TestSprite

Sep 10,2025

TESTING REPORT

Report Contents

This report provides key insights from TestSprite's AI-powered testing. For questions or customized needs, contact us using Calendly or join our Discord community.

Table of Contents

Executive Summary

1 High-Level Overview

OVERVIEW	
Total APIs Tested	1 APIs
Total Websites Tested	0 Websites
Pass/Fail Rate	Backend: 4/6 Frontend: 0/0

2 Key Findings

Test Summary

The project exhibits a stable output with a reasonable quality score, though backend testing is limited, highlighting potential areas for improvement in backend APIs. Frontend tests aren't available, which restricts the analysis and may affect overall user experience. Observed patterns indicate that continued testing is required to assure stability, ensuring both components meet robust quality standards.

What could be better

The major weakness is the absence of completed frontend tests, which limits visibility into user interaction performance. The limited backend testing data also raises concerns about untested functionalities that could lead to undiscovered critical failures affecting the overall project reliability.

Recommendations

It is crucial to initiate comprehensive frontend testing to ensure thorough evaluation of user experience and performance. Additionally, enhancing backend API tests will help uncover potential weaknesses and foster improvements, ultimately solidifying overall project stability and reliability.

Backend API Test Results

3 Test Coverage Summary

API NAME	TEST CASES	TEST CATEGORY	PASS/FAIL RATE
ListarPerfisdeUsuários	10	5 Functional Tests 5 Edge Case Tests	4 Pass/6 Fail

> **Note**
> The test cases were generated based on the API specifications and observed behaviors. Some tests were adapted dynamically during execution based on API responses.

4 Test Execution Summary

ListarPerfisdeUsuários Execution Summary

TEST CASE	TEST DESCRIPTION	IMPACT	STATUS

Functional Tests			
Unauthorized Access Test	Attempt to send a GET request without a JWT token in the authorization header. Verify that the response status is 401 Unauthorized, ensuring proper RLS enforcement.	High	Passed
Empty Response Handling	Test the API with a JWT that has no accessible profiles. Verify that the response returns an empty array with a 200 status to handle cases with no data correctly.	Medium	Failed
Valid GET Request	Send a valid GET request to the API endpoint with required headers. Verify the response status is 200 OK and contains a JSON array of user profiles.	High	Failed
Check Required Fields	After a successful GET request, validate that each returned profile object includes required fields: id, email, and full_name, to ensure data integrity.	High	Passed
Incorrect Apikey Handling	Send a GET request with an incorrect apikey in the headers. Confirm the response status is 403 Forbidden, ensuring access control is functioning correctly.	Medium	Passed
Edge Case Tests			
Long Email Field	Check the response for a profile where the email field has maximum length. Ensure the profile is returned correctly and adheres to email constraints.	Medium	Passed
Duplicate Profiles Handling	Verify how the API handles duplicate profiles in the database. Check that it still returns unique profiles without duplication in the response.	Low	Failed
Special Characters in Full Name	Create a test to include special characters in the full_name field in outputs. Verify that profiles display and process such characters without errors.	Medium	Failed
Check Response Time	Measure the response time for a valid GET request. This will ensure that the API responds efficiently under normal conditions while adhering to performance standards.	Medium	Failed
Malformed JWT Token	Send a GET request with a malformed JWT token in the authorization header. Ensure the response status is 401 Unauthorized to validate error handling.	High	Failed

5 Test Execution Breakdown

ListarPerfisdeUsuários Failed Test Details

Duplicate Profiles Handling

ATTRIBUTES	
Status	Failed
Priority	Low
Description	Verify how the API handles duplicate profiles in the database. Check that it still returns unique profiles without duplication in the response.

