MANUAL DE INSTALACION Y EJECUCION

Prueba técnica Double V Partners

Andres Velez
bavelch.0328@gmail.com

Contenido

Framework para utilizar

Playwright

Justificación:
Se elige Playwright porque es más escalable y tiene mayor soporte completo de navegadores, lo que es clave en pruebas modernas. Además:

- Permite ejecuciones paralelas de forma nativa sin depender de un servicio externo.
- Está pensado para automatización de pruebas End-to-End complejas, ideal para tu flujo de compra con modales, formularios y múltiples pasos.
- Microsoft lo respalda, garantizando mantenimiento activo y evolución constante.

Instalación del framework y el IDE a utilizar

El proceso de implementación y ejecución se desarrolla en los siguientes pasos:

Requisitos previos

- Tener instalado **Node.js** (versión 18 o superior).
- Contar con un entorno de desarrollo como **Visual Studio Code**.
- Acceso a línea de comandos (CLI).

Clonar el proyecto

Clonar el repositorio del proyecto desde GitHub o GitLab:

git clone https://github.com/BorisMatrix1200/Entrega_Double_v_Partners.git

cd <carpeta del proyecto>

Instalar las dependencias

Una vez clonado, instalar las dependencias del proyecto:

npm install
npx playwright install

Ejecutar las pruebas

Para ejecutar las pruebas automatizadas de login:

npx playwright test

Generar y visualizar el reporte Allure

Una vez finalizadas las pruebas, generar el reporte:

npx allure generate allure-results --clean -o allure-report && npx allure open

Estructura del proyecto

La estructura de carpetas es la siguiente:

- pages/: contiene los objetos de página
  - AddProductPage.ts
  - ConfirmPurchasePage.ts
  - NavigateTo.ts
  - NavigateToMenuPage.ts
  - RegisterPage.ts
  - SecurePage.ts
  - ViewCartPage.ts
- assertions/: validaciones específicas de todo el flujo:
  - SecurePageAssertions.ts
- tests/: contiene el archivo que ejecuta el flujo:
  - test_Double_Partners.spec.ts
  - auditoriaLighthouse.spec.ts
- utils/: funcionalidades auxiliares como capturas:
  - screenshotHelper.ts
- screenshots/: imágenes generadas por cada paso
- allure-results/ y allure-report/: carpetas para reportería con allure reports
- playwright.config.ts: configuración general del entorno de pruebas.
- Gitignore: archivos que no se desean subir al repositorio Github
- lighthouse-report.html: Muestra el reporte de las pruebas de rendimiento y accesibilidad.

Resultados

Tras la ejecución de las pruebas, el sistema genera los siguientes resultados:

Capturas de pantalla paso a paso en la carpeta screenshots/:

- 01-Navigate To.png
- 02-Register User.png
- 03-Test Forgot Password.png
- 04-Login.png
- 05-Add product.png
- 06-Search producto add.png
- 07-Delete producto cart.png
- 08-Change quantity product cart.png
- 09-Confirm purchase.png

Reporte Allure interactivo, accesible desde el navegador
Después de ejecutar el test tejecutar el comando para generar el reporte:
- npx allure generate ./allure-results --clean -o ./allure-report

Luego de generarlo ejecutar el comando para abrir el reporte en el navegador:
- npx allure open ./allure-report

Diseño de caso de prueba en Gherkin

Feature: Automatización de flujos críticos

Background:

Given que el usuario accede al sitio https://opencart.abstracta.us/

Scenario: Registro de usuario exitoso

Given que el usuario navega a la página de registro desde "My Account > Register"

When completa todos los campos obligatorios del formulario

And acepta los términos y condiciones

And hace clic en "Continue"

Then debería ver el mensaje "Your Account Has Been Created!"

Scenario: Inicio de sesión exitoso

Given que el usuario tiene una cuenta registrada

When navega a "My Account > Login"

And ingresa credenciales válidas

And hace clic en "Login"

Then debería ser redirigido al "My Account Dashboard"

Scenario: Restablecimiento de contraseña

Given que el usuario tiene una cuenta registrada

When navega a "My Account > Login > Forgotten Password"

And ingresa su correo registrado

And hace clic en "Continue"

Then debería ver el mensaje "An email with a confirmation link has been sent your email address."

