**Desenvolupament, proves i automatització amb Selenium i Github Actions**

**Objectius:**

- Desenvolupar una aplicació senzilla amb **Flask** que accepti dades d’un formulari i mostri un resultat.

*Flask és un framework web lleuger per al llenguatge Python, dissenyat per crear aplicacions web de manera senzilla i flexible. És molt popular entre desenvolupadors per la seva facilitat d'ús, modularitat i extensibilitat, i és especialment adequat per a projectes petits o mitjans, així com per a prototips.*

- Escriure proves funcionals amb **Selenium** per comprovar el funcionament correcte de l’aplicació.

*Selenium és un conjunt d’eines per a l’automatització de navegadors web. És àmpliament utilitzat per provar aplicacions web, ja que permet interactuar amb pàgines com ho faria un usuari real: escrivint en camps, clicant botons, navegant entre pàgines, etc.*

- Configurar un workflow de Github **Actions** per executar els tests automàticament cada vegada que es faci un `push` al repositori.

**Part 1: Configuració de l’entorn.**

1. Crea una carpeta per al projecte:

  **mkdir practica\_selenium**

 **cd practica\_selenium**

2. Configura l’entorn virtual i instal·la dependències:

 **python -m venv venv**

 **source venv/bin/activate**

 **pip install flask selenium**

 **pip freeze > requirements.txt**

3. Instal·lació de Google Chrome i ChromeDriver

*ChromeDriver és un controlador (driver) que permet a Selenium interactuar amb el navegador Google Chrome de manera programada.*

- Instal·la Google Chrome:

 **wget https://dl.google.com/linux/direct/google-chrome-stable\_current\_amd64.deb**

 **sudo apt install ./google-chrome-stable\_current\_amd64.deb**

 **google-chrome --version**

 Exemple de sortida: `Google Chrome 131.0.6778.85`

4. Instal·la ChromeDriver:

 - Identifica la versió de Chrome instal·lada:

 **google-chrome --version**

 - Descarrega la versió corresponent de ChromeDriver (en aquest cas, 131):

 **wget https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/131.0.6778.85/linux64/chromedriver-linux64.zip**

 **unzip chromedriver-linux64.zip**

 **sudo mv chromedriver-linux64/chromedriver /usr/local/bin/**

 **sudo chmod +x /usr/local/bin/chromedriver**

 **chromedriver --version**

 Exemple de sortida: `ChromeDriver 131.0.6778.85`

**Part 2: Desenvolupament de l'aplicació**

1. Crea el fitxer `app.py`:

from flask import Flask, request

app = Flask(\_\_name\_\_)

@app.route("/", methods=["GET", "POST"])

def home():

 if request.method == "POST":

 name = request.form.get("name")

 age = request.form.get("age")

 # Si els camps estan buits, torna un missatge per evitar errors

 if not name or not age:

 return "Els camps no poden estar buits!."

 return f"Hola, {name}, tens {age} anys!"

 # Aquesta resposta s'envia si la petició no és POST

 return '''

 <form method="post">

 Nom: <input type="text" name="name"><br>

 Edat: <input type="text" name="age"><br>

 <input type="submit" value="Enviar">

 </form>

 '''

if \_\_name\_\_ == "\_\_main\_\_":

 app.run(debug=True)

 if \_\_name\_\_ == "\_\_main\_\_":

 app.run(debug=True)

**Explicació del codi:**

*- `from flask import Flask, request`:*

 *Importa la classe `Flask` per crear l'aplicació i l'objecte `request` per accedir a les dades enviades a través de formularis o URL.*

*- `app = Flask(\_\_name\_\_)`:*

 *Crea una instància de l'aplicació Flask. L'argument `\_\_name\_\_` és el nom del mòdul actual i ajuda Flask a localitzar recursos i configurar l'aplicació.*

*@app.route("/", methods=["GET", "POST"])*

*- `@app.route`:*

 *És un decorador que defineix una ruta. En aquest cas, la ruta és `/` (la pàgina principal).*