Test Code

```python
import requests
import json
import time

def test_duplicate_profiles_handling():
    url = "https://rsvjnndhbyyxktbczlnk.supabase.co/rest/v1/profiles"
    headers = {
        "Authorization": "Bearer
        eyJhbGciOiJIUzI1NiIsImtpZCI6InFFaFA0TDltV0hnVXMrUlYiLCJ0eXAiOi
        JKV1Q....",
        "apikey": "your_api_key"
    }

    start_time = time.time()
    response = requests.get(url, headers=headers)
    response_time = time.time() - start_time

    print(f"Response Status Code: {response.status_code}")
    print(f"Response Time: {response_time:.2f} seconds")
    print(f"Response Body: {response.text}")

    assert response.status_code == 200, f"Expected status code 200
    but got {response.status_code}"
    assert response_time < 2, "Response time exceeded 2 seconds"

    profiles = response.json()
    for field in ["id", "email", "full_name"]:
        assert field in profiles[0], f"Expected field '{field}' not
        found in profiles"

test_duplicate_profiles_handling()
```

Error

Expected status code 200 but got 401

Trace

Duplicate Profiles Handling

```
Traceback (most recent call last):
  File "/var/task/main.py", line 60, in target
    exec(code, env)
  File "<string>", line 27, in <module>
  File "<string>", line 20, in test_duplicate_profiles_handling
AssertionError: Expected status code 200 but got 401

```

Cause

The API returned a 401 Unauthorized status code, indicating that the Bearer token provided in the request headers is either invalid, expired, or the user does not have the required permissions to access the endpoint due to the Row Level Security (RLS) rules configured in Supabase.

Fix

Verify that the Bearer token is valid and has not expired. Ensure that the user associated with the token has the appropriate permissions to access the profiles endpoint under the current RLS settings. If necessary, regenerate the token or adjust the RLS policies to allow the intended user access.

Empty Response Handling

ATTRIBUTES	
Status	Failed
Priority	Medium
Description	Test the API with a JWT that has no accessible profiles. Verify that the response returns an empty array with a 200 status to handle cases with no data correctly.

Test Code

```python
import requests
import json
import time

def test_empty_response_handling():
    url = "https://rsvjnndhbyyxktbczlnk.supabase.co/rest/v1/profiles"
    headers = {
        "Authorization": "Bearer eyJhbGciOiJIUzI1NiIsImtpZCI6InFFaFA0TDltV0hnVXMrUlYiLCJ0eXAiOiJKV1Q....",
        "apikey": "your_api_key"  # replace with actual API key if needed
    }

    start_time = time.time()
    response = requests.get(url, headers=headers)
    response_time = time.time() - start_time

    print("Response Status Code:", response.status_code)
    print("Response Time:", response_time)
    print("Response JSON:", response.json())

    assert response.status_code == 200, f"Expected status 200 but got {response.status_code}"
    assert response_time < 2, f"Response time is too slow: {response_time} seconds"

    data = response.json()
    assert isinstance(data, list), f"Expected response to be a list but got {type(data)}"
    assert len(data) == 0, f"Expected empty response but got {len(data)} items"

test_empty_response_handling()
```

Error

Expected status 200 but got 401

Trace

Empty Response Handling

```
Traceback (most recent call last):
  File "/var/task/main.py", line 60, in target
    exec(code, env)
  File "<string>", line 27, in <module>
  File "<string>", line 20, in test_empty_response_handling
AssertionError: Expected status 200 but got 401

```

Cause

The API returned a 401 Unauthorized status, indicating that the provided Bearer token is either invalid, expired, or does not have the necessary permissions to access the '/profiles' endpoint due to Row Level Security (RLS) restrictions.

Fix

Ensure that the Bearer token is valid and has not expired. Verify the token's claims to ensure the user associated with the token has the required permissions to access the '/profiles' endpoint under the current RLS policy. If necessary, update the RLS policy to allow access for the user.

Special Characters in Full Name

ATTRIBUTES	
Status	Failed
Priority	Medium
Description	Create a test to include special characters in the full_name field in outputs. Verify that profiles display and process such characters without errors.

