# T6 - Testarea interfeței grafice a unei aplicații web

Ancuța Vlad-Adrian

Informatica ID

Anul 3, Grupa 2

# Cuprins

1. **Introducere**
2. **Studiu comparativ între Selenium și Playwright  
   2.1. Prezentare generală  
   2.2. Caracteristici tehnice și diferențe relevante**
3. **Alegerea framework-ului pentru implementare**
4. **Implementarea testelor de interfață  
   4.1. Teste de funcționalitate – login corect  
   4.2. Teste de validare – formular  
   4.3. Teste de navigare între pagini  
   4.4. Teste de compatibilitate între browsere**
5. **Raport de folosire a unui tool de AI**
6. **Bibliografie**

# Introducere

Interfața grafică a unei pagini web este mediul unde utilizatorul interacționează cu aplicația web prin elemente vizuale cum ar fi butoane, checkbox-uri, textbox-uri, formulare si tabele.

Buna funcționare a interfeței grafice este unul dintre cele mai importante aspecte într-o aplicație web, unde se pune foarte mult accent pe experiența vizuală a utilizatorului. Aici intervine testarea automată a interfeței grafice a unei pagini web, care are ca scop detectarea problemelor ce pot aparea în cadrul interefetelor grafice, eliminând sarcinile repetitive.

Scopul acestei lucrări este de a realiza o comparație între două framework-uri de testare pentru interfața grafică a unei aplicații web și cele doua framework-uri alese pentru comparație sunt Selenium si Playwright.

# Studiu comparativ între Selenium și Playwright

## Prezentare generală

Primul framework prezentat este Selenium. Acesta este un instrument folosit pentru a simula interacțiunea dintre utilizator și aplicația web.

Interfața WebDriver se află la temelia Selenium. Prin această interfață un browser este manipulat de cod asemenea modului în care ar fi manipulat de un utilizator.[1]

Această intefață este obiect-orientată, ușor de utilizat pentru a permite controlul browser-ului. [2]

Selenium este compatibil cu multiple limbaje de programare cum ar fi Python, Java, C#, Ruby, JavaScript sau Kotlin.

Playwright este cel de-al doilea framework prezentat. Acesta a fost creat de Microsoft și are același scop ca Selenium, adică testarea aplicațiilor web. [3]

Playwright poate fi folosit pe orice browser și în orice sistem de operare.

## Caracteristici tehnice și diferențe relevante

Deși la o primă vedere par similare, totuși Selenium si Playwright prezintă diferențe atât tehnice, cât și în ceea ce privește atat sintaxa.

În primul rând, pe parte de instalare, este mult mai ușor de instalat Playwright decât Selenium. Playwright va necestita doar node.js instalat și se poate rula direct comanda “npm init playwright@latest”, fara a fi nevoie de un WebDriver, pe cand Selenium va necesita instalarea Python/Java, instalarea unui IDE si instalarea WebDriver-ului pentru browserul specific, ceea ce face Selenium mai anevoios în procesul de instalare. [4]

În ceea ce privește simplificarea procesului de testare, cele două framework-uri au abordări asemănătoare. Playwright oferă Codegen, o soluție care urmărește activitatea utilizatorului pe site și care generează automat, în timp real, codul sursă, pe baza activității. Selenium folosește Selenium IDE care înregistrează și redă acțiunile utilizatorului fără a fi nevoie de cunoștiințe de programare. [4]

Pe partea de selectare a elementelor, există diferențe între cele două framework-uri: dacă în Selenium utilizatorul trebuie să specifice programului să aștepte până când un element s-a încărcat (explicit wait), Playwright știe că trebuie să aștepte și nu va mai necesita cod suplimentar pentru această operațiune (auto-wait). [4]

