

Testowanie aplikacji internetowych z wykorzystanie Selenium oraz języka Python

autor:
Bernard Jabłoński
Warszawa 2022r.

Przygotowanie środowiska

System Operacyjny: Linux Mint 20.3 Cinnamon
Python 3.8.10
pip 20.0.2

stworzone środowisko wirtualne dla projektu (venv) zgodnie z instrukcją:

<https://www.infoworld.com/article/3239675/virtualenv-and-venv-python-virtual-environments-explained.html>

```
python -m venv ./venv
```

Środowisko nie zostało w pełni stworzone ze względu na braki w pakiecie. Wymagało jeszcze doinstalowania jednego elementu:

```
apt install python3.8-venv
```

bez tego elementu nie generowały się pliki do aktywacji środowiska wirtualnego.

aktywacja środowiska w terminalu:

```
source venv/bin/activate
```

Przygotowanie sterowników do przeglądarek:

instrukcja:

<https://selenium-python.readthedocs.io/installation.html>

pobrane wersje sterowników:

ChromeDriver 102.0.5005.61

geckodriver 0.31.0

Instrukcja mówi aby pobrane pliki umieścić w katalogach

/usr/bin

lub

/usr/local/bin - wybrane przeze mnie

Jaki powstał z tym problem?

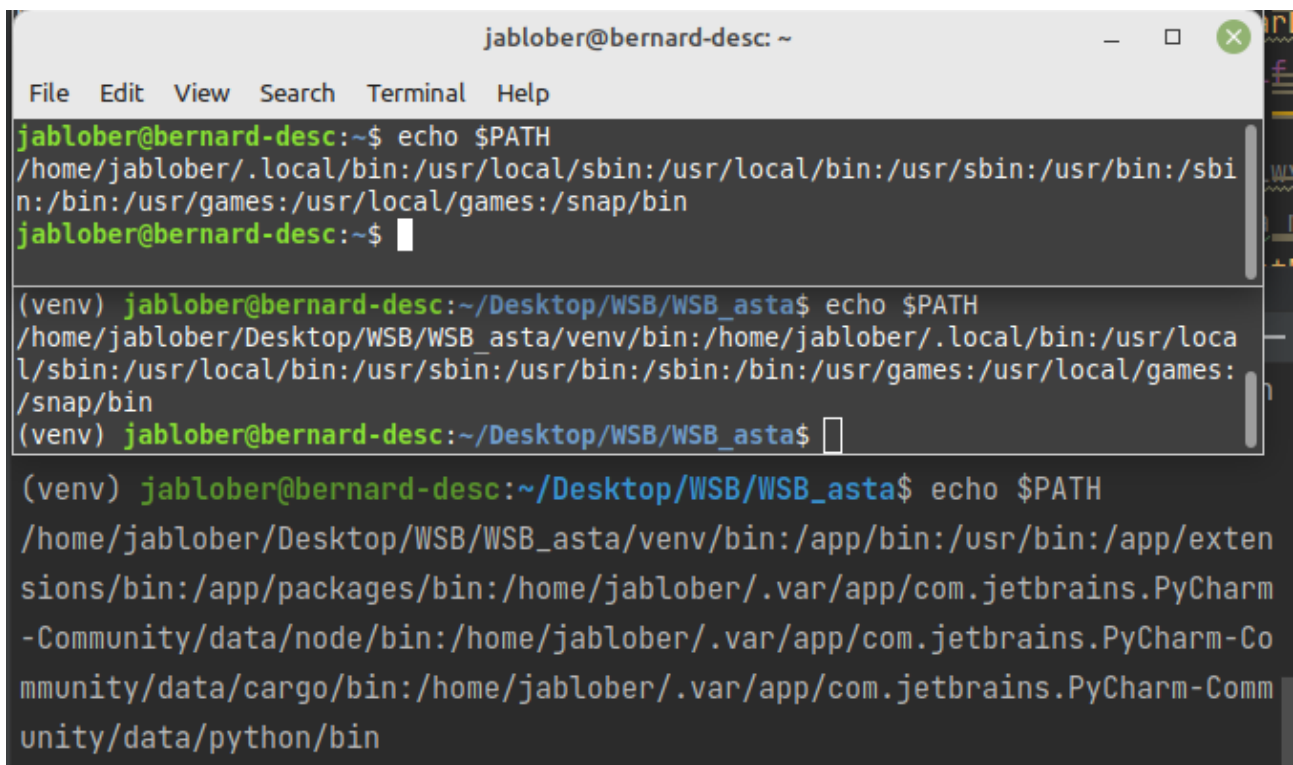
Weryfikacja działania środowiska z poziomu terminala systemowego jest poprawne i wszystko działa jak jest zakładane.

Przy wykorzystaniu terminala w PyCharm zwraca błędy. Dlaczego?

Weryfikacja:

Środowisko wirtualne aktywowane w terminalu działa poprawnie. Odpytywane o sterowniki przeglądarek - nie podaje danych -> pierwszy ślad błędu.

