

Guía de ejercicios

Evaluación de Sistema QA  
Semana 6



**Nombre:** Herramientas para QA basados en IA

### Desarrollo de la guía

#### Instrucciones

Estimado(a) estudiante:

El objetivo de esta guía es enseñar cómo realizar pruebas automatizadas en una aplicación web sencilla utilizando Selenium. Además, se integrará un componente de inteligencia artificial (IA) mediante el uso del algoritmo Random Forest, para validar los resultados de las operaciones. Este documento está diseñado para comprender tanto los conceptos básicos de la automatización de pruebas como la aplicación práctica de modelos de IA en un entorno de prueba.

Existen otras herramientas que también utilizan la inteligencia artificial para realizar pruebas de software, aunque muchas de ellas son de carácter privado y sus versiones gratuitas presentan limitaciones. Un ejemplo destacado es MABL, una plataforma avanzada que aprovecha la IA para automatizar pruebas funcionales y de regresión, pero su uso completo requiere suscripción debido a las restricciones en las funcionalidades disponibles en su versión gratuita.

Es fundamental destacar que este recurso no constituye una evaluación, sino más bien un material adicional destinado a brindarte la oportunidad de practicar lo estudiado a nivel teórico y reforzar tus habilidades.

¡Anímate a abordar estos desafíos y descubre cómo tu habilidad para automatizar pruebas puede crecer con el ejercicio analizado, comprendido y completado!

#### Pasos generales para utilizar Selenium con IA

##### Definición de casos de prueba

- Identificar los casos de uso críticos y definir los escenarios de prueba específicos que cubran todas las funcionalidades clave del sistema de pedidos en línea.
- Crear casos de prueba tanto para situaciones normales como para condiciones extremas.

##### Configuración del entorno de pruebas

- Configurar el entorno de desarrollo y pruebas, incluyendo la instalación de Selenium WebDriver, la configuración de navegadores y la integración con herramientas de CI/CD (Integración Continua/Despliegue Continuo).
- Establecer el entorno para pruebas de carga y rendimiento, simulando múltiples usuarios concurrentes.

### Desarrollo de scripts de pruebas automatizadas

- Escribir scripts de Selenium para automatizar las pruebas funcionales y no funcionales del sistema.
- Integrar modelos de IA en los scripts de prueba para evaluar la eficacia en la detección de fraudes y en el análisis de datos.

### Ejecución de pruebas y monitoreo

- Ejecutar los scripts de prueba de manera regular y bajo diferentes condiciones de carga.
- Utilizar herramientas de monitoreo para rastrear el rendimiento del sistema y la detección de anomalías durante las pruebas.

**Análisis de resultados y optimización:** analizar los resultados de las pruebas para identificar áreas de mejora y optimización.

**Mantenimiento y actualización continua:** mantener los scripts de prueba actualizados con cualquier cambio en el sistema. Realizar pruebas periódicas para garantizar que el sistema siga siendo robusto, seguro y eficiente a lo largo del tiempo.

### Ejercicio

La base de esta guía es una aplicación web simple desarrollada con Flask, que permite realizar operaciones matemáticas básicas: suma, resta, multiplicación y división. A continuación, se presentan los archivos de la aplicación (es la misma calculadora utilizada previamente).

calculadora.py

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/calculate', methods=['POST'])
def calculate():
    num1 = float(request.form['num1'])
    num2 = float(request.form['num2'])
    operator = request.form['operator']

    if operator == 'sumar':
        result = num1 + num2
    elif operator == 'restar':
        result = num1 - num2
    elif operator == 'multiplicar':
        result = num1 * num2
```