În ceea ce privește limbajele de programare și browserele suportate, Playwright vine cu suport integrat pentru Chrome, Firefox și Safari și suportă limbajele de programare populare cum ar fi Python, Java, Javascript si .NET. Pe de altă parte, Selenium are nevoie de instalarea manuală a WebDriver-elor pentru browsere, dar oferă suport pentru mai multe limbaje de programare (Ruby și C#, pe lângă cele suportate de Playwright). [4]

Pe partea de execuție paralelă, Playwright vine nativ cu acest feature, pe când Selenium va avea nevoie de un framework extern cum ar fi TestNG, ceea ce face Playwright mai ușor de folosit când vine vorba de testare paralelă. Totuși, Selenium oferă mai mult control – în Playwright doar trebuie specificat numărul de workers care vor lucre la testele paralele iar în Selenium va fi nevoie de completarea manuală a unui fișier XLM pentru a permite și a configura execuția paralelă. [4]

Din punct de vedere al testelor ce pică intermitent, Playwright oferă varianta de a re-încerca testele eșuate, pe lângă soluțiile de auto-wait, waitForLoadState și waitForSelector. Selenium nu dispune de aceste soluții. Astfel, Playwright gestionează mai bine testele ce pică intermitent iar intervenția programatorului este minimă. [4]

## Diferențele de sintaxă între Selenium și Playwright

Între Plawright și Selenium există diferențe clare de sintaxă iar în tabelul de mai jos se pot vedea aceste diferențe

|  |  |  |
| --- | --- | --- |
| **Operațiune** | **Selenium** | **Playwright** |
| Căutare după ID | driver.find\_element(By.ID,"id") | page.locator("#id") |
| Căutare după nume | driver.find\_element(By.NAME, "name") | page.locator("[name=\"name\"]") |
| Click | element.click() | page.click("button#id") |
| Introducere text | element.send\_keys("text") | page.fill("#search", "text") |
| Căutare după selector CSS | driver.find\_element(By.CSS\_SELECTOR, ".class") | page.locator(".class") |
| Căutare după XPath | driver.find\_element(By.XPATH, "//input[@type='submit']") | |  | | --- | |  |  |  | | --- | | page.locator("//input[@type='submit']") | |

[5]

Exemplu cu Selenium

driver.find\_element(:id, username).send\_keys("agileway")  
driver.find\_element(:name, username).send\_keys("testwise")  
driver.find\_element(:xpath, "//input[@value='Sign in']").click  
expect(driver.find\_element(:tag\_name, "body").text).to include("Signed in")

[6]

Exemplu cu Playwright

await driver.fill("#username", "agileway")  
await driver.fill("#password", "testwise")  
await driver.click("input:has-text('Sign in')")  
await driver.textContent("body").then(function(body\_text) {  
 assert(body\_text.contains("Signed in"))  
})

[6]

# Motivația alegerii framework-ului pentru implementare

Pentru acest proiect am ales Playwright datorită modului său ușor de instalat și de folosit, ideal pentru o persoană cu mai puțină experiență, datorită simplității codului și datorită integrării rapide cu framework-uri de testare cum ar fi Pytest.

De asemenea, sunt mult mai familiarizat cu acest framework datorita unei scurte experiențe de 3 luni ca tester de interfață web, perioadă în care am avut ocazia să lucrez cu el si cu Pytest, în cadrul unui proiect real.

