



UNIVERSIDAD
NACIONAL
DE COLOMBIA

Universidad Nacional de Colombia - sede Bogotá
Facultad de Ingeniería
Departamento de Sistemas e Industrial
Curso: Ingeniería de Software 1 (2016701)

Taller del módulo de requerimientos

Alejandro Alejandro Arguello Munoz

Steven David Alfonso Galindo

Daniel Santiago Delgado Pinilla

Juan Luis Vergara Novoa

Testing

Introducción breve: Descripción general de la aplicación.

Resumen de los tests, Tabla o listado con:


- Nombre del integrante
- Tipo de prueba realizada
- Descripción breve del componente probado
- Herramienta o framework usado
- Screenshot del código del test
- Resultado de la ejecución: Screenshot del test ejecutado y su salida.

Lecciones aprendidas y dificultades: Reflexión grupal sobre la experiencia de testear.

Descripción de la aplicación.

EcoScan es una aplicación móvil diseñada para optimizar y cambiar un poco la forma en que gestionamos el reciclaje y el manejo de residuos, con el objetivo general de educar sobre el tema y promover buenas prácticas de reciclaje. Mediante el uso de inteligencia artificial y visión por computadora, la app permite a los usuarios capturar imágenes de objetos y recibir una clasificación automática que los identifica y caracteriza, facilitando así su disposición correcta. Además, EcoScan ofrece funcionalidades complementarias como el registro y análisis del historial de escaneos, la personalización de perfiles y el establecimiento de objetivos ambientales, lo que impulsa hábitos sostenibles y educa sobre buenas prácticas de reciclaje. En última instancia, la aplicación busca motivar y empoderar a sus usuarios para contribuir activamente con el medio ambiente.

Resumen de los tests.

Nombre	Daniel Santiago Delgado Pinilla
Prueba	Prueba de integración
Descripción	Prueba de conexión entre el frontend (React Native)
Herramienta	Jest
Screenshot	 <pre>frontend > EcoScan > _test_ > JS RegisterUser.test.js > ... 1 import React from 'react'; 2 import { render, fireEvent, waitFor } from '@testing-library/react-native'; 3 import RegistroScreen from '../app/register'; 4 5 6 7 global.fetch = jest.fn(() => 8 Promise.resolve({ 9 ok: true, 10 json: () => Promise.resolve({ message: 'Registro exitoso' }), 11 }); 12); 13 14 test('Register a new user', async () => { 15 const navigation = { goBack: jest.fn() }; 16 const { getByPlaceholderText, getByText } = render(<RegistroScreen navigation={navigation} />); 17 18 fireEvent.changeText(getByPlaceholderText('Nombre'), 'Usuario Test'); 19 fireEvent.changeText(getByPlaceholderText('Correo electrónico'), 'test@example.com'); 20 fireEvent.changeText(getByPlaceholderText('Contraseña'), 'password123'); 21 fireEvent.press(getByText('Registrarse')); 22 23 await waitFor(() => expect(fetch).toHaveBeenCalledWith(24 'http://127.0.0.1:8000/register/', expect.objectContaining({ method: 'POST' }) 25)); 26 27 }); 28</pre>

Resultado

Watch Usage

- > Press **f** to run only failed tests.
- > Press **o** to only run tests related to changed files.
- > Press **p** to filter by a filename regex pattern.
- > Press **t** to filter by a test name regex pattern.
- > Press **q** to quit watch mode.
- > Press **Enter** to trigger a test run.

console.log

CSRF token recibido

at log (app/register.js:15:28)

PASS __test__/RegisterUser.test.js

✓ Register a new user (316 ms)

Test Suites: **1 passed**, 1 total

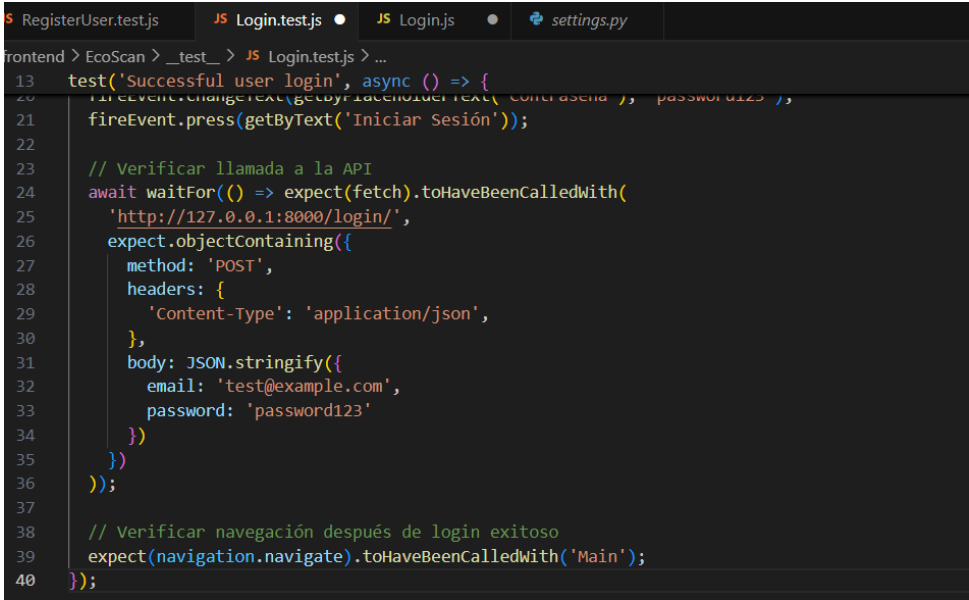
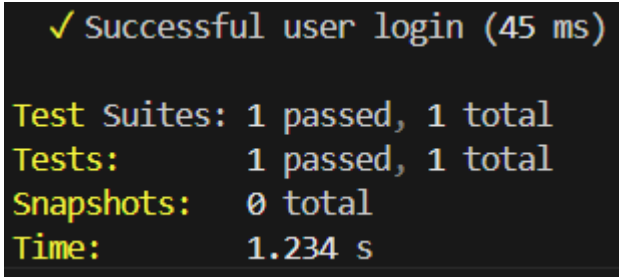
Tests: **1 passed**, 1 total

Snapshots: 0 total

Time: 1.638 s, estimated 2 s

Ran all test suites.

Watch Usage: Press **w** to show more.

Nombre	Alejandro Argüello Muñoz
Prueba	Prueba de Login
Descripción	Prueba de logueo de un usuario en la aplicación
Herramienta	Jest
Screenshot	 <pre> 5 RegisterUser.test.js JS Login.test.js JS Login.js settings.py frontend > EcoScan > _test_ > JS Login.test.js > ... 13 test('Successful user login', async () => { 14 fireEvent.changeText(getByLabelText('contraseña'), 'password123'); 21 fireEvent.press(getByText('Iniciar Sesión')); 22 23 // Verificar llamada a la API 24 await waitFor(() => expect(fetch).toHaveBeenCalled()); 25 expect(fetch).toHaveBeenCalledWith(26 'http://127.0.0.1:8000/login/', 27 expect.objectContaining({ 28 method: 'POST', 29 headers: { 30 'Content-Type': 'application/json', 31 }, 32 body: JSON.stringify({ 33 email: 'test@example.com', 34 password: 'password123' 35 }) 36 }); 37 38 // Verificar navegación después de login exitoso 39 expect(navigation.navigate).toHaveBeenCalledWith('Main'); 40 }); </pre>
Resultado	 <pre> ✓ Successful user login (45 ms) Test Suites: 1 passed, 1 total Tests: 1 passed, 1 total Snapshots: 0 total Time: 1.234 s </pre>

Nombre	Steven David Alfonso Galindo
Prueba	Prueba de Escáner
Descripción	Se valida la correcta respuesta del endpoint que procesa la imagen y retorna la etiqueta y recomendación del contenedor.
Herramienta	Django REST Framework (APITestCase)
Screenshot	 <pre> Repositorio-grupal--Ingenier-a-de-Software-1---2024-2-Grupo-11 > Proyecto > scanner > tests > test_endpoints.py > PredictEndpointTest > t 1 from django.urls import reverse 2 from django.core.files.uploadedfile import SimpleUploadedFile 3 from rest_framework.test import APITestCase 4 from rest_framework import status 5 from PIL import Image 6 import io 7 8 class PredictEndpointTest(APITestCase): 9 def test_predict_endpoint(self): 10 url = reverse('predict-image') 11 12 image = Image.new('RGB', (224, 224), color=(255, 0, 0)) 13 image_io = io.BytesIO() 14 image.save(image_io, format='JPEG') 15 image_io.seek(0) 16 17 uploaded_image = SimpleUploadedFile(18 "test.jpg", 19 image_io.read(), 20 content_type="image/jpeg" 21) 22 23 data = { 24 'image': uploaded_image 25 } 26 27 response = self.client.post(url, data, format='multipart') 28 29 self.assertEqual(response.status_code, status.HTTP_200_OK) 30 31 self.assertIn('predicted_class', response.data) 32 self.assertIn('recommended_container', response.data) 33 </pre>
Resultado	 <pre> 1/1 ██████████ 1s 1s/step . ----- Ran 1 test in 1.319s OK Destroying test database for alias 'default'... </pre>

Nombre	Juan Luis Vergara Novoa
Prueba	Prueba de Personalizacion de Usuario
Descripción	Prueba de conexión entre el frontend, que valida que se permite actualizar datos de los usuarios.
Herramienta	Jest
Screenshot	 <pre> frontend > EcoScan > __test__ > JS PersonalizeUser.test.js > ... 1 // __test__/JS_PersonalizeUser.test.js 2 import React from 'react'; 3 import { render, fireEvent, waitFor } from '@testing-library/react-native'; 4 import PersonalizeUserScreen from '../app/PersonalizeUser'; 5 6 // Mock de fetch 7 global.fetch = jest.fn(() => 8 Promise.resolve({ 9 ok: true, 10 json: () => Promise.resolve({ message: 'Perfil actualizado' }), 11 }) 12); 13 14 describe('PersonalizeUserScreen', () => { 15 test('Actualiza el perfil correctamente', async () => { 16 const navigation = { goBack: jest.fn() }; 17 const { getByPlaceholderText, getByText } = render(18 <PersonalizeUserScreen navigation={navigation} /> 19); 20 21 // Simular cambios en los inputs 22 fireEvent.changeText(getByPlaceholderText('Nombre'), 'Nuevo Nombre'); 23 fireEvent.changeText(getByPlaceholderText('Preferencias'), 'Tema oscuro'); 24 25 // Simular clic en el botón 26 fireEvent.press(getByText('Guardar Cambios')); 27 28 // Verificar llamada a la API 29 await waitFor(() => { 30 expect(fetch).toHaveBeenCalledWith(31 'http://127.0.0.1:8080/update-profile/', 32 expect.objectContaining({ 33 method: 'POST', 34 body: JSON.stringify({ 35 nombre: 'Nuevo Nombre', 36 preferencias: 'Tema oscuro' 37 }) 38 }) 39); 40 }); 41 }); 42 }); </pre>
Resultado	 <pre> (venv) PS C:\Users\user\OneDrive\Desktop\Proyecto\frontend\EcoScan__test__> npm test PersonalizeUser.test.js console.log CSRF token recibido at log (app/register.js:15:28) console.log CSRF token recibido at log (app/register.js:15:28) PASS __test__/PersonalizeUser.test.js ✓ Register a new user (214 ms) Test Suites: 1 passed, 1 total Tests: 1 passed, 1 total Snapshots: 0 total Time: 1.081 s, estimated 3 s Ran all test suites matching /PersonalizeUser.test.js/i. Watch Usage: Press w to show more. </pre>

Lecciones aprendidas:

- Nos dimos cuenta de que hacer las pruebas desde el inicio facilita las cosas pues es más fácil darse cuenta de errores en las fases tempranas de desarrollo. Por ejemplo, en el registro, ocurrió un error del que no me había dado cuenta y había seguido programando. Ya después cuando se añadan más cosas va a ser más complicado arreglar los problemas.
- Hacer las pruebas desde el inicio facilita las cosas pues ya que es más fácil darse cuenta de los errores al inicio de desarrollo. Sin embargo, configurar el test fue una tarea complicada.
- Aprendimos que era importante seguir cierta convención de nomenclatura para que Django detecte de manera automática los test.