Dochodzenia wykazało że w PyCharm nie widzi katalogu gdzie umieściłem sterowniki oraz pomimo że wskazuje na takie samo środowisko wirtualne to zaczytuje inne dane co widać w na załączonym obrazku:



```
jablober@bernard-desc: ~  
File Edit View Search Terminal Help  
jablober@bernard-desc:~$ echo $PATH  
/home/jablober/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin  
jablober@bernard-desc:~$  
  
(venv) jablober@bernard-desc:~/Desktop/WSB/WSB_asta$ echo $PATH  
/home/jablober/Desktop/WSB/WSB_asta/venv/bin:/home/jablober/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin  
(venv) jablober@bernard-desc:~/Desktop/WSB/WSB_asta$  
  
(venv) jablober@bernard-desc:~/Desktop/WSB/WSB_asta$ echo $PATH  
/home/jablober/Desktop/WSB/WSB_asta/venv/bin:/app/bin:/usr/bin:/app/extensions/bin:/app/packages/bin:/home/jablober/.var/app/com.jetbrains.PyCharm-Community/data/node/bin:/home/jablober/.var/app/com.jetbrains.PyCharm-Community/data/cargo/bin:/home/jablober/.var/app/com.jetbrains.PyCharm-Community/data/python/bin
```

Screen przedstawia 3 widoki wywołania `echo $PATH` patrząc od góry:

1. wywołanie z poziomu terminala systemowego na głównym środowisku systemowym
2. wywołanie z poziomu terminala systemowego na wirtualnym środowisku venv przygotowanym do testów
3. wywołanie z poziomu terminala IDE PyCharm na wirtualnym środowisku venv.

Jak widać są znaczące różnice w tym co widzą dane środowiska

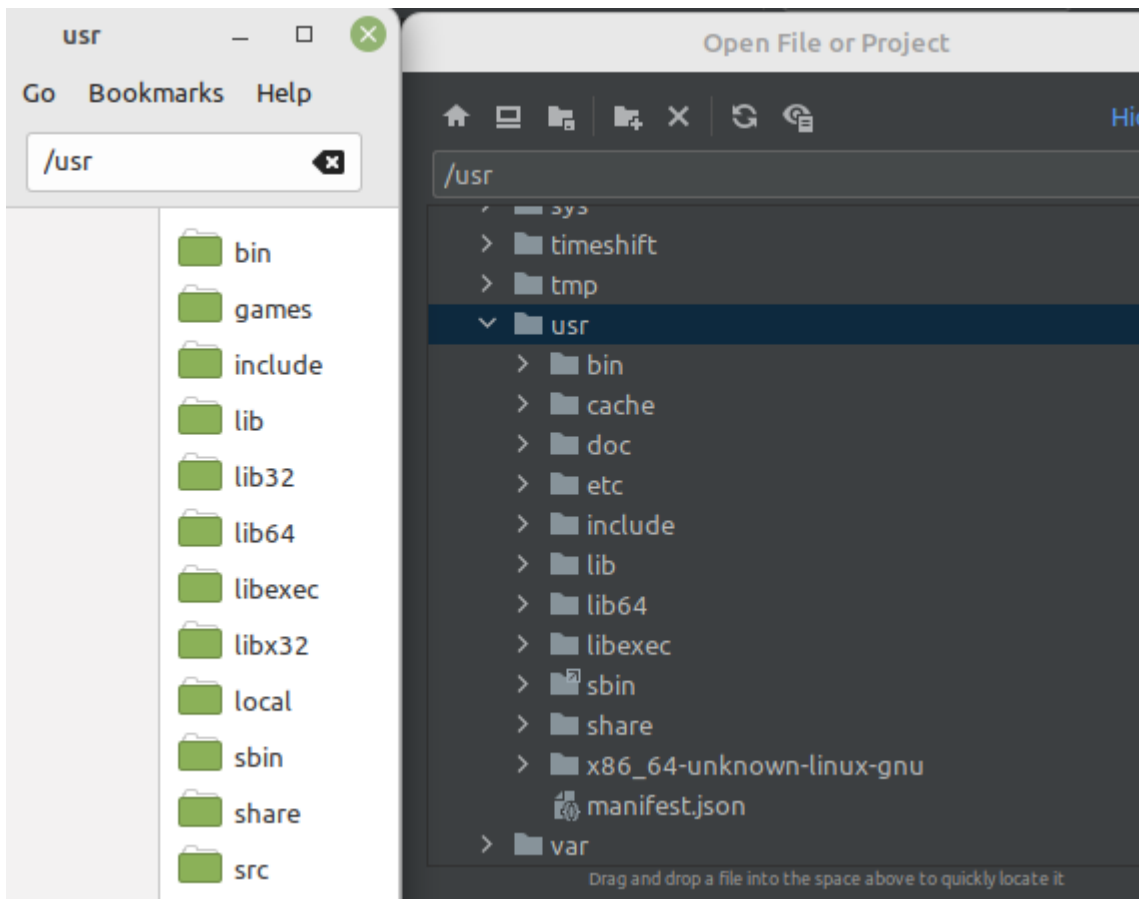
Dlaczego taki problem występuje?