# Implementarea testelor cu Playwright.

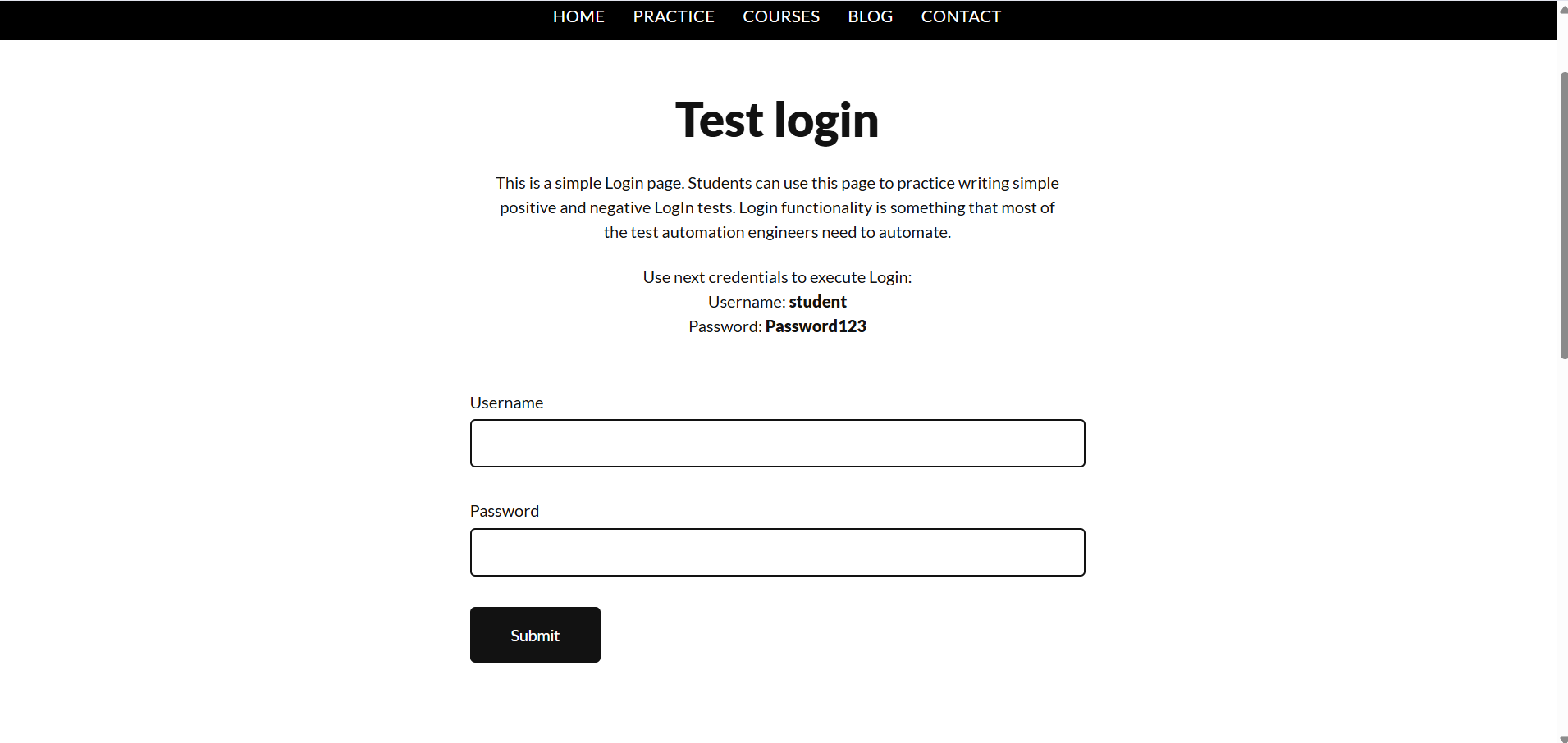
### Teste de login

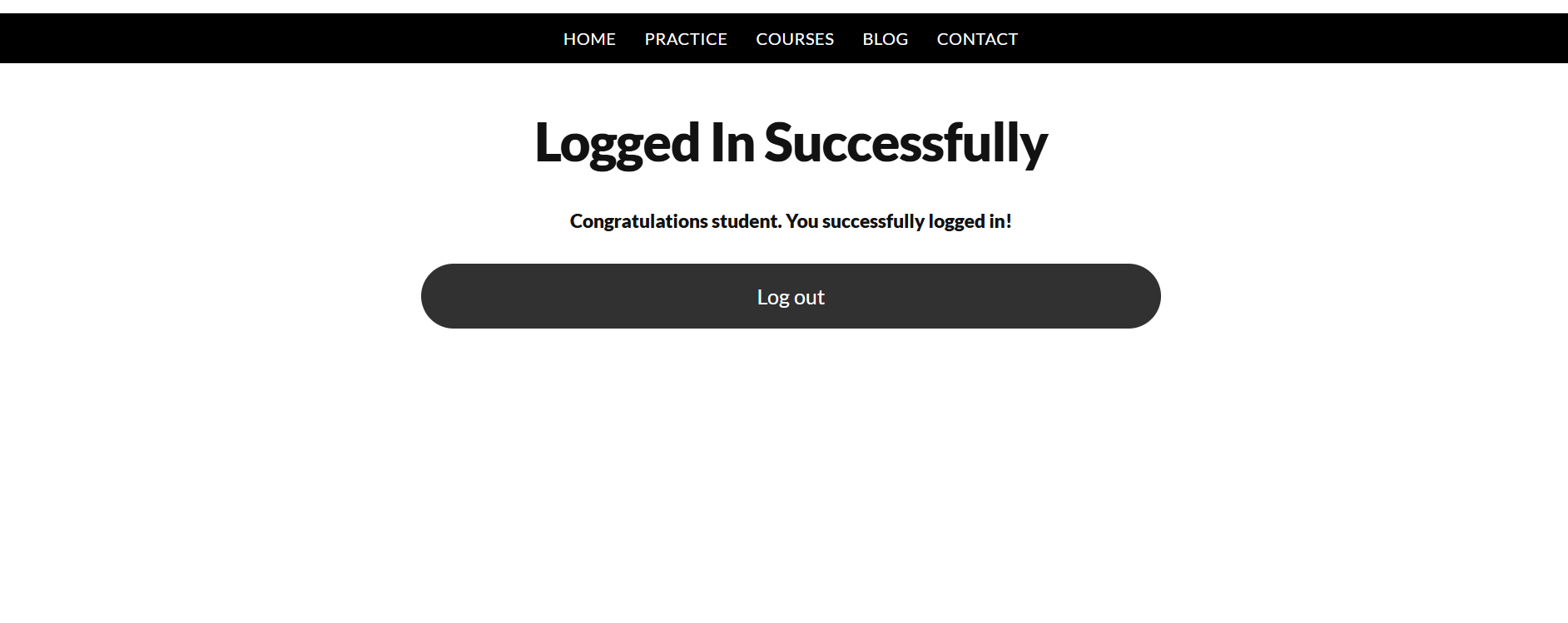
Pentru a testa cu Playwright și Pytest funcționalitatea de login, voi folosi <https://practicetestautomation.com/practice-test-login/> , un site special creat pentru a testa această funcționalitate.

Pentru a folosi Playwright în mod sincron (adică acțiunile să se execute una câte una) și pentru a ma folosi de ’expect’ în cadrul testelor, voi importa sync\_playwright și expect:

from playwright.sync\_api import sync\_playwright, expect

1. Testez dacă login-ul este reușit folosind credențialele corecte. Mă aștept să fiu redirecționat spre o pagină cu login successful:

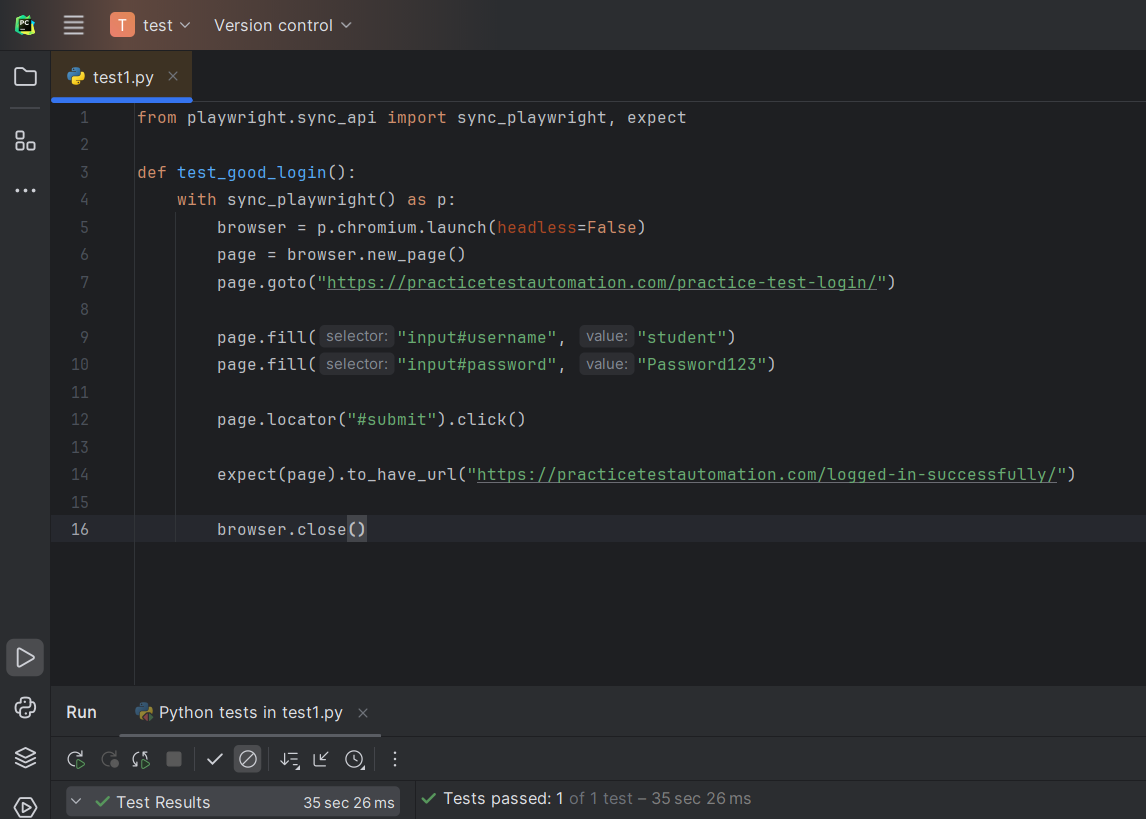




Cod test

def test\_good\_login():  
 with sync\_playwright() as p:  
 browser = p.chromium.launch(headless=False)  
 page = browser.new\_page()  
 page.goto("https://practicetestautomation.com/practice-test-login/")  
  
 page.fill("input#username", "student")  
 page.fill("input#password", "Password123")  
  
 page.locator("#submit").click()  
  
 expect(page).to\_have\_url("https://practicetestautomation.com/logged-in-successfully/")  
  
 browser.close()

Rezultat test



1. Testez dacă login-ul este nereusit si daca apare o eroare folosind parola gresită. Dacă introduc parola greșită, va apărea eroarea -Your password is invalid!-.

A screenshot of a login form

AI-generated content may be incorrect.

Cod test

def test\_bad\_password\_login():  
 with sync\_playwright() as p:  
 browser = p.chromium.launch(headless=False)  
 page = browser.new\_page()  
 page.goto("https://practicetestautomation.com/practice-test-login/")  
  
 page.fill("input#username", "student")  
 page.fill("input#password", "Password1234")  
  
 page.locator("#submit").click()  
  
 expect(page.locator('#error')).to\_be\_visible()  
 expect(page.locator('#error')).to\_have\_text("Your password is invalid!")  
  
 browser.close()

Rezultat

A screenshot of a computer program

AI-generated content may be incorrect.

1. Testez dacă login-ul este nereusit si daca apare o eroare folosind userul greșit. Dacă introduc userul greșit, va apărea eroarea -Your username is invalid!-.

A screenshot of a test login

AI-generated content may be incorrect.