```
elif operator == 'dividir':
    if num2 != 0:
        result = num1 / num2
    else:
        result = 'Error: Division por cero'
else:
    result = 'Error: Operador inválido'

return render_template('index.html', result=result)

if __name__ == '__main__':
    app.run(debug=True)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Calculadora</title>
</head>
<body>
    <h2>Calculadora sencilla</h2>
    <form action="/calculate" method="post">
        <input type="text" name="num1" placeholder="Ingrese primer número"
required><br><br>
        <input type="text" name="num2" placeholder="Ingrese segundo número"
required><br><br>
        <select name="operator">
            <option value="sumar">Suma</option>
            <option value="restar">Resta</option>
            <option value="multiplicar">Multiplica</option>
            <option value="dividir">Divide</option>
        </select><br><br>
        <input type="submit" value="Calcular">
    </form>
    {% if result %}
    <h2>Resultado:</h2>
    <p>{{ result }}</p>
    {% endif %}
</body>
</html>
```

## Explicación del programa

El programa es una aplicación web sencilla que permite realizar operaciones matemáticas básicas. Utilizando Flask, un microframework para Python. A continuación se explica cada parte del código.

- **Importación de librerías:** se importan las funciones necesarias de Flask para crear la aplicación web, renderizar plantillas HTML y manejar solicitudes HTTP.
- **Inicialización de la aplicación Flask:** se crea una instancia de Flask.
- **Ruta para la página principal:** define la ruta principal ("/") y la función index que renderiza la plantilla index.html. Ruta para el cálculo: Define la ruta "/calculate" que maneja las solicitudes POST cuando se envía el formulario. La función calculate extrae los números y el operador del formulario, realiza la operación y devuelve el resultado.
- **Ejecución de la aplicación:** inicia el servidor de flask en modo depuración.

**Ruta para el cálculo:** este programa utiliza Selenium para automatizar la interacción con la aplicación web y un modelo de Random Forest para validar los resultados de las operaciones.

```
@app.route('/calculate', methods=['POST'])
def calculate():
    num1 = float(request.form['num1'])
    num2 = float(request.form['num2'])
    operator = request.form['operator']

    if operator == 'sumar':
        result = num1 + num2
    elif operator == 'restar':
        result = num1 - num2
    elif operator == 'multiplicar':
        result = num1 * num2
    elif operator == 'dividir':
        if num2 != 0:
            result = num1 / num2
        else:
            result = 'Error: Division por cero'
    else:
        result = 'Error: Operador inválido'

    return render_template('index.html', result=result)
```

- . @app.route('/calculate', methods=['POST']): define una ruta para manejar las solicitudes POST a la URL /calculate. Esto sucede cuando el usuario envía el formulario desde la página principal.
- . calculate(): función que maneja las solicitudes POST a /calculate.

### Dentro de calculate()

#### 1. Extracción de datos del formulario:

```
num1 = float(request.form['num1'])
num2 = float(request.form['num2'])
operator = request.form['operator']
```

- . Extrae los valores num1, num2, y operator del formulario enviado.
- . Convierte num1 y num2 a float para poder realizar operaciones matemáticas.

#### 2. Realización de la Operación:

```
if operator == 'sumar':
    result = num1 + num2
elif operator == 'restar':
    result = num1 - num2
elif operator == 'multiplicar':
    result = num1 * num2
elif operator == 'dividir':
    if num2 != 0:
        result = num1 / num2
    else:
        result = 'Error: Division por cero'
else:
    result = 'Error: Operador inválido'
```

- . Dependiendo del operador seleccionado (sumar, restar, multiplicar, dividir), realiza la operación correspondiente.
- . Maneja el caso especial de división por cero.
- . Si el operador es inválido, devuelve un mensaje de error.

#### 3. Renderización del resultado:

```
return render_template('index.html', result=result)
```

- . Renderiza de nuevo la plantilla index.html, pero esta vez incluye el resultado de la operación para ser mostrado en la página.

## Ejecución de la Aplicación

```
if __name__ == '__main__':  
    app.run(debug=True)
```

. if \_\_name\_\_ == '\_\_main\_\_': verifica si el script se está ejecutando directamente (no importado como módulo).

. app.run(debug=True): inicia el servidor de desarrollo de Flask con el modo de depuración activado. El modo de depuración reinicia automáticamente el servidor cuando se detectan cambios en el código y proporciona una página de depuración en caso de errores.

## Resumen

El programa permite a los usuarios realizar operaciones matemáticas básicas a través de una interfaz web simple. Los pasos son los siguientes:

1. El usuario accede a la página principal y ve un formulario.
2. El usuario ingresa dos números y selecciona una operación.
3. Al enviar el formulario, se envía una solicitud POST a la ruta /calculate.
4. El servidor realiza la operación y devuelve el resultado a la página principal, donde se muestra al usuario

## Programa de prueba

A continuación, se muestra el código del programa realizado en Python, en donde se utiliza Selenium y el algoritmo de IA Random Forest.

```
from selenium import webdriver  
from selenium.webdriver.common.keys import Keys  
import time  
import numpy as np  
from sklearn.ensemble import RandomForestRegressor  
  
def iniciar_navegador():  
    driver = webdriver.Chrome()  
    return driver  
  
def abrir_pagina(driver, url):  
    driver.get(url)  
  
def realizar_operacion(driver, num1, num2, operador):  
    num1_input = driver.find_element("name", "num1")  
    num2_input = driver.find_element("name", "num2")  
    operator_select = driver.find_element("name", "operator")  
  
    submit_button = driver.find_element("xpath", "//*[@type='submit']")  
  
    num1_input.clear()  
    num2_input.clear()
```



```
num1_input.send_keys(str(num1))
num2_input.send_keys(str(num2))

if operador == 1:
    operator_select.send_keys(Keys.DOWN)
elif operador == 2:
    operator_select.send_keys(Keys.DOWN)
    operator_select.send_keys(Keys.DOWN)
elif operador == 3:
    operator_select.send_keys(Keys.DOWN)
    operator_select.send_keys(Keys.DOWN)
    operator_select.send_keys(Keys.DOWN)

submit_button.click()

time.sleep(2) # Esperar a que se muestre el resultado

def obtener_resultado(driver):
    result_element = driver.find_element("xpath", "//p")
    return result_element.text

def cerrar_navegador(driver):
    driver.quit()

def probar_operacion(driver, num1, num2, operador, resultado_esperado, model):

    realizar_operacion(driver, num1, num2, operador)
    resultado_obtenido = obtener_resultado(driver)

    if operador == 0:
        accion = "más"
    elif operador == 1:
        accion = "menos"
    elif operador == 2:
        accion = "por"
    elif operador == 3:
        accion = "entre"

    print(f"Operación: {num1} {accion} {num2}")
    print("Resultado esperado:", resultado_esperado)
    print("Resultado obtenido:", resultado_obtenido)

    # Validar con el modelo de Random Forest
    X_test = np.array([[num1, num2, operador]])
    resultado_predicho = model.predict(X_test)[0]
    print("Resultado predicho por el modelo:", resultado_predicho)
```



```
resultado_predicho = round(resultado_predicho)

print("Resultado predicho por el modelo con redondeo:", resultado_predicho)

if str(resultado_esperado) == resultado_obtenido and
abs(float(resultado_obtenido) - resultado_predicho) < 0.01:
    print("¡Prueba exitosa!")
else:
    print("¡Prueba fallida!")

def entrenar_modelo():
    # Datos de entrenamiento ampliados: [num1, num2, operador], resultado
    X_train = []
    y_train = []

    # Agregar más datos de entrenamiento
    for num1 in range(1, 101):
        for num2 in range(1, 101):
            X_train.append([num1, num2, 0]) # Suma
            y_train.append(num1 + num2)
            X_train.append([num1, num2, 1]) # Resta
            y_train.append(num1 - num2)
            X_train.append([num1, num2, 2]) # Multiplicación
            y_train.append(num1 * num2)
            if num2 != 0: # Evitar división por cero
                X_train.append([num1, num2, 3]) # División
                y_train.append(num1 / num2)

    X_train = np.array(X_train)
    y_train = np.array(y_train)

    model = RandomForestRegressor(n_estimators=100)
    model.fit(X_train, y_train)
    return model

def ejecutar_pruebas():
    driver = iniciar_navegador()
    abrir_pagina(driver, "http://localhost:5000")

    model = entrenar_modelo()

    # Pruebas de adición
    probar_operacion(driver, 12, 5, 0, '17.0', model)
    probar_operacion(driver, 200, 30, 0, '230.0', model)
```

```
# Pruebas de sustracción
probar_operacion(driver, 12, 5, 1, '7.0', model)
probar_operacion(driver, 40, 30, 1, '10.0', model)

# Pruebas de multiplicación
probar_operacion(driver, 5, 6, 2, '30.0', model)
probar_operacion(driver, 7, 7, 2, '49.0', model)

# Pruebas de división
probar_operacion(driver, 100, 2, 3, '50.0', model)
probar_operacion(driver, 40, 2, 3, '20.0', model)

cerrar_navegador(driver)

# Ejecutar las pruebas
ejecutar_pruebas()
```

### Explicación del programa

En este programa, se utiliza el modelo predictivo Random Forest para predecir resultados de operaciones matemáticas (suma, resta, multiplicación y división) basadas en dos números de entrada y un operador. Visualiza a continuación una explicación detallada de cómo se implementa este modelo en el programa.

### Importación de librerías

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time
import numpy as np
from sklearn.ensemble import RandomForestRegressor
```

1. Selenium: se usa para automatizar la interacción con el navegador.
2. Numpy: se usa para manejar arrays numéricos.
3. RandomForestRegressor: es el modelo de aprendizaje automático utilizado para las predicciones.

### Funciones para manejo del navegador mediante Selenium

1. iniciar\_navegador: inicia una instancia del navegador Chrome.
2. abrir\_pagina: abre una página web en el navegador.
3. realizar\_operacion: rellena los campos de entrada de la operación matemática en la página web y envía el formulario.
4. obtener\_resultado: obtiene el resultado de la operación de la página web.
5. cerrar\_navegador: cierra el navegador.

### Función para probar la operación

```
def probar_operacion(driver, num1, num2, operador, resultado_esperado, model):

    realizar_operacion(driver, num1, num2, operador)
    resultado_obtenido = obtener_resultado(driver)

    if operador == 0:
        accion = "más"
    elif operador == 1:
        accion = "menos"
    elif operador == 2:
        accion = "por"
    elif operador == 3:
        accion = "entre"

    print(f"Operación: {num1} {accion} {num2}")
    print("Resultado esperado:", resultado_esperado)
    print("Resultado obtenido:", resultado_obtenido)

    # Validar con el modelo de Random Forest
    X_test = np.array([[num1, num2, operador]])
    resultado_predicho = model.predict(X_test)[0]
    print("Resultado predicho por el modelo:", resultado_predicho)

    resultado_predicho = round(resultado_predicho)

    print("Resultado predicho por el modelo con redondeo:", resultado_predicho)

    if str(resultado_esperado) == resultado_obtenido and \
    abs(float(resultado_obtenido) - resultado_predicho) < 0.01:
        print("¡Prueba exitosa!")
    else:
        print("¡Prueba fallida!")
```

#### Esta función realiza los siguientes pasos:

1. Realiza la operación en la página web.
2. Obtiene el resultado de la operación desde la página web.
3. Usa el modelo Random Forest para predecir el resultado basado en num1, num2, y operador.
4. Compara el resultado esperado y el obtenido desde la página web con el resultado predicho por el modelo.
5. Imprime si la prueba fue exitosa o fallida.

## Función para entrenar el modelo

```
def entrenar_modelo():
    # Datos de entrenamiento ampliados: [num1, num2, operador], resultado
    X_train = []
    y_train = []

    # Agregar más datos de entrenamiento
    for num1 in range(1, 101):
        for num2 in range(1, 101):
            X_train.append([num1, num2, 0]) # Suma
            y_train.append(num1 + num2)
            X_train.append([num1, num2, 1]) # Resta
            y_train.append(num1 - num2)
            X_train.append([num1, num2, 2]) # Multiplicación
            y_train.append(num1 * num2)
            if num2 != 0: # Evitar división por cero
                X_train.append([num1, num2, 3]) # División
                y_train.append(num1 / num2)

    X_train = np.array(X_train)
    y_train = np.array(y_train)

    model = RandomForestRegressor(n_estimators=100)
    model.fit(X_train, y_train)
    return model
```

### a. Preparación de datos de entrenamiento

- . X\_train y y\_train: dos listas que almacenan las características de entrada (números y operación) y los resultados respectivos.
- . Bucle for: genera datos de entrenamiento para todas las combinaciones posibles de números del 1 al 100 y para cada tipo de operación (suma, resta, multiplicación y división).
  - . Suma: X\_train.append([num1, num2, 0]) y y\_train.append(num1 + num2)
  - . Resta: X\_train.append([num1, num2, 1]) y y\_train.append(num1 - num2)
  - . Multiplicación: X\_train.append([num1, num2, 2]) y y\_train.append(num1 \* num2)
  - . División: X\_train.append([num1, num2, 3]) y y\_train.append(num1 / num2) (con la condición de que num2 no sea 0)

### b. Creación y entrenamiento del modelo

- . RandomForestRegressor: se instancia un modelo de regresión de bosque aleatorio con n\_estimators=100 (100 árboles de decisión).
- . model.fit(X\_train, y\_train): el modelo se entrena con los datos de entrada y los resultados correspondientes.

### Función para ejecutar las pruebas

```
def ejecutar_pruebas():  
    driver = iniciar_navegador()  
    abrir_pagina(driver, "http://localhost:5000")  
  
    model = entrenar_modelo()  
  
    # Pruebas de adición  
    probar_operacion(driver, 12, 5, 0, '17.0', model)  
    probar_operacion(driver, 200, 30, 0, '230.0', model)  
  
    # Pruebas de sustracción  
    probar_operacion(driver, 12, 5, 1, '7.0', model)  
    probar_operacion(driver, 40, 30, 1, '10.0', model)  
  
    # Pruebas de multiplicación  
    probar_operacion(driver, 5, 6, 2, '30.0', model)  
    probar_operacion(driver, 7, 7, 2, '49.0', model)  
  
    # Pruebas de división  
    probar_operacion(driver, 100, 2, 3, '50.0', model)  
    probar_operacion(driver, 40, 2, 3, '20.0', model)  
  
    cerrar_navegador(driver)
```

1. Inicio y apertura de la página: se inicia el navegador y se abre la página web.
2. Entrenamiento del modelo: se entrena el modelo con la función `entrenar_modelo`.
3. Ejecución de pruebas:

#### a. Realización de la operación y obtención del Resultado

- La función `realizar_operacion` envía los números y el operador a la página web y hace clic en el botón de envío.
- La función `obtener_resultado` extrae el resultado mostrado en la página web.

#### b. Predicción con el modelo

- `X_test`: un arreglo numpy que contiene los números y el operador de la operación actual.
- `model.predict(X_test)`: utiliza el modelo entrenado para predecir el resultado de la operación.
- Redondeo: el resultado predicho se redondea al número entero más cercano para comparar con el resultado obtenido de la página web.

### c. Validación de resultados

- Compara el resultado esperado y el obtenido de la página web.
- Compara el resultado obtenido de la página web con el resultado predicho por el modelo.
- Imprime si la prueba es exitosa o fallida basada en estas comparaciones.

**4. Cierre del navegador:** se cierra el navegador después de ejecutar las pruebas.

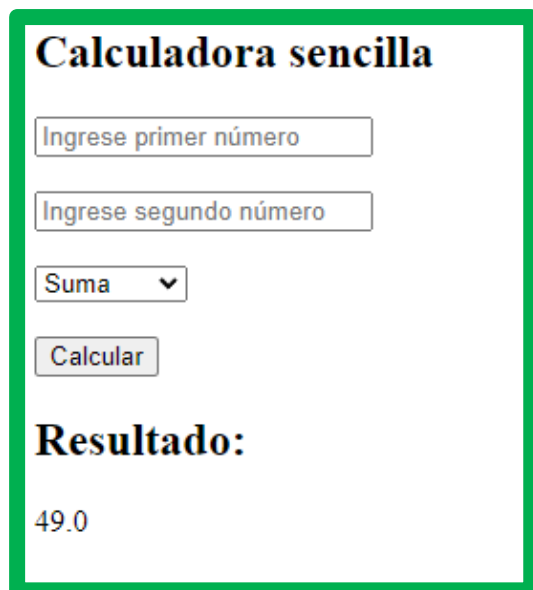
Resumen: el programa utiliza Selenium para automatizar la interacción con una página web que realiza operaciones matemáticas. Se entrena un modelo de Random Forest con un conjunto ampliado de datos de entrenamiento para predecir los resultados de estas operaciones. Las pruebas comprueban si los resultados predichos por el modelo coinciden con los resultados obtenidos de la página web, validando así la precisión del modelo.

### Resultado de la ejecución de la prueba

Se debe ejecutar la aplicación calculadora.py, y luego ejecutar la aplicación de prueba.

Luego, automáticamente Selenium abrirá la ventana de la aplicación y se podrá observar cómo los campos con la información y la operación deseada se llenan y seleccionan automáticamente, de acuerdo con las especificaciones de las pruebas.

A continuación, se muestra un ejemplo de la operación automatizada en la página renderizada de la aplicación calculadora.py:



**Calculadora sencilla**

▼

**Resultado:**

49.0

El resultado de las pruebas se obtiene en la consola de Visual Studio Code.

```
DevTools listening on ws://127.0.0.1:56802/devtools/browser/77434215-81c1-43ca-9baa-24aedef2bb530
Operación: 12 más 5
Resultado esperado: 17.0
Resultado obtenido: 17.0
Resultado predicho por el modelo: 16.95
Resultado predicho por el modelo con redondeo: 17
¡Prueba exitosa!
Operación: 200 más 30
Resultado esperado: 230.0
Resultado obtenido: 230.0
Resultado predicho por el modelo: 129.69
Resultado predicho por el modelo con redondeo: 130
¡Prueba fallida!
Created TensorFlow Lite XNNPACK delegate for CPU.
Operación: 12 menos 5
Resultado esperado: 7.0
Resultado obtenido: 7.0
Resultado predicho por el modelo: 6.91
Resultado predicho por el modelo con redondeo: 7
¡Prueba exitosa!
Operación: 40 menos 30
Resultado esperado: 10.0
Resultado obtenido: 10.0
Resultado predicho por el modelo: 9.97
Resultado predicho por el modelo con redondeo: 10
¡Prueba exitosa!
Operación: 5 por 6
Resultado esperado: 30.0
Resultado obtenido: 30.0
Resultado predicho por el modelo: 29.74
Resultado predicho por el modelo con redondeo: 30
¡Prueba exitosa!
Operación: 7 por 7
Resultado esperado: 49.0
Resultado obtenido: 49.0
Resultado predicho por el modelo: 49.07
Resultado predicho por el modelo con redondeo: 49
¡Prueba exitosa!
Operación: 100 entre 2
Resultado esperado: 50.0
Resultado obtenido: 50.0
Resultado predicho por el modelo: 49.695
Resultado predicho por el modelo con redondeo: 50
¡Prueba exitosa!
```



```
Operación: 40 entre 2  
Resultado esperado: 20.0  
Resultado obtenido: 20.0  
Resultado predicho por el modelo: 19.89  
Resultado predicho por el modelo con redondeo: 20  
¡Prueba exitosa!
```

Random Forest es una técnica poderosa y flexible que mejora la precisión y la robustez de las predicciones mediante la combinación de múltiples árboles de decisión independientes. Utiliza la técnica de bagging y la selección aleatoria de características para construir árboles menos correlacionados y reduce el riesgo de sobreajuste. Aunque puede ser computacionalmente costoso y menos interpretable, su capacidad para manejar grandes conjuntos de datos y alta dimensionalidad lo convierte en una opción popular en muchos problemas de aprendizaje supervisado.