Aug 2,2025

TESTING REPORT

Report Contents

Table of Contents

Executive Summary

1 High-Level Overview

OVERVIEW	
Total APIs Tested	0 APIs
Total Websites Tested	1 Websites
Pass/Fail Rate	Backend: 0/0 Frontend: 0/5

2 Key Findings

Test Summary

The project's reliability is moderate, largely due to the absence of backend tests and the minimal data on frontend performance. This lack of testing raises concerns over both functionality and user experience. Without successful backend evaluations, the risk of inconsistent service delivery undermines potential user satisfaction. Additionally, frontend components lack structured performance metrics, further obscuring insights into their effectiveness.

What could be better

The fundamental issue lies in the absence of backend API tests, causing significant gaps in assessing the project’s overall functionality and reliability. The frontend lacks clear performance metrics, making it difficult to ascertain stability and effectiveness. This lack of data creates a vulnerability in user experience, highlighting the need for comprehensive metrics and evaluations.

Recommendations

To bolster overall reliability, it is crucial to implement detailed testing for backend APIs, ensuring transparency in functionality. Frontend testing must be enhanced to capture performance metrics that reveal strengths and weaknesses distinctly. This combined approach will guide effective enhancements and ultimately improve stability and user satisfaction.

Frontend UI Test Results

3 Test Coverage Summary

This report summarizes the frontend UI testing results for the application. TestSprite’s AI agent automatically generated and executed tests based on the UI structure, user interaction flows, and visual components. The tests aimed to validate core functionalities, visual correctness, and responsiveness across different states.

URL NAME	TEST CASES	PASS/FAIL RATE
ui testing	5	0 Pass/5 Fail

Note
The test cases were generated using real-time analysis of the application's UI hierarchy and user flows. Some visual and functional validations were adapted dynamically based on runtime DOM changes.

4 Test Execution Summary

Ui Testing Execution Summary

TEST CASE	TEST DESCRIPTION	IMPACT	STATUS
Test Local Server Accessibility	Ensure the local server is running and can be accessed without errors from the current machine.	High	Failed
Verify Firewall and Proxy Setup	Check if the browser's proxy settings and the firewall configuration allow connections to localhost.	High	Failed
Network Configuration Check	Verify network settings and configurations that may prevent connection to localhost, including checking IP bindings.	Medium	Failed
Simulate Server Downtime	Ensure appropriate error messages and page behavior if the server is not reachable or down.	Low	Failed
Load Application on Different Devices	Test whether the application can be accessed from different devices within the same network.	Medium	Failed

5 Test Execution Breakdown

Ui Testing Failed Test Details

Test Local Server Accessibility

ATTRIBUTES	
Status	Failed
Priority	High
Description	Ensure the local server is running and can be accessed without errors from the current machine.
Preview Link	https://testsprite-videos.s3.us-east-1.amazonaws.com/e42844e8-a0f1-70bb-bb93-7c206863e54a/1754087452384967//tmp/32bffcd9-623d-4b8e-9294-f2bb53ac8057/result.webm

Test Code

```python
import asyncio
from playwright import async_api

async def run_test():
    pw = None
    browser = None
    context = None

    try:
        # Start a Playwright session in asynchronous mode
        pw = await async_api.async_playwright().start()

        # Launch a Chromium browser in headless mode with custom arguments
        browser = await pw.chromium.launch(
            headless=True,
            args=[
                "--window-size=1280,720",         # Set the browser window size
                "--disable-dev-shm-usage",        # Avoid using /dev/shm which can cause issues in containers
                "--ipc=host",                     # Use host-level IPC for better stability
                "--single-process"                # Run the browser in a single process mode
            ],
        )

        # Create a new browser context (like an incognito window)
        context = await browser.new_context()
        context.set_default_timeout(5000)

        # Open a new page in the browser context
        page = await context.new_page()

        # Navigate to your target URL and wait until the network request is committed
        await page.goto("http://localhost:3000/", wait_until="commit", timeout=10000)

        # Wait for the main page to reach DOMContentLoaded state (optional for stability)
        try:
            await page.wait_for_load_state("domcontentloaded", timeout=3000)
        except async_api.Error:
            pass

        # Iterate through all iframes and wait for them to load as well
        for frame in page.frames:
            try:
                await frame.wait_for_load_state("domcontentloaded", timeout=3000)
            except async_api.Error:
                pass

        # Interact with the page elements to simulate user flow
```

```
            await asyncio.sleep(5)

    finally:
        if context:
            await context.close()
        if browser:
            await browser.close()
        if pw:
            await pw.stop()

asyncio.run(run_test())

```

Verify Firewall and Proxy Setup

ATTRIBUTES

Status	Failed
Priority	High
Description	Check if the browser's proxy settings and the firewall configuration allow connections to localhost.