*- `methods=["GET", "POST"]`:*

 *Indica que aquesta ruta accepta dos tipus de peticions HTTP.*

*def home():*

 *Defineix la funció `home`, que gestiona les accions per a la ruta `/`.*

*- `request.method == "POST"`:*

 *Comprova si la petició és una submissió de formulari (POST).*

*- `request.form.get("name")` i `request.form.get("age")`:*

 *Obtén els valors dels camps del formulari amb els noms `"name"` i `"age"`. Si els camps no existeixen, retorna `None`.*

*if not name or not age:*

 *return "Els camps no poden estar buits!."*

 *Si algun dels camps (`name` o `age`) està buit, la funció retorna un missatge d'error perquè no s'envien dades incompletes.*

*return f"Hola, {name}, tens {age} anys!"*

*Si tots dos camps tenen valors vàlids, la funció retorna un missatge personalitzat.*

*return '''*

*<form method="post">*

 *Nom: <input type="text" name="name"><br>*

 *Edat: <input type="text" name="age"><br>*

 *<input type="submit" value="Enviar">*

*</form>*

*'''*

*Si la petició no és POST (és a dir, GET), la funció retorna un formulari HTML.*

*- `if \_\_name\_\_ == "\_\_main\_\_":`*

 *Assegura que el codi només s'executi quan el fitxer es crida directament, no quan es importa com a mòdul.*

*- `app.run(debug=True)`:*

 *Executa el servidor web Flask localment.*

 *El mode `debug=True` activa eines de depuració, com la recàrrega automàtica en modificar el codi i missatges d'error més detallats.*

2. Executa l’aplicació:

 **python app.py**

 Accedeix a `http://127.0.0.1:5000/` des del navegador i comprova que funciona.

**Part 3: Proves funcionals amb Selenium**

1. Crea el fitxer `test\_functional.py`:

import unittest

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.chrome.service import Service

from selenium.webdriver.chrome.options import Options

class TestWebApp(unittest.TestCase):

 @classmethod

 def setUpClass(cls):

 # Configura Chrome amb Selenium

 options = Options()

 # options.add\_argument("--headless") # Executa en mode sense capçalera (desactiva per veure el navegador)

 options.add\_argument("--no-sandbox") # Necessari en entorns CI/CD

 options.add\_argument("--disable-dev-shm-usage") # Evita problemes de memòria compartida

 service = Service("/usr/local/bin/chromedriver") # Ruta del ChromeDriver

 cls.driver = webdriver.Chrome(service=service, options=options)

 cls.driver.get("http://127.0.0.1:5000/") # URL de l'aplicació Flask

 @classmethod

 def tearDownClass(cls):

 # Elimina el tancament automàtic del navegador

 print("El navegador es mantindrà obert. Tanca'l manualment si cal.")

 # cls.driver.quit() # Comenta o elimina aquesta línia

 def test\_form\_submission(self):

 # Troba els camps del formulari i introdueix dades

 name\_field = self.driver.find\_element(By.NAME, "name")

 age\_field = self.driver.find\_element(By.NAME, "age")

 submit\_button = self.driver.find\_element(By.XPATH, '//input[@type="submit"]')

 name\_field.send\_keys("Joan")

 age\_field.send\_keys("30")

 submit\_button.click()

 # Comprova el resultat

 result = self.driver.find\_element(By.TAG\_NAME, "body").text

 self.assertIn("Hola, Joan, tens 30 anys!", result)

if \_\_name\_\_ == "\_\_main\_\_":

 unittest.main()

**Explicació del codi:**

*class TestWebApp(unittest.TestCase):*

*Hereta de `unittest.TestCase`, una classe base que proporciona les eines per escriure i executar tests.*

*@classmethod*

*def setUpClass(cls):*

*Aquest mètode s’executa una sola vegada abans de tots els tests de la classe. És ideal per inicialitzar recursos comuns, com el navegador en aquest cas.*

*Opcions del navegador:*

 *options = Options()*

 *options.add\_argument("--no-sandbox") # Evita problemes de permisos en entorns CI/CD*

 *options.add\_argument("--disable-dev-shm-usage") # Evita problemes de memòria*

 *S’afegeixen paràmetres per assegurar que el navegador funcioni de manera fluida, especialment en entorns virtuals o de contenidors.*

 *# options.add\_argument("--headless")*

 *Si està descomentat, el navegador s’executarà sense interfície gràfica. És útil per a proves automatitzades en servidors.*