Scenario: Navegar a la sección Show all laptops & notebooks

Given que el usuario está en la página principal

When navega a "Laptops & Notebooks > Show all laptops & notebooks"

Then debería ver el listado de todos los productos de la categoría

Scenario: Agregar MacBook Pro al carrito

Given que el usuario navega a la sección "Laptops & Notebooks"

When selecciona el producto "MacBook Pro"

And hace clic en "Add to Cart"

Then debería ver el mensaje "Success: You have added MacBook Pro to your shopping cart!"

Scenario: Buscar y agregar Samsung Galaxy Tablet al carrito

Given que el usuario está en la página principal

When busca el producto "Samsung Galaxy" en la barra de búsqueda

And selecciona el producto de la lista de resultados

And hace clic en "Add to Cart"

Then debería ver el mensaje de confirmación de que el producto fue agregado al carrito

Scenario: Eliminar MacBook Pro del carrito

Given que el usuario tiene el producto "MacBook Pro" en el carrito

When navega al carrito de compras

And elimina el producto "MacBook Pro"

Then el producto ya no debería aparecer en el carrito

And el total debería actualizarse correctamente

Scenario: Agregar otra unidad de Samsung Galaxy Tablet

Given que el usuario tiene "Samsung Galaxy Tablet" en el carrito

When cambia la cantidad de 1 a 2

And hace clic en "Update"

Then el carrito debería mostrar 2 unidades de "Samsung Galaxy Tablet"

And el precio total debería actualizarse correctamente

Scenario: Completar proceso de compra

Given que el usuario tiene productos en el carrito

When hace clic en "Checkout"

And completa los datos de facturación, entrega y método de pago

And confirma la orden

Then debería ver el mensaje "Your order has been placed!"

El script para automatizar el flujo de usuario se realizó en Playwright

- Es modular, reutilizable y fácil de entender.
- Se usa un patron de diseño POM (Page Object Model)
- Para los reportes se usa Allure Report
- Para las pruebas Performance y de Accesibilidad de la pagina se usa LightHouse
- Se utiliza la librería Faker para generar datos dinámicos en la automatización sin repetir datos, para instalarla en visual studio se ejecuta en consola el comando:
  -- npm install @faker-js/faker

Pruebas de accesibilidad y rendimiento

- Para instalar las dependencias de Lighthouse se ejecuta por consola el comando:
npm install --save-dev playwright @playwright/test lighthouse chrome-launcher

- Pruebas de accesibilidad: Se utilizo Lighthouse
- Pruebas de rendimiento: Se utilizo Lighthouse
- Para visualizar el reporte generado tenemos 2 posibilidades:

a. Ejecutar el comando *start lighthouse-report.html*

b. En la raíz del proyecto existe un archivo *lighthouse-report.html* abrirlo con el navegador de su preferencia.

Análisis e informe de pruebas

Se ejecuta el reporte con Allure Reports, mostrando como resultado que el caso de prueba ejecutado es exitoso.