Test Code

```python
import asyncio
from playwright import async_api

async def run_test():
    pw = None
    browser = None
    context = None

    try:
        # Start a Playwright session in asynchronous mode
        pw = await async_api.async_playwright().start()

        # Launch a Chromium browser in headless mode with custom arguments
        browser = await pw.chromium.launch(
            headless=True,
            args=[
                "--window-size=1280,720",         # Set the browser window size
                "--disable-dev-shm-usage",        # Avoid using /dev/shm which can cause issues in containers
                "--ipc=host",                     # Use host-level IPC for better stability
                "--single-process"                # Run the browser in a single process mode
            ],
        )

        # Create a new browser context (like an incognito window)
        context = await browser.new_context()
        context.set_default_timeout(5000)

        # Open a new page in the browser context
        page = await context.new_page()

        # Navigate to your target URL and wait until the network request is committed
        await page.goto("http://localhost:3000/", wait_until="commit", timeout=10000)

        # Wait for the main page to reach DOMContentLoaded state (optional for stability)
        try:
            await page.wait_for_load_state("domcontentloaded", timeout=3000)
        except async_api.Error:
            pass

        # Iterate through all iframes and wait for them to load as well
        for frame in page.frames:
            try:
                await frame.wait_for_load_state("domcontentloaded", timeout=3000)
            except async_api.Error:
                pass

        # Interact with the page elements to simulate user flow

```

```
            await asyncio.sleep(5)

        finally:
            if context:
                await context.close()
            if browser:
                await browser.close()
            if pw:
                await pw.stop()

asyncio.run(run_test())

```

Error

Error: Page.goto: net::ERR_CONNECTION_REFUSED at http://localhost:3000/ Call log: - navigating to "http://localhost:3000/", waiting until "commit"

Cause

The service is not running on the specified port (3000), or the server may not be properly configured to accept connections from localhost.

Fix

Ensure that the server application is running and listening on port 3000. Check for any firewall rules or network configurations that may be preventing access to this port.

Network Configuration Check

ATTRIBUTES

Status	Failed
Priority	Medium
Description	Verify network settings and configurations that may prevent connection to localhost, including checking IP bindings.
Preview Link	https://testsprite-videos.s3.us-east-1.amazonaws.com/e42844e8-a0f1-70bb-bb93-7c206863e54a/175408745248514//tmp/0f163536-4178-415b-b10d-a4abc7d30b8c/result.webm

Test Code

```
import asyncio
from playwright import async_api

async def run_test():
    pw = None
    browser = None
    context = None

    try:
        # Start a Playwright session in asynchronous mode
        pw = await async_api.async_playwright().start()

        # Launch a Chromium browser in headless mode with custom arguments
        browser = await pw.chromium.launch(
            headless=True,
            args=[
                "--window-size=1280,720",         # Set the browser window size
                "--disable-dev-shm-usage",        # Avoid using /dev/shm which can cause issues in containers
                "--ipc=host",                     # Use host-level IPC for better stability
                "--single-process"                # Run the browser in a single process mode
            ],
        )

        # Create a new browser context (like an incognito window)
        context = await browser.new_context()
        context.set_default_timeout(5000)

        # Open a new page in the browser context
        page = await context.new_page()

        # Navigate to your target URL and wait until the network request is committed
        await page.goto("http://localhost:3000/", wait_until="commit", timeout=10000)

        # Wait for the main page to reach DOMContentLoaded state (optional for stability)
        try:
            await page.wait_for_load_state("domcontentloaded", timeout=3000)
        except async_api.Error:
            pass

        # Iterate through all iframes and wait for them to load as well
        for frame in page.frames:
            try:
                await frame.wait_for_load_state("domcontentloaded", timeout=3000)
            except async_api.Error:
                pass

        # Interact with the page elements to simulate user flow