Cod test

def test\_bad\_username\_login():  
 with sync\_playwright() as p:  
 browser = p.chromium.launch(headless=False)  
 page = browser.new\_page()  
 page.goto("https://practicetestautomation.com/practice-test-login/")  
   
 page.fill("input#username", "student1")  
 page.fill("input#password", "Password123")  
   
 page.locator("#submit").click()  
   
 expect(page.locator('#error')).to\_be\_visible()  
 expect(page.locator('#error')).to\_have\_text("Your username is invalid!")  
   
 browser.close()

Rezultat

A screenshot of a computer program

AI-generated content may be incorrect.

### Teste de validare – formular

Pentru această parte am folosit <https://demoqa.com/automation-practice-form> care are scopul de a oferi un formular spre testare.

**În cadrul acestui subcapitol, părțile de analiză a culorii border-ului au fost generate cu ChatGPT.** Pe baza unui inspect pe site-ulfolosit, am cerut ca border-ul să aibă culoarea verde în cazul în care informația este corectă și roșu altfel.

border\_color = page.locator(locator).evaluate(  
 "el => getComputedStyle(el).borderColor"  
)  
assert border\_color == "rgb(40, 167, 69)", f"Expected green border, got {border\_color}"

**[7]**

Am definit functia go\_get\_page care se va apela in fiecare test pentru a deschide si a returna pagina pe care vreau să desfășor testele.

def go\_get\_page(p):  
 browser = p.chromium.launch(headless=False)  
 page = browser.new\_page()  
 page.goto("https://demoqa.com/automation-practice-form")  
  
 return page, browser

1. Testez la completarea corectă a datelor din formular și la apăsarea submit dacă câmpurile completate corect vor apărea cu o bordură verde. Testul se realizează pentru 3 câmpuri – nume, prenume și număr de telefon.

A screenshot of a registration form

AI-generated content may be incorrect.

Cod test

def test\_good\_values\_validated():  
 *"""  
 Se testeaza daca dupa ce se introduc date intr-un field, borderul acestuia devine verde.  
 """* with sync\_playwright() as p:  
  
 page, browser = go\_get\_page(p)  
  
 page.fill("#firstName", "Ancuta")  
 page.fill("#lastName", "Vlad")  
 page.fill("#userNumber", "1234567890")  
  
 page.locator("#submit").scroll\_into\_view\_if\_needed()  
 page.click("#submit")  
  
 page.wait\_for\_timeout(500)  
 for locator in ["#lastName", "#firstName", "#userNumber"]:  
 border\_color = page.locator(locator).evaluate(  
 "el => getComputedStyle(el).borderColor"  
 )  
 assert border\_color == "rgb(40, 167, 69)", f"Expected green border, got {border\_color}"  
  
 browser.close()

Rezultat

A screenshot of a computer program

AI-generated content may be incorrect.

1. Testez la apăsarea submit pentru un formular ce are câmpuri obligatorii necompletate dacă câmpurile necompletate vor apărea cu un border roșu. Testul se realizează pentru 3 câmpuri – nume, prenume și număr de telefon.

A screenshot of a registration form

AI-generated content may be incorrect.

Cod test

def test\_no\_mandatory\_value\_error():  
 *""""  
 Se testeaza daca apare eroare semnalata prin boder rosu al unui field  
 obligatoriu ce nu a fost completat inainte de a se apasa submit.  
 """* with sync\_playwright() as p:  
 page, browser = go\_get\_page(p)  
  
 page.locator("#submit").scroll\_into\_view\_if\_needed()  
 page.click("#submit")  
  
 page.wait\_for\_timeout(500)  
  
 for locator in ["#lastName", "#firstName", "#userNumber"]:  
 border\_color = page.locator(locator).evaluate(  
 "el => getComputedStyle(el).borderColor"  
 )  
 assert border\_color == "rgb(220, 53, 69)", f"Expected red border, got {border\_color}"  
  
 browser.close()

Rezultat

A screenshot of a computer program

AI-generated content may be incorrect.

1. Testez dacă atunci când toate câmpurile sunt completate și apăs submit se deschide un summary și dacă acel summary oferă informația corectă.

