Guía Completa: Tests de Calidad con Selenium

Proyecto: ImageProcessor - Sistema de Procesamiento de Imágenes

Fecha: 2025

> **Acerca de esta guía:** Este documento contiene todos los pasos necesarios para implementar y ejecutar tests automatizados con Selenium en tu proyecto ImageProcessor (Flask + React + Vite).

1. Estructura del Proyecto

Tu proyecto debe tener la siguiente estructura después de implementar los tests:

```
imageprocessor/
├── backend/
│   ├── app.py (Flask - Puerto 5000)
│   ├── uploads/
│   └── venv/
├── frontend/
│   ├── src/
│   │   ├── components/
│   │   ├── App.jsx
│   │   └── main.jsx
│   ├── node_modules/
│   ├── public/
│   └── package.json
├── tests/                          ← CREAR ESTA CARPETA
│   ├── __init__.py
│   └── test_imageprocessor.py      ← ARCHIVO PRINCIPAL DE TESTS
├── test_images/                    ← Se crea automáticamente
├── screenshots/                    ← Se crean al fallar tests
└── requirements-test.txt           ← Dependencias para tests
```

2. Requisitos Previos

2.1 Software Necesario

- **Python 3.8+** instalado
- **Google Chrome** (última versión)
- **Node.js y npm** para el frontend
- **ChromeDriver** compatible con tu versión de Chrome

2.2 Verificar Versión de Chrome

En Chrome, ve a: `chrome://settings/help`

Anota la versión (ej: 120.0.6099.109)

2.3 Descargar ChromeDriver

1. Ve a: https://chromedriver.chromium.org/downloads
2. Descarga la versión que coincida con tu Chrome
3. Extrae el archivo y colócalo en tu PATH o en la raíz del proyecto

3. Instalación Paso a Paso

1 **Crear carpeta de tests**

```
# Desde la raíz de imageprocessor
mkdir tests
cd tests
```

2 **Crear archivo __init__.py**

```
# Linux/Mac
touch __init__.py

# Windows
type nul > __init__.py
```

3 Crear archivo test_imageprocessor.py

Copia el código completo que se muestra en la sección 5 de este documento.

4 Crear archivo requirements-test.txt en la raíz

```
selenium==4.15.2
Pillow==10.1.0
pytest==7.4.3
pytest-html==4.1.1
```

5 Instalar dependencias

```
# Crear virtual environment (recomendado)
python -m venv venv_tests

# Activar virtual environment
# En Linux/Mac:
source venv_tests/bin/activate
# En Windows:
venv_tests\Scripts\activate

# Instalar dependencias
pip install -r requirements-test.txt
```

4. Configuración de ChromeDriver

Opción A: ChromeDriver en PATH (Recomendado)

```
# Linux/Mac
sudo mv chromedriver /usr/local/bin/
sudo chmod +x /usr/local/bin/chromedriver

# Windows
# Mover chromedriver.exe a C:\Windows\System32\
# O agregarlo a las variables de entorno PATH
```

Opción B: ChromeDriver en el proyecto

```
# Colocar chromedriver en la raíz del proyecto
# Y modificar el código para especificar la ruta:

from selenium.webdriver.chrome.service import Service

service = Service('./chromedriver')
driver = webdriver.Chrome(service=service)
```