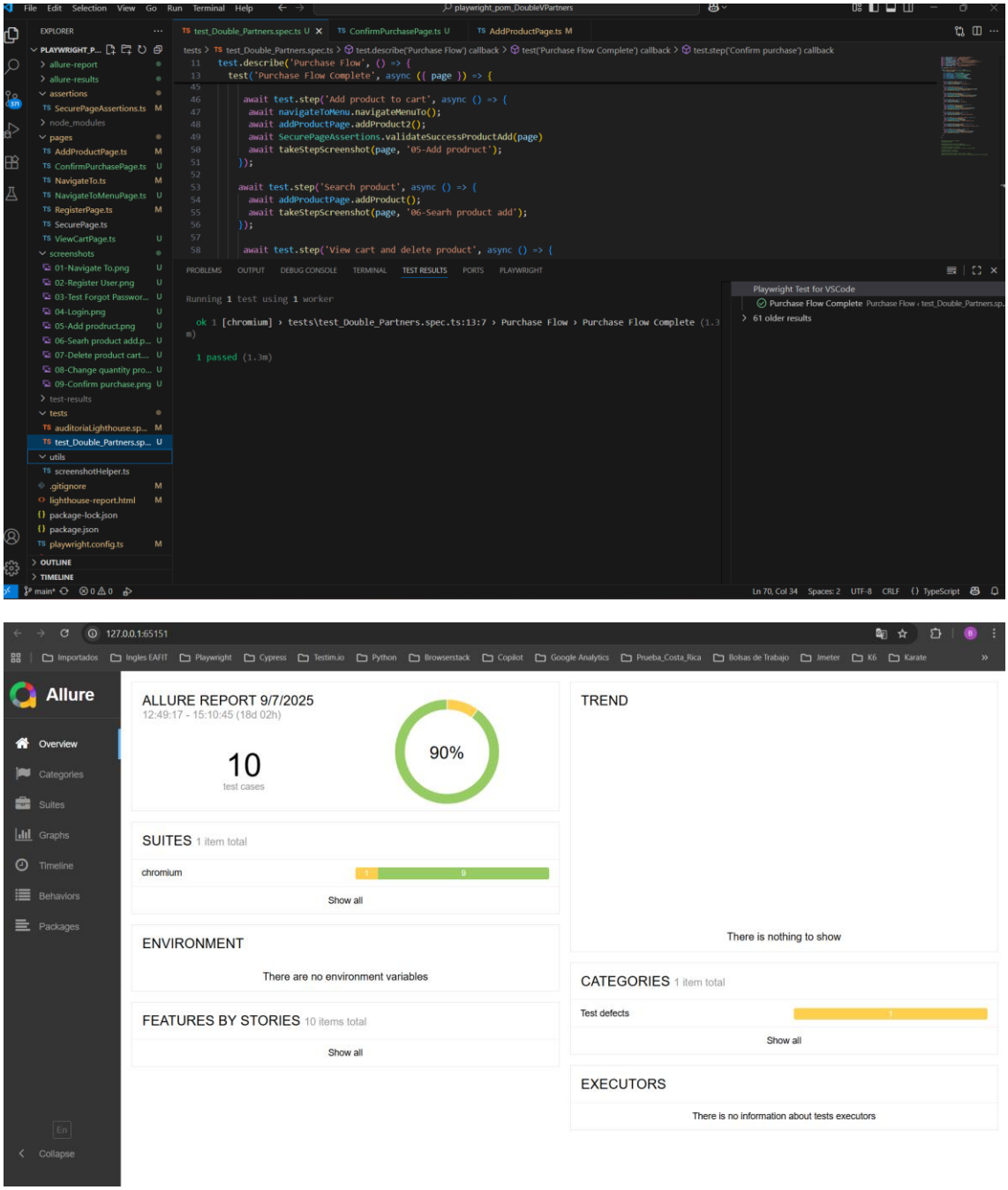

Figura 1

Grafico en Allure total de casos ejecutados

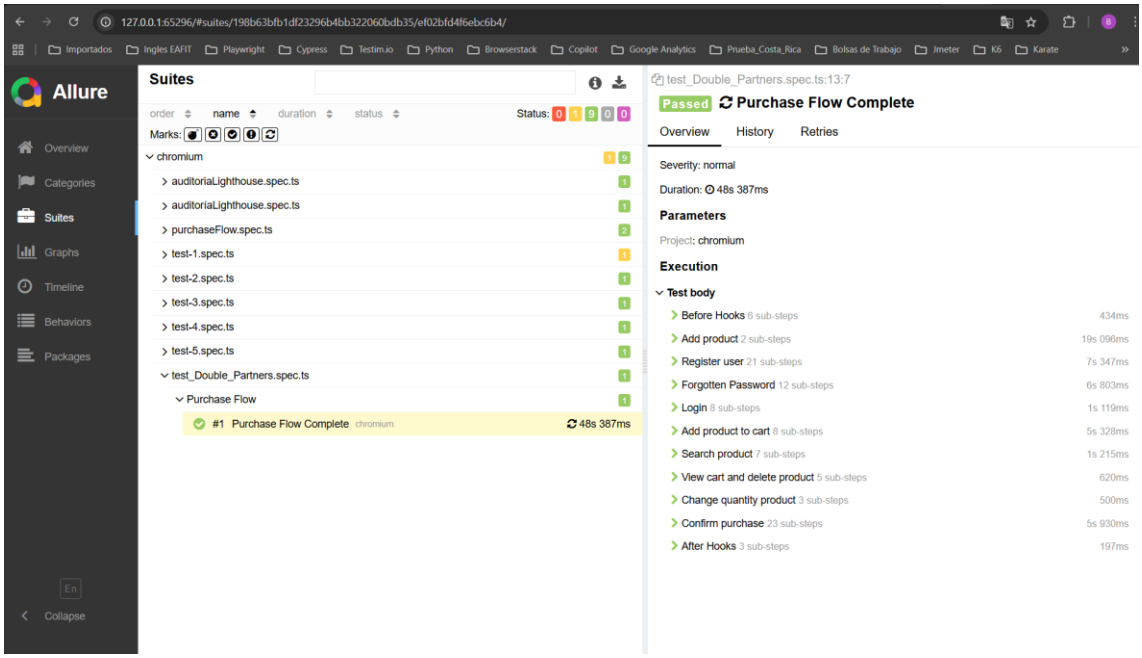

Figura 2

Grafico en Allure suite de casos ejecutados

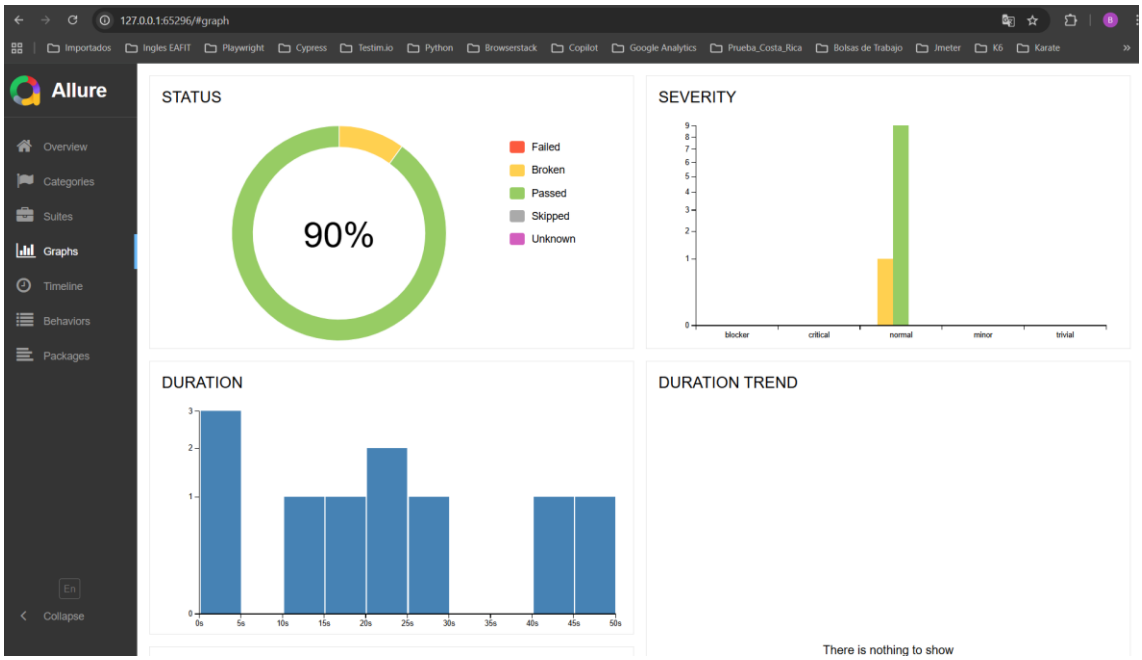

Figura 3

Grafico en Allure total de casos ejecutados

Puntuaciones de Performance con LightHouse

Indica que el performance para ese caso es de 76 de acuerdo con la escala que dice que para un performance medio es de 50-89, nos dice que el performance obtenido es nivel medio