Test Code

```
import requests
import json
import time

def test_get_profiles_special_characters_in_full_name():
    url = "https://rsvjnndhbyyxktbczlnk.supabase.co/rest/v1/profiles"
    headers = {
        "Authorization": "Bearer
        eyJhbGciOiJIUzI1NiIsImtpZCI6InFFaFA0TDltV0hnVXMrUlYiLCJ0eXAiOi
        JKV1Q....",
        "apikey": "your_apikey_here"
    }

    start_time = time.time()
    response = requests.get(url, headers=headers)
    end_time = time.time()

    print("Response Status Code:", response.status_code)
    print("Response Time:", end_time - start_time)
    print("Response Body:", response.text)

    assert response.status_code == 200, f"Expected status code 200
    but got {response.status_code}"

    response_json = response.json()
    assert isinstance(response_json, list), "Expected response to be
    a list"

    for profile in response_json:
        assert 'id' in profile, "Profile does not have 'id' field"
        assert 'email' in profile, "Profile does not have 'email'
        field"
        assert 'full_name' in profile, "Profile does not have
        'full_name' field"
        assert isinstance(profile['full_name'], str), "Expected
        'full_name' to be a string"

        # Check for special characters
        if any(char in profile['full_name'] for char in '!@#$%^&*()_
        +-={}[]|;:'",.<>?/'):
            print(f"Profile with special characters: {profile
            ['full_name']}")

test_get_profiles_special_characters_in_full_name()
```

Error

unterminated string literal (detected at line 32) (<string>, line 32)

Trace

Special Characters in Full Name

```
Traceback (most recent call last):
  File "/var/task/main.py", line 60, in target
    exec(code, env)
  File "<string>", line 32
    if any(char in profile['full_name'] for char in '!@#$%^&*()_+-={}
    []|;:'",.<>?/'):

     ^
SyntaxError: unterminated string literal (detected at line 32)

```

Cause

The error is caused by a syntax error in the Python code due to an unterminated string in the check for special characters in the full name. The string is not properly enclosed with quotes, which leads to a SyntaxError when executing the code.

Fix

To fix the issue, ensure all strings are properly enclosed. Specifically, replace the current definition of the special characters string with a properly formatted string, such as: '"!@#$%^&*()_+-={}[]|;:',.<>?/"'. Ensure that the double quotes are enclosed correctly within the single quotes.

Valid GET Request

ATTRIBUTES	
Status	Failed
Priority	High
Description	Send a valid GET request to the API endpoint with required headers. Verify the response status is 200 OK and contains a JSON array of user profiles.

Test Code

```python
import requests
import json
import time

def test_valid_get_request():
    url = "https://rsvjnndhbyyxktbczlnk.supabase.co/rest/v1/profiles"
    headers = {
        "Authorization": "Bearer eyJhbGciOiJIUzI1NiIsImtpZCI6InFFaFA0TDltV0hnVXMrUlYiLCJ0eXAiOiJKV1Q....",
        "apikey": "<YOUR_API_KEY>"
    }

    start_time = time.time()
    response = requests.get(url, headers=headers)
    response_time = time.time() - start_time

    print(f"Response Status Code: {response.status_code}")
    print(f"Response Time: {response_time} seconds")
    print(f"Response Body: {response.text}")

    assert response.status_code == 200, f"Expected status code 200, but got {response.status_code}"
    assert response_time < 2, f"Response time exceeded 2 seconds: {response_time} seconds"

    response_json = response.json()
    if isinstance(response_json, list) and len(response_json) > 0:
        assert 'id' in response_json[0], "Expected 'id' field not found in response"
        assert 'email' in response_json[0], "Expected 'email' field not found in response"
        assert 'full_name' in response_json[0], "Expected 'full_name' field not found in response"
    else:
        assert False, "Expected a JSON array but received a different structure"

test_valid_get_request()
```

Error

Expected status code 200, but got 401

Trace

Valid GET Request

```
Traceback (most recent call last):
  File "/var/task/main.py", line 60, in target
    exec(code, env)
  File "<string>", line 31, in <module>
  File "<string>", line 20, in test_valid_get_request
AssertionError: Expected status code 