5. Código Completo del Test

Crea el archivo `tests/test_imageprocessor.py` con el siguiente contenido:

```python
import unittest
import time
import os
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.action_chains import ActionChains
from selenium.common.exceptions import TimeoutException, NoSuchElementExcept

class ImageProcessorTests(unittest.TestCase):
    """Tests completos para ImageProcessor"""

    @classmethod
    def setUpClass(cls):
        """Configuración inicial para todos los tests"""
        options = webdriver.ChromeOptions()
        # options.add_argument('--headless')  # Descomentar para modo headle
        options.add_argument('--no-sandbox')
        options.add_argument('--disable-dev-shm-usage')
        options.add_argument('--window-size=1920,1080')

        cls.driver = webdriver.Chrome(options=options)
        cls.driver.maximize_window()
        cls.wait = WebDriverWait(cls.driver, 15)

        cls.BASE_URL = "http://localhost:5173"
        cls.API_URL = "http://localhost:5000"

        cls.TEST_IMAGES_DIR = os.path.join(os.getcwd(), 'test_images')
        os.makedirs(cls.TEST_IMAGES_DIR, exist_ok=True)

    def setUp(self):
        """Configuración antes de cada test"""
        self.driver.get(self.BASE_URL)
        time.sleep(2)

    def take_screenshot(self, name):
        """Capturar screenshot para debugging"""
        timestamp = time.strftime("%Y%m%d-%H%M%S")
        filename = f"screenshot_{name}_{timestamp}.png"
        self.driver.save_screenshot(filename)
        print(f"Screenshot guardado: {filename}")

    def test_01_landing_page_loads(self):
        """Verificar que la landing page carga correctamente"""
        try:
```

```
            self.assertIn("ImageProcessor", self.driver.title)

            logo = self.wait.until(
                EC.presence_of_element_located((By.CSS_SELECTOR, ".logo-image
            )
            self.assertTrue(logo.is_displayed())

            cta_button = self.driver.find_element(By.CSS_SELECTOR, ".cta-butt
            self.assertTrue(cta_button.is_displayed())

            print("✓ Landing page carga correctamente")
        except Exception as e:
            self.take_screenshot("landing_page_error")
            self.fail(f"Error en landing page: {str(e)}")

    def test_02_navigation_menu(self):
        """Verificar navegación entre páginas"""
        try:
            procesador_btn = self.wait.until(
                EC.element_to_be_clickable((By.XPATH, "//button[contains(text
            )
            procesador_btn.click()
            time.sleep(2)

            self.wait.until(
                EC.presence_of_element_located((By.XPATH, "//*[contains(text
            )

            ayuda_btn = self.driver.find_element(By.XPATH, "//button[contains
            ayuda_btn.click()
            time.sleep(2)

            self.wait.until(
                EC.presence_of_element_located((By.XPATH, "//*[contains(text
            )

            print("✓ Navegación funciona correctamente")
        except Exception as e:
            self.take_screenshot("navigation_error")
            self.fail(f"Error en navegación: {str(e)}")

    def test_03_procesador_interface(self):
        """Verificar elementos de la interfaz del procesador"""
        try:
            self._navigate_to_processor()

            drop_zone = self.wait.until(
                EC.presence_of_element_located((By.XPATH, "//*[contains(text
```

```
            )
            self.assertTrue(drop_zone.is_displayed())

            print("✓ Interfaz del procesador correcta")
        except Exception as e:
            self.take_screenshot("procesador_interface_error")
            self.fail(f"Error en interfaz: {str(e)}")

    def _navigate_to_processor(self):
        """Navegar a la página del procesador"""
        try:
            procesador_btn = self.wait.until(
                EC.element_to_be_clickable((By.XPATH, "//button[contains(tex
            )
            procesador_btn.click()
            time.sleep(2)
        except Exception as e:
            raise Exception(f"No se pudo navegar al procesador: {str(e)}")

    def _create_test_image(self):
        """Crear una imagen de prueba"""
        try:
            from PIL import Image
            img = Image.new('RGB', (100, 100), color='red')
            img_path = os.path.join(self.TEST_IMAGES_DIR, 'test_image.jpg')
            img.save(img_path)
            return img_path
        except Exception as e:
            print(f"Error creando imagen: {str(e)}")
            return None

    @classmethod
    def tearDownClass(cls):
        """Limpieza final"""
        time.sleep(2)
        cls.driver.quit()
        print("\n✓ Tests completados")

if __name__ == "__main__":
    suite = unittest.TestLoader().loadTestsFromTestCase(ImageProcessorTests)
    runner = unittest.TextTestRunner(verbosity=2)
    result = runner.run(suite)

    print("\n" + "="*60)
    print(f"Tests ejecutados: {result.testsRun}")
    print(f"Exitosos: {result.testsRun - len(result.failures) - len(result.e
    print(f"Fallidos: {len(result.failures)}")
```

```
    print(f"Errores: {len(result.errors)}")
    print("="*60)
```