W IDE nie widzi / nie ma dostępu do niektórych zasobów systemu. W tym przypadku był problem z dostępem do:

`/usr/local/bin`

Dlaczego tak się stało?

Podejrzenie pada na formę instalacji aplikacji czyli Flatpak. Weryfikacja na innych IDE takich jak ATOM czy VSC ta przestrzeń nie jest widoczna (również zainstalowane za pomocą Flatpak)



Screen prezentuje 2 widoki. Patrząc od lewej strony:

1. widok zasobów katalogu /user z poziomu systemu operacyjnego
2. widok zasobów katalogu /user z poziomu IDE PyCharm (instalacja Flatpak)

Widać różnice w dostępie do katalogów systemowych w przypadku instalacji aplikacji za pomocą Flatpak'a.

Rozwiązanie

Problem został rozwiązany przez zainstalowanie IDE w inny sposób omijając narzędzie Flatpak. Czyli instalację bezpośrednio na systemie operacyjnym.

Znalazłem też inną możliwość rozwiązania problemu:

w kodzie można dodać bezwzględną lub względną ścieżkę do sterowników w przestrzeni widocznej dla IDE w danej wersji gdzie będzie widziana przez program.

Przykłady odwołania się do takich zasobów umieszczenia sterowników w kodzie:

```
self.driver = webdriver.Chrome('./chromedriver')
lub
os.environ["PATH"]+="C:/SeleniumDrivers"
driver=webdriver.Chrome()
```

Testy

Testy wprowadzenia i weryfikacji wyników danych po wykonaniu akcji

Warunki wstępne:

1. Wejście przez stronę główną <https://demo.seleniumeasy.com/>
2. zaakceptowanie pojawiającego się komunikatu powitalnego.

Wykonanie testów

- przejście do właściwej podstrony przeklikując się przez elementy strony:
Basic > Simple Form Demo
- podstrona docelowa <https://demo.seleniumeasy.com/basic-first-form-demo.html>

Dla testu pierwszego:

pole przyjmuje dowolną wartość i ma ją wyświetlić w komunikacie poniżej po aktywacji przycisku.

Dla testu drugiego:

dwa pola przyjmujące wartości cyfrowe które są przeliczane jako suma wprowadzonych wartości i wyświetlane poniżej jako wynik obliczenia

Kod:

```
import unittest
from selenium import webdriver
from selenium.webdriver.common.by import By
from time import sleep

frazzaTestowa = "Paczkord"
cyfra1 = "65"
cyfra2 = "45"

class APRegistration(unittest.TestCase):
    """Analogia: Przypadek/scenariusz testowy"""

    # Przygotowanie do każdego testu
    # (Warunki wstępne)
    def setUp(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://demo.seleniumeasy.com/")
        self.driver.maximize_window()
        sleep(1)
        self.driver.find_element(By.ID, "at-cv-lightbox-close").click()
        self.driver.implicitly_wait(2)

    # Sprzątanie po każdym teście
    def tearDown(self):
```

```
self.driver.quit()
```

```
def testSingleInputField(self):
    self.driver.find_element(By.ID, "basic_example").click()
    sleep(1)
    self.driver.find_element(By.XPATH, '//*[@id="basic"]/div/a[1]').click()
    sleep(1)
    wiadomosc = self.driver.find_element(By.ID, "user-message")
    wiadomosc.clear()
    wiadomosc.send_keys(frazaTestowa)
    self.driver.find_element(By.XPATH, '// *[@id = "get-input"] / button').click()
    SingleInputField = self.driver.find_element(By.ID, "display").text
    print(SingleInputField)
    assert SingleInputField == frazaTestowa
    sleep(2)
```

```
def testTwoInputFields(self):
    self.driver.find_element(By.ID, "basic_example").click()
    sleep(1)
    self.driver.find_element(By.XPATH, '//*[@id="basic"]/div/a[1]').click()
    sleep(1)
    sum_1 = self.driver.find_element(By.ID, "sum1")
    sum_1.send_keys(cyfra1)
    sum_2 = self.driver.find_element(By.ID, "sum2")
    sum_2.send_keys(cyfra2)
    sumTestA = int(cyfra2) + int(cyfra1)
    self.driver.find_element(By.XPATH, '//*[@id="gettotal"]/button').click()
    sumTestB = self.driver.find_element(By.ID, "displayvalue").text
    print(str(sumTestA) + " vs " + str(sumTestB))
    sleep(2)
    assert str(sumTestA) == str(sumTestB)
```

```
# Uruchom test jesli uruchamiamy
# ten plik
if __name__ == "__main__":
    unittest.main()
```

Podsumowanie testów

testy wykonane poprawnie i przeszły pozytywnie.

Kod został też umieszczony na github:

https://github.com/BernardJablonski/WSB_asta