Testowanie Aplikacji Mobilnych z użyciem Appium

mgr inż. Bartosz Bryniarski | email: bartosz.bryniarski@wroclaw.merito.pl | bartosz@bryniarski.pl | tel.: 607 052 360

Czym jest Appium?

Appium to otwartoźródłowe narzędzie do automatyzacji testów aplikacji mobilnych. Pozwala ono na testowanie aplikacji natywnych, hybrydowych i webowych na platformach iOS i Android, używając tego samego interfejsu API.

Umożliwia pisanie testów w różnych językach programowania, w tym w Pythonie, Java, Ruby i innych.

Dlaczego Appium?

Cross-platform Testing: Appium pozwala na pisanie testów, które działają zarówno na iOS jak i Androidzie, bez konieczności zmiany kodu.

Język i Framework Agnostic: Możliwość wykorzystania różnych języków programowania i frameworków testowych.

Realne Urządzenia i Emulatory: Appium umożliwia testowanie zarówno na rzeczywistych urządzeniach, jak i emulatorach/symulatorach.

Standard Selenium WebDriver: Wykorzystanie dobrze znanego interfejsu WebDriver zapewnia łatwość integracji i użycia.

Unittest w Pythonie

Unittest jest wbudowanym modułem Pythona, używanym do tworzenia i uruchamiania testów jednostkowych. Jest to kluczowy element praktyk programistycznych, takich jak test-driven development (TDD), i jest powszechnie stosowany w projektach Pythona w celu zapewnienia stabilności i niezawodności kodu.

Funkcje i Cechy Unittest

Struktura Testów:
Unittest pozwala na tworzenie testów, grupując je w klasach jako metody. Każda metoda testowa w klasie to oddzielny test jednostkowy.
Testy są zwykle organizowane w modułach testowych, które są skryptami Pythona.

Metody Testowe:
Typowa metoda testowa w Unittest zaczyna się od słowa test, np. test_functionality, co pozwala na automatyczne wykrywanie i uruchamianie testów.

SetUp i TearDown:
setUp(): Metoda wywoływana przed każdym testem w klasie. Służy do przygotowania środowiska testowego, np. inicjalizacji obiektów.
tearDown(): Metoda wywoływana po każdym teście, służąca do sprzątania po teście, np. zamykania połączeń z bazą danych.

Definicja klasy testowej

```
import unittest

class MyTestClass(unittest.TestCase):

    def setUp(self):
        # Kod inicjalizujący
        pass

    def test_something(self):
        # Przykładowy test
        self.assertEqual(1 + 1, 2)

    def tearDown(self):
        # Kod sprzątający
        pass

if __name__ == '__main__':
    unittest.main()
```

Ustawienia testów

```
 def setUp(self):
    options = UiAutomator2Options()
    options.platform_name = 'Android'
    options.device_name = 'emulator-XXX’
    options.app_package = 'com.example.myapplication’
    options.app_activity = '.MainActivity'
    options.automation_name = 'UiAutomator2'

    self.driver =
webdriver.Remote('http://localhost:4723', options=options)
```

Asercje Porównania:

assertEqual(a, b)	Sprawdza, czy a jest równe b.
assertNotEqual(a, b)	Sprawdza, czy a nie jest równe b.
assertTrue(x)	Sprawdza, czy x jest prawdziwe (True).
assertFalse(x)	Sprawdza, czy x jest fałszywe (False).
assertIs(a, b)	Sprawdza, czy a i b to ten sam obiekt.
assertIsNot(a, b)	Sprawdza, czy a i b to nie ten sam obiekt.
assertIsNone(x)	Sprawdza, czy x jest None.
assertIsNotNone(x)	Sprawdza, czy x nie jest None.
assertIn(a, b)	Sprawdza, czy a znajduje się w b.
assertNotIn(a, b)	Sprawdza, czy a nie znajduje się w b.
assertIsInstance(a, b)	Sprawdza, czy a jest instancją b.
assertNotIsInstance(a, b)	Sprawdza, czy a nie jest instancją b.

Asercje Dokładności:

assertAlmostEqual(a, b)	Sprawdza, czy a i b są prawie równe (domyślna tolerancja to 7 miejsc po przecinku).
assertNotAlmostEqual(a, b)	Sprawdza, czy a i b nie są prawie równe.

Testowanie Wyjątków:

assertRaises(exc, fun, *args, **kwds)
Sprawdza, czy wywołanie fun(*args, **kwds) zgłasza wyjątek exc.

assertRaisesRegex(exc, regex, fun, *args, **kwds)
Podobnie jak assertRaises, ale dodatkowo sprawdza, czy komunikat wyjątku pasuje do regex.

Testowanie Ostrzeżeń:

assertWarns(warn, fun, *args, **kwds)
Sprawdza, czy wywołanie fun(*args, **kwds) generuje ostrzeżenie warn.

assertWarnsRegex(warn, regex, fun, *args, **kwds)
Podobnie jak assertWarns, ale dodatkowo sprawdza, czy komunikat ostrzeżenia pasuje do regex.

Metody Pomocnicze:

assertRegex(text, regex)
Sprawdza, czy text pasuje do wyrażenia regularnego regex.

assertNotRegex(text, regex)
Sprawdza, czy text nie pasuje do wyrażenia regularnego regex.

assertCountEqual(a, b)
Sprawdza, czy a i b mają te same elementy w tej samej liczbie, niezależnie od ich kolejności.

Metody dla Typów Kolekcji:

assertListEqual(a, b)	Sprawdza, czy listy a i b są równe.
assertTupleEqual(a, b)	Sprawdza, czy krotki a i b są równe.
assertSetEqual(a, b)	Sprawdza, czy zbiory a i b są równe.
assertDictEqual(a, b)	Sprawdza, czy słowniki a i b są równe.

Inne Metody Testowe:

setUp()
Metoda wywoływana przed każdym testem, służy do przygotowania środowiska testowego.

tearDown()
Metoda wywoływana po każdym teście, służy do sprzątania po teście.

skipTest(reason)
Pomija wykonanie testu.

fail(msg=None)
Powoduje niepowodzenie testu z opcjonalnym komunikatem msg.

Dziękuję za uwagę!