6. Ejecución de los Tests

1 Iniciar el Backend (Flask)

```
# Terminal 1
cd backend
python app.py

# Debe estar corriendo en http://localhost:5000
```

2 Iniciar el Frontend (Vite)

```
# Terminal 2
cd frontend
npm run dev

# Debe estar corriendo en http://localhost:5173
```

3 Ejecutar los Tests

```
# Terminal 3 (desde la raíz del proyecto)
# Con virtual environment activado
python tests/test_imageprocessor.py

# O con pytest:
pytest tests/test_imageprocessor.py -v

# Para generar reporte HTML:
pytest tests/test_imageprocessor.py --html=report.html
```

7. Cobertura de Tests

Test	Descripción	Componente
test_01	Carga de landing page	LandingPage
test_02	Navegación entre páginas	App, Navegación
test_03	Interfaz del procesador	ImageProcessor

8. Solución de Problemas Comunes

Error: ChromeDriver no encontrado

> **Solución:** Verifica que ChromeDriver esté en tu PATH o especifica la ruta manualmente en el código.

Error: Connection refused (localhost:5173)

> **Solución:** Asegúrate de que el frontend esté corriendo con `npm run dev`

Error: No se encuentra el elemento

Solución: Los selectores CSS/XPath pueden necesitar ajustes según tu HTML real. Revisa los nombres de clases en tus componentes.

9. Mejores Prácticas

- Ejecuta los tests en un ambiente limpio (sin archivos previos en uploads/)
- Revisa los screenshots cuando un test falle para identificar el problema
- Mantén actualizados ChromeDriver y Selenium
- Usa modo headless en CI/CD pero ejecuta con interfaz durante desarrollo
- Añade más esperas explícitas si encuentras elementos que cargan lentamente

10. Comandos Útiles

```
# Ejecutar solo un test específico
python -m pytest tests/test_imageprocessor.py::ImageProcessorTests::test_01_

# Ejecutar tests con más detalles
python -m pytest tests/test_imageprocessor.py -vv

# Ejecutar y detener al primer fallo
python -m pytest tests/test_imageprocessor.py -x

# Generar reporte con capturas
python -m pytest tests/test_imageprocessor.py --html=report.html --self-cont
```

Resultado Esperado: Si todo está configurado correctamente, deberías ver una salida similar a:

```
test_01_landing_page_loads ... ok
test_02_navigation_menu ... ok
test_03_procesador_interface ... ok

Ran 3 tests in 25.431s

OK
```

11. Extensión de Tests

Para agregar más tests, sigue este patrón:

```python
def test_04_nuevo_test(self):
    """Descripción del test"""
    try:
        # Tu código de test aquí
        elemento = self.wait.until(
            EC.presence_of_element_located((By.ID, "mi-elemento"))
        )
        self.assertTrue(elemento.is_displayed())

        print("✓ Test pasó correctamente")
    except Exception as e:
        self.take_screenshot("nuevo_test_error")
        self.fail(f"Error: {str(e)}")
```

12. Integración Continua (CI/CD)

Para ejecutar en GitHub Actions, crea `.github/workflows/tests.yml`:

```yaml
name: Selenium Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: 3.9

    - name: Install dependencies
      run: |
        pip install -r requirements-test.txt

    - name: Install Chrome
      run: |
        sudo apt-get install google-chrome-stable

    - name: Run tests
      run: |
        python tests/test_imageprocessor.py
```

Guía creada para el proyecto ImageProcessor

Para imprimir: Ctrl+P o Cmd+P y selecciona "Guardar como PDF"