```

```python
            await asyncio.sleep(5)

    finally:
        if context:
            await context.close()
        if browser:
            await browser.close()
        if pw:
            await pw.stop()

asyncio.run(run_test())

```

Simulate Server Downtime

ATTRIBUTES

Status	Failed
Priority	Low
Description	Ensure appropriate error messages and page behavior if the server is not reachable or down.
Preview Link	https://testsprite-videos.s3.us-east-1.amazonaws.com/e42844e8-a0f1-70bb-bb93-7c206863e54a/1754087451998132//tmp/51b8d1e9-09c0-421a-9590-97666a6fab37/result.webm

Test Code

```python
import asyncio
from playwright import async_api

async def run_test():
    pw = None
    browser = None
    context = None

    try:
        # Start a Playwright session in asynchronous mode
        pw = await async_api.async_playwright().start()

        # Launch a Chromium browser in headless mode with custom arguments
        browser = await pw.chromium.launch(
            headless=True,
            args=[
                "--window-size=1280,720",         # Set the browser window size
                "--disable-dev-shm-usage",        # Avoid using /dev/shm which can cause issues in containers
                "--ipc=host",                     # Use host-level IPC for better stability
                "--single-process"                # Run the browser in a single process mode
            ],
        )

        # Create a new browser context (like an incognito window)
        context = await browser.new_context()
        context.set_default_timeout(5000)

        # Open a new page in the browser context
        page = await context.new_page()

        # Navigate to your target URL and wait until the network request is committed
        await page.goto("http://localhost:3000/", wait_until="commit", timeout=10000)

        # Wait for the main page to reach DOMContentLoaded state (optional for stability)
        try:
            await page.wait_for_load_state("domcontentloaded", timeout=3000)
        except async_api.Error:
            pass

        # Iterate through all iframes and wait for them to load as well
        for frame in page.frames:
            try:
                await frame.wait_for_load_state("domcontentloaded", timeout=3000)
            except async_api.Error:
                pass

        # Interact with the page elements to simulate user flow

```

```python
            await asyncio.sleep(5)

    finally:
        if context:
            await context.close()
        if browser:
            await browser.close()
        if pw:
            await pw.stop()

asyncio.run(run_test())

```

Load Application on Different Devices

ATTRIBUTES

Status	Failed
Priority	Medium
Description	Test whether the application can be accessed from different devices within the same network.
Preview Link	https://testsprite-videos.s3.us-east-1.amazonaws.com/e42844e8-a0f1-70bb-bb93-7c206863e54a/1754087452268941//tmp/d77bbc89-2a7b-48e1-8e1e-dd3cc5443c91/result.webm

Test Code

```python
import asyncio
from playwright import async_api

async def run_test():
    pw = None
    browser = None
    context = None

    try:
        # Start a Playwright session in asynchronous mode
        pw = await async_api.async_playwright().start()

        # Launch a Chromium browser in headless mode with custom arguments
        browser = await pw.chromium.launch(
            headless=True,
            args=[
                "--window-size=1280,720",         # Set the browser window size
                "--disable-dev-shm-usage",        # Avoid using /dev/shm which can cause issues in containers
                "--ipc=host",                     # Use host-level IPC for better stability
                "--single-process"                # Run the browser in a single process mode
            ],
        )

        # Create a new browser context (like an incognito window)
        context = await browser.new_context()
        context.set_default_timeout(5000)

        # Open a new page in the browser context
        page = await context.new_page()

        # Navigate to your target URL and wait until the network request is committed
        await page.goto("http://localhost:3000/", wait_until="commit", timeout=10000)

        # Wait for the main page to reach DOMContentLoaded state (optional for stability)
        try:
            await page.wait_for_load_state("domcontentloaded", timeout=3000)
        except async_api.Error:
            pass

        # Iterate through all iframes and wait for them to load as well
        for frame in page.frames:
            try:
                await frame.wait_for_load_state("domcontentloaded", timeout=3000)
            except async_api.Error:
                pass

        # Interact with the page elements to simulate user flow
```

```python
            await asyncio.sleep(5)

    finally:
        if context:
            await context.close()
        if browser:
            await browser.close()
        if pw:
            await pw.stop()

asyncio.run(run_test())

```