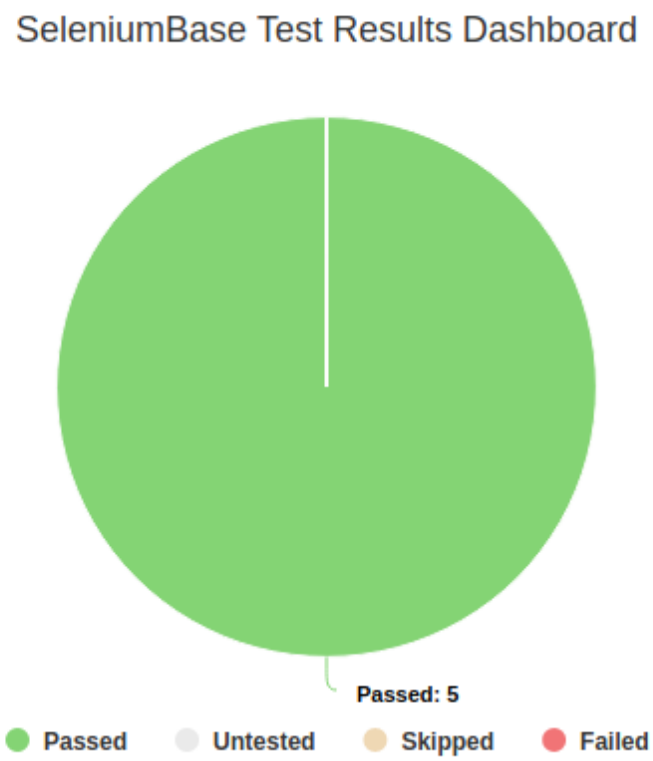
- Тест прошивки устройства (листинг)
- Репозитарий среды тестирования

Отчёты о тестировании

Выбранная библиотека SeleniumBase позволяет получать отчёты о тестировании в следующем виде:

1. HTML Dashboard

pytest test_suite.py --dashboard



Result	Test
Passed	dragn_drop_firmware.py::ApplyFirmware::test_1_DRAGN_DROP_FIRMWARE_VALID
Passed	dragn_drop_firmware.py::ApplyFirmware::test_2_FIRMWARE_LOADED
Passed	dragn_drop_firmware.py::ApplyFirmware::test_3_FIRMWARE_CODEBASE_REPLACED
Passed	dragn_drop_firmware.py::ApplyFirmware::test_4_NEW_WEBINTERFACE_PAGES_LOADED
Passed	dragn_drop_firmware.py::ApplyFirmware::test_5_FIRMWARE_APPLIED

Last updated: Wednesday, 10 March 2021 at 1:14:11 PM (MSK, UTC+03:00)

Status: Test Run Complete: **Success!** (All tests passed)