A screenshot of a registration form

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Cod test

def test\_submit\_form\_check\_summary():  
with sync\_playwright() as p:  
 page, browser = go\_get\_page(p)  
  
 page.fill("#firstName", "Ana")  
 page.fill("#lastName", "Popescu")  
 page.fill("#userEmail", "ana.popescu@example.com")  
 page.click('label[for="gender-radio-2"]')   
 page.fill("#userNumber", "0712345678")  
  
 page.locator("#submit").scroll\_into\_view\_if\_needed()  
 page.click("#submit")  
  
 page.wait\_for\_timeout(1000)  
  
 def get\_table\_value(label):  
 return page.locator(f"//td[text()='{label}']/following-sibling::td").inner\_text()  
  
 assert get\_table\_value("Student Name") == "Ana Popescu"  
 assert get\_table\_value("Student Email") == "ana.popescu@example.com"  
 assert get\_table\_value("Gender") == "Female"  
 assert get\_table\_value("Mobile") == "0712345678"  
  
 browser.close()

Rezultat

A screenshot of a computer program

AI-generated content may be incorrect.

1. În cadrul formularului, dacă email nu se completează, nu se va semnala cu un border roșu, însă acest field ar trebui să fie mandatory, deci până când se va remedia problema, voi testa această funcționalitate și o voi marca cu xfail însoțit de un mesaj. xfail este o funcționalitate a bibliotecii pytest care permite unui test să fie marcat și ignorat în caz că acesta pică din cauza unei probleme cunoscute. Pentru a dispune de xfail, am importat biblioteca pytest.

A screenshot of a form

AI-generated content may be incorrect.

Cod test

@pytest.mark.xfail(reason='email ar trebui sa fie mandatory dar nu se coloreaza rosu daca ramane necompletat')  
def test\_bad\_email\_xfail():  
  
 with sync\_playwright() as p:  
  
 page, browser = go\_get\_page(p)  
  
 page.fill("#firstName", "Ancuta")  
 page.fill("#lastName", "Vlad")  
 page.fill("#userNumber", "1234567890")  
  
 page.locator("#submit").scroll\_into\_view\_if\_needed()  
 page.click("#submit")  
  
 page.wait\_for\_timeout(500)  
  
 locator = "#userEmail"  
 border\_color = page.locator(locator).evaluate(  
 "el => getComputedStyle(el).borderColor"  
 )  
 assert border\_color == "rgb(220, 53, 69)", f"Expected red border, got {border\_color}"  
  
 browser.close()

Rezultat

A screenshot of a computer program

AI-generated content may be incorrect.

### Teste de navigare între pagini

Pentru a testa navigarea între paginile unui site web am folosit site-ul facultății.

A screenshot of a computer

AI-generated content may be incorrect.

Vreau să mă asigur că fiecare buton de aici mă trimite spre pagina corectă și când apas back, mă trimite înapoi spre site-ul facultății.

Cod test

def test\_navigate\_between\_pages():  
  
 with sync\_playwright() as p:  
 page, browser = go\_get\_page(p)  
 for option in {"planuri-de-invatamant", "admitere", "studenti", "profesori"}:  
 page.click(f"a[href='/{option}/']")  
 page.wait\_for\_timeout(1000)  
 assert page.url == f"https://fmi.unibuc.ro/{option}/"  
 page.go\_back()  
 page.wait\_for\_timeout(1000)  
 assert page.url == "https://fmi.unibuc.ro/"  
  
 browser.close()

Rezultat

A screenshot of a computer program

AI-generated content may be incorrect.

### Teste de compatibilitate între browsere

Pentru a testa compatibilitatea între browsere se va folosi testul precedent, doar că va fi rulat pe mai multe browsere (chromium, firefox, webkit).