 *service = Service("/usr/local/bin/chromedriver")*

 *cls.driver = webdriver.Chrome(service=service, options=options)*

 *Es defineix el camí cap a ChromeDriver i s’inicialitza el navegador.*

 *cls.driver.get("http://127.0.0.1:5000/")*

 *Navega a l’URL de l’aplicació web que s’ha de provar.*

*@classmethod*

*def tearDownClass(cls):*

 *print("El navegador es mantindrà obert. Tanca'l manualment si cal.")*

 *# cls.driver.quit() # Aquesta línia està comentada*

 *name\_field = self.driver.find\_element(By.NAME, "name")*

 *age\_field = self.driver.find\_element(By.NAME, "age")*

 *submit\_button = self.driver.find\_element(By.XPATH, '//input[@type="submit"]')*

 *Utilitza els localitzadors `By.NAME` i `By.XPATH` per trobar els camps d’entrada (`name`, `age`) i el botó d’enviament.*

 *name\_field.send\_keys("Joan")*

 *age\_field.send\_keys("30")*

 *submit\_button.click()*

 *`send\_keys`: Escriu dades en els camps.*

 *`click`: Simula un clic al botó.*

 *result = self.driver.find\_element(By.TAG\_NAME, "body").text*

 *self.assertIn("Hola, Joan, tens 30 anys!", result)*

 *Cerca el text del cos (`<body>`) de la pàgina resultant.*

 *Verifica que conté el missatge esperat amb `assertIn`.*

*if \_\_name\_\_ == "\_\_main\_\_":*

 *unittest.main()*

*Aquesta línia assegura que els tests s’executin només si el fitxer es crida directament.*

*`unittest.main()`: Detecta i executa automàticament tots els mètodes que comencen per `test\_`.*

2. Executa el test funcional localment:

 **python -m unittest test\_functional.py -v**

**Part 4: Automatització amb GitHub Actions**

1. Configura un repositori GitHub:

 - Inicia un repositori nou al teu compte GitHub.

 - Desa el codi creat (`app.py`, `test\_functional.py`, `requirements.txt`) al repositori:

 **git init**

 **git add .**

 **git commit -m "Primera versió de l'aplicació"**

 **git branch -M main**

 **git remote add origin git@github.com:<el\_teu\_nom\_dusuari>/<el\_nom\_del\_repositori>.git**

 **git push -u origin main**

2. Crea el workflow de GitHub Actions:

 - Crea la carpeta `.github/workflows` i afegeix el fitxer `test.yml`:

name: CI

on:

 push:

 branches:

 - main

jobs:

 test:

 runs-on: ubuntu-latest

 steps:

 - name: Checkout code

 uses: actions/checkout@v4

 - name: Debug file structure

 run: ls -la # Llista els fitxers per depurar

 - name: Set up Python

 uses: actions/setup-python@v4

 with:

 python-version: "3.10"

 - name: Install dependencies

 run: |

 python -m pip install --upgrade pip

 pip install -r requirements.txt

 - name: Install Chrome and ChromeDriver

 run: |

 sudo apt-get update

 sudo apt-get install -y google-chrome-stable

 sudo apt-get install -y chromium-chromedriver

 sudo ln -s /usr/lib/chromium-browser/chromedriver /usr/local/bin/chromedriver

 - name: Verify Chrome and ChromeDriver

 run: |

 google-chrome --version || (echo "Chrome no està instal·lat correctament" && exit 1)

 chromedriver --version || (echo "ChromeDriver no està instal·lat correctament" && exit 1)

 - name: Start Flask app

 run: |

 python app.py > flask.log 2>&1 &

 sleep 10 # Dona temps perquè Flask es posi en marxa

 curl -f http://127.0.0.1:5000 || (echo "Flask no està en execució" && cat flask.log && exit 1)

 - name: Run Selenium tests

 run: |

 python -m unittest discover -v

3. Puja el workflow al repositori:

 **git add .github/workflows/test.yml**

 **git commit -m "Afegit workflow CI per executar tests"**

 **git push**

4. Comprova els workflows a GitHub:

 - Ves a la pestanya **Actions** del repositori.

 - Comprova que el workflow s’executa i passa els tests correctament.