Generated by: **SeleniumBase**

2. XML Report (для Jenkins)

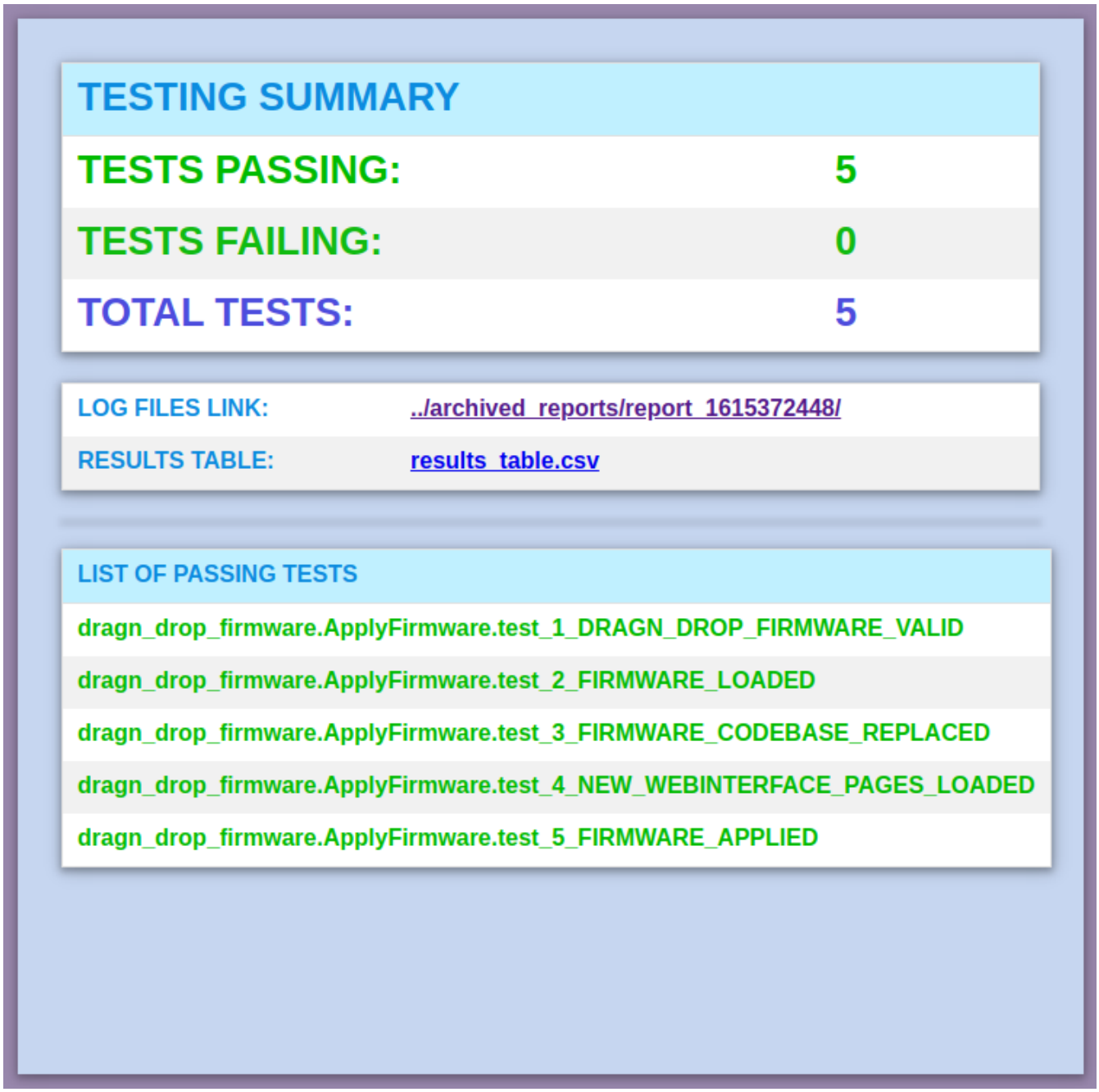
pytest test_suite.py --junit-xml=report.xml

If viewing pytest html reports in Jenkins, you may need to configure Jenkins settings for the html to render correctly. This is due to Jenkins CSP changes.

You can also use `--junit-xml=report.xml` to get an xml report instead. Jenkins can use this file to display better reporting for your tests.

3. Nosetest Reports

nosetests test_suite.py --report



4. CSV Reports

nosetests test_suite.py --report (см. далее в папке /archived_reports)

	Num	Result	Stacktrac	Screensh	URL	Browser	Epoch Time	Duration	Test Case Address	Additional Info
2	1	Passed!	*	*	*	chrome	1615372353	5.426s	dragn_drop_firmware.ApplyFirmware.test_1_DRAGN_DROP_FIRMWARE_VALID	*
3	2	Passed!	*	*	*	chrome	1615372395	41.986s	dragn_drop_firmware.ApplyFirmware.test_2_FIRMWARE_LOADED	*
4	3	Passed!	*	*	*	chrome	1615372412	16.222s	dragn_drop_firmware.ApplyFirmware.test_3_FIRMWARE_CODEBASE_REPLACED	*
5	4	Passed!	*	*	*	chrome	1615372446	34.033s	dragn_drop_firmware.ApplyFirmware.test_4_NEW_WEBINTERFACE_PAGES_LOADED	*
6	5	Passed!	*	*	*	chrome	1615372448	1.830s	dragn_drop_firmware.ApplyFirmware.test_5_FIRMWARE_APPLIED	*

ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ

- Пример ручного тестирования описан здесь:
  [IPRJ:23776943] Инструкция по тестированию 2/PWR-220 v13/GSM3G & NetPing 2/PWR-220 v12/ETH

Рекомендации по запуску виртуальной среды Python

См. Readme в репозитарии: https://github.com/antoncom/np_autotest

Полезные опции инструмента SeleniumBase

- Interactive Product Tours¶
- Interactive Charts
- Полный перечень опций SeleniumBase

Прочие инструменты тестирования:

- Кратко "Selenium, Selenoid, Selenide, Selendroid… Что все это значит?": https://habr.com/ru/post/463525/
- Полезный конспект доклада “Selenoid — сотни параллельных UI-тестов легко и быстро"
- Подкаст тестировщиков - аудио-обсуждение - ответы на вопросы создателей Selenide: http://radio-qa.com/vypusk-20-luchshe-selenide-v-rukah-chem-selenium-v-nebe/
- Selene - Selenide port to Python. Интересный проект, надо присмотреться.
- Любопытный опыт: Как написать Selenide на Python за 15 минут (видеокаст от автора Selene)