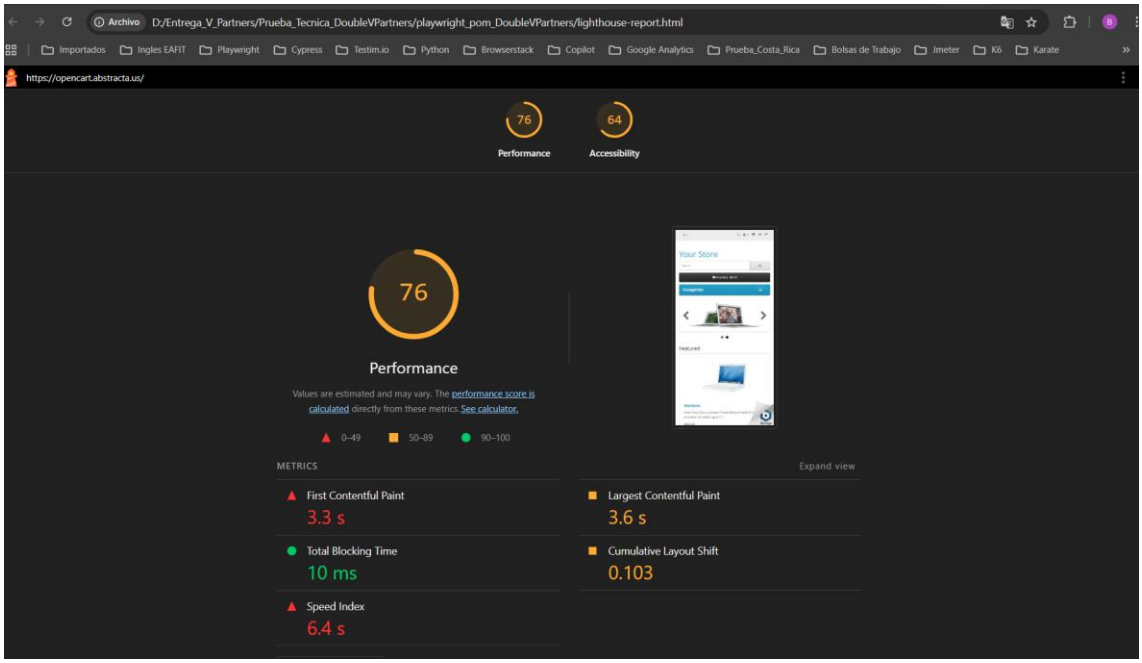

Figura 5

Puntuación de prueba performance

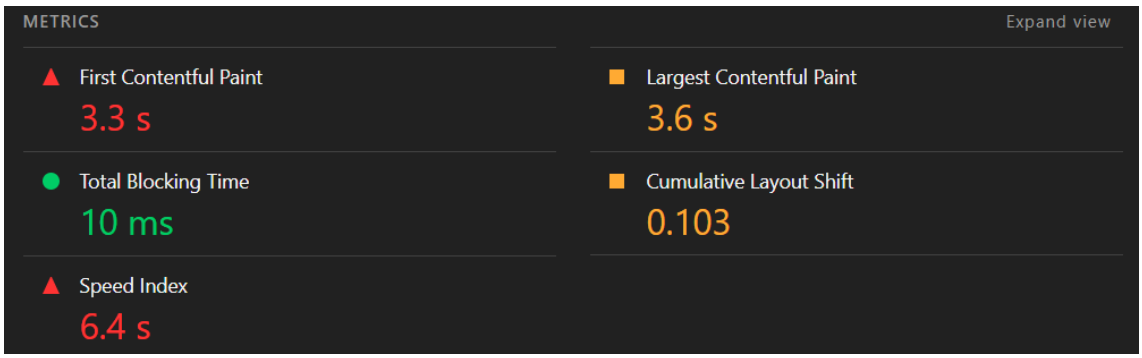

Figura 6

Métricas de prueba performance

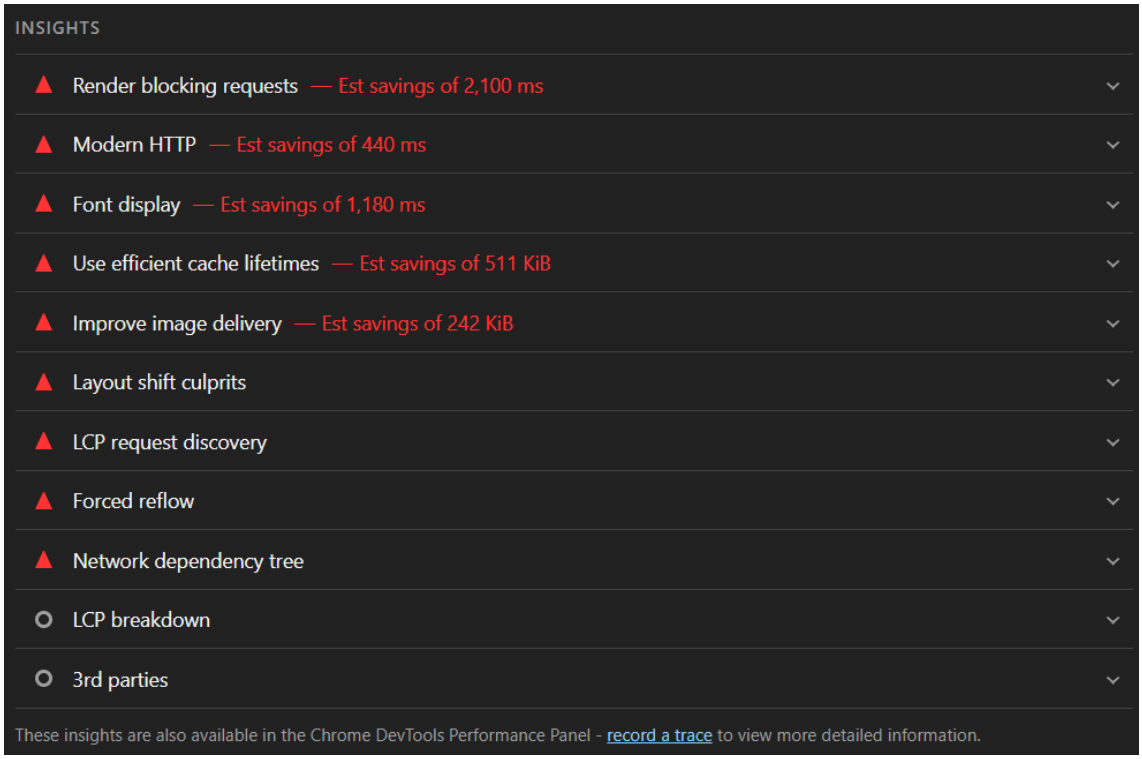

Figura 7

Insights de prueba performance

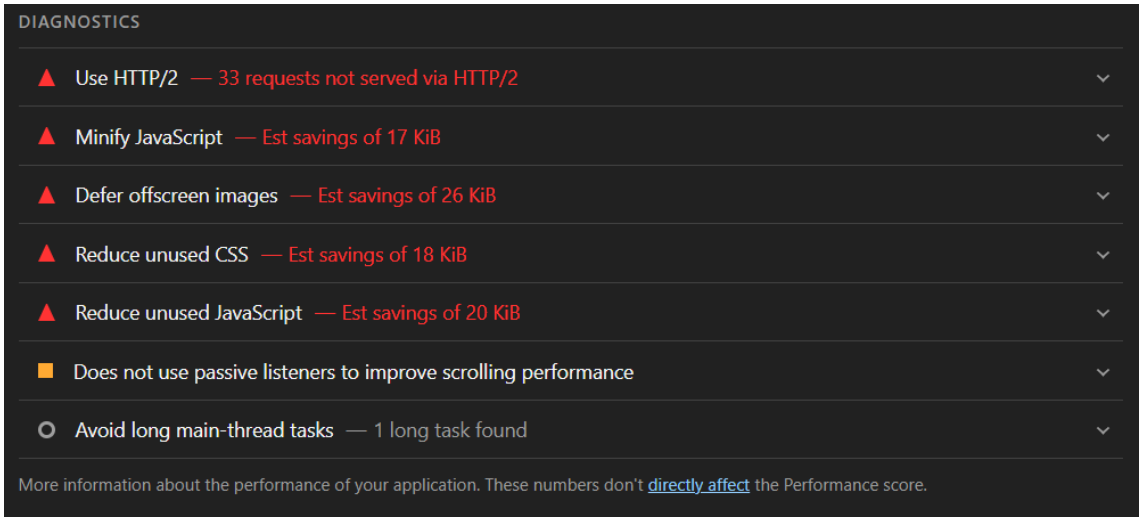

Figura 8

Diagnostics de prueba performance

Puntuaciones de Accesibilidad

Indica que la accesibilidad para ese caso es de 84 es un promedio aceptable

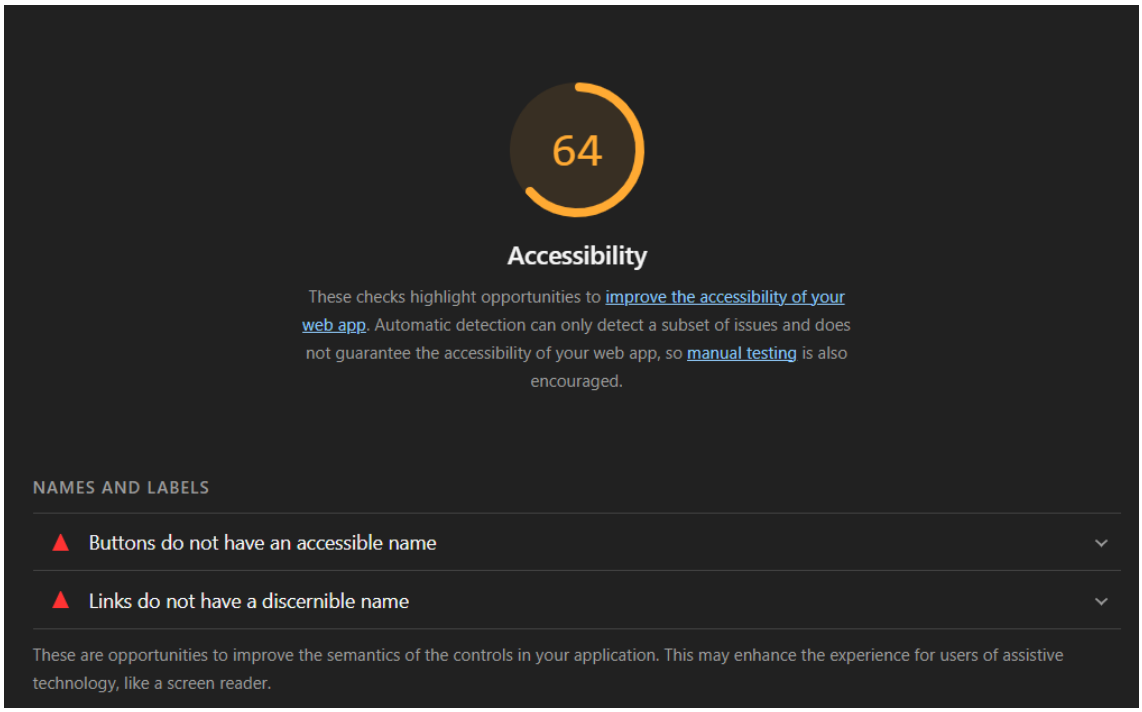

Figura 9

Prueba de accesibilidad

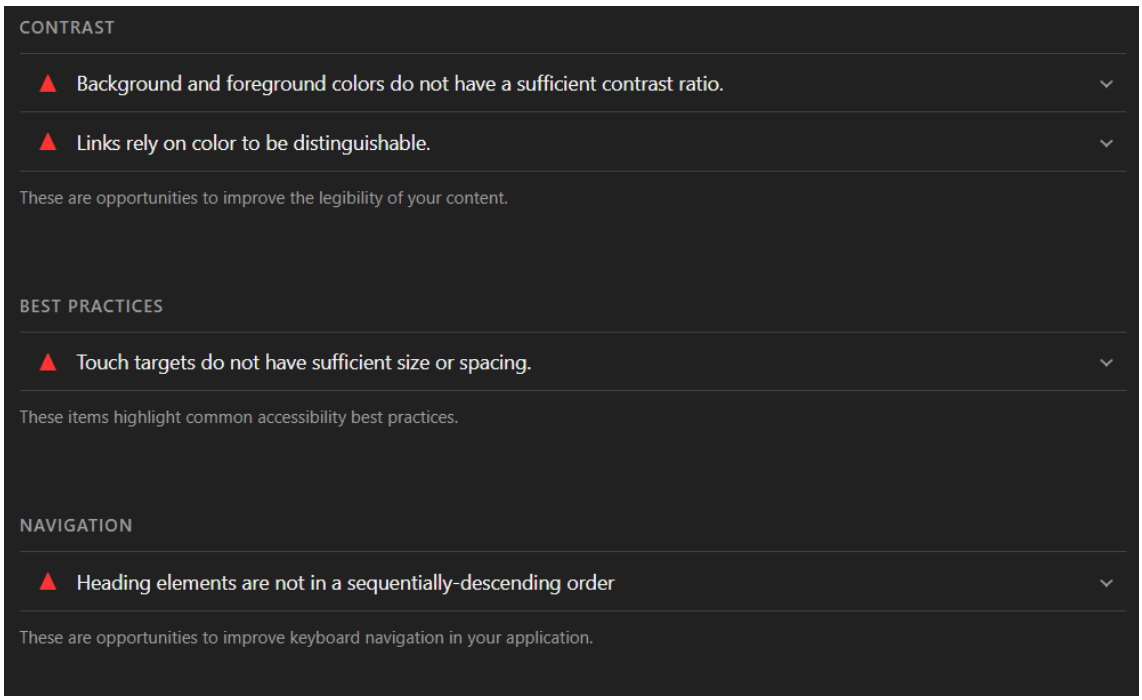

Figura 9

Prueba de accesibilidad

Resultados por categoría:
Performance: (64/100 – Nivel Medio)
Métricas principales:

- First Contentful Paint (FCP): 3.3 s (mejorable)
- Largest Contentful Paint (LCP): 3.6 s (aceptable, objetivo < 2.5s)
- Total Blocking Time (TBT): 10 ms (muy bueno)
- Cumulative Layout Shift (CLS): 0.103 (bueno, objetivo < 0.1)
- Speed Index: 6.4 s (alto, necesita optimización)

Oportunidades de mejora:

- Recursos bloqueando renderizado (ahorro estimado 2.1 s).
- Migrar a HTTP/2 (33 requests no servidos por HTTP/2).
- Configurar font-display para evitar bloqueos de fuentes (ahorro 1.1 s).
- Implementar caché eficiente (ahorro estimado 511 KiB).
- Optimizar entrega de imágenes (ahorro estimado 242 KiB).
- Minificar y reducir JS/CSS sin uso.

Fortaleza: Bajo tiempo de bloqueo (TBT).
Debilidad: Velocidad de renderizado y optimización de imágenes/recursos, con esta debilidad es posible que la automatización sufra fluctuaciones a la hora de ejecutar los casos de prueba descritos.

Accessibility
Puntaje: 64/100 – Bajo.
Problemas detectados:

- Botones sin nombre accesible.
- Links sin texto discernible.
- Contraste insuficiente en colores de fondo y texto.
- Links solo diferenciados por color.
- Estructura de encabezados incorrecta (no secuencial).
- Tamaño insuficiente de touch targets (difícil en dispositivos móviles).

Recomendaciones:

- Añadir aria-label y textos alternativos claros.
- Revisar paleta de colores para cumplir con WCAG AA.
- Asegurar jerarquía correcta de encabezados (h1 > h2 > h3).
- Aumentar tamaño y espacio en botones e íconos interactivos.

Conclusiones:

- El sitio presenta desempeño aceptable en TBT y CLS, pero necesita mejoras en rendimiento visual (FCP, LCP, Speed Index), esto también afecta la automatización de pruebas, haciendo que algunos casos presenten fallos por TimeOut.
- Accesibilidad es el punto más crítico (64/100). Afecta directamente la experiencia de usuarios con limitaciones visuales o que navegan en móvil.
- Se identificaron oportunidades claras: optimización de imágenes, migración a HTTP/2, y mejora de contraste y labels accesibles.
- Con estas optimizaciones, el puntaje de Performance podría subir a +85/100 y el de Accessibility a +90/100.