Cod test

from playwright.sync\_api import sync\_playwright  
import pytest  
  
def test\_compatibilitate\_browsere():  
 for browser\_type in ["chromium", "firefox", "webkit"]:  
 with sync\_playwright() as p:  
  
 browser = getattr(p, browser\_type).launch(headless=False)  
 page = browser.new\_page()  
 page.goto("https://fmi.unibuc.ro/")  
  
 for option in {"planuri-de-invatamant", "admitere", "studenti", "profesori"}:  
 page.click(f"a[href='/{option}/']")  
 page.wait\_for\_timeout(1000)  
 assert page.url == f"https://fmi.unibuc.ro/{option}/"  
 page.go\_back()  
 page.wait\_for\_timeout(1000)  
 assert page.url == "https://fmi.unibuc.ro/"  
  
 browser.close()

Rezultat

A screenshot of a computer program

AI-generated content may be incorrect.

# Raport de folosire a unui tool de AI.

În cadrul acestui proiect am folosit ChatGPT de la OpenAI pentru o varietate de task-uri.

În primul rând, la partea de comparație din punct de vedere al sintaxei între Playwright și Selenium, întrucât nu am lucrat cu Selenium ci doar cu Playwright, mi-a fost mai ușor să obțin diferențele de sintaxă cu ajutorul ChatGPT.

De asemenea, ChatGPT m-a ajutat la părțile unde locatorii erau mai complicat de obținut, fiind necesar doar să copiez obiectul la care voiam să fac referire din partea de Inspect a paginii web. Acest lucru a eficientizat timpul care ar fi fost pierdut în încercarea de a selecta corect obiectul. Un exemplu ar fi:

page.locator(f"//td[text()='{label}']/following-sibling::td").inner\_text()

Tot pe partea de cod, am folosit inteligența artificială pentru a realiza mai rapid evaluări mai dificile, cum ar fi să verific dacă border-ul unui textbox are o anumită culoare:

border\_color = page.locator(locator).evaluate(  
 "el => getComputedStyle(el).borderColor"  
)  
assert border\_color == "rgb(40, 167, 69)", f"Expected green border, got {border\_color}"

De altfel, ChatGPT a fost folosit la sectiunea 4.2, testul numarul 3) , unde AI-ul a completat formularul și a ajutat la obținerea referinței pentru tabelul de summary :

def get\_table\_value(label):  
 return page.locator(f"//td[text()='{label}']/following-sibling::td").inner\_text()

De asemenea, AI-ul a fost folosit la corectarea și îmbunătățirea codului scris de mine, ca ajutor legat de sintaxă și ajutor legat de lucrurile care trebuie testate, dar și ca debugger în caz de erori.

# Bibliografie

[1] The Selenium Browser Automation Project [Internet]. Selenium. Disponibil de la: <https://www.selenium.dev/documentation/>. Accesat la: 11 aprilie 2025.

[2] Selenium. WebDriver [Internet]. Selenium; 2024. Disponibil de la: <https://www.selenium.dev/documentation/webdriver/>. Accesat la: 11 aprilie 2025.

[3] Installation | Playwright Python [Internet]. Playwright.dev; 2016. Disponibil de la: <https://playwright.dev/python/docs/intro>. Accesat la: 12 aprilie 2025.

[4] Playwright vs Selenium vs Cypress: A Detailed Comparison [Internet]. LambdaTest; 2023. Disponibil de la: <https://www.lambdatest.com/blog/playwright-vs-selenium-vs-cypress/>. Accesat la: 14 aprilie 2025.

[5] OpenAI, ChatGPT, https://chatgpt.com/, Data generării: 20 aprilie 2025

[6] Zhan C. Playwright vs Selenium WebDriver: comparație de sintaxă prin exemple [Internet]. Disponibil la: <https://courtneyzhan.medium.com/playwright-vs-selenium-webdriver-syntax-comparison-by-example-ca41e06179d0> Accesat la : 01 mai 2025.

[7] OpenAI, ChatGPT, https://chatgpt.com/, Data generării: 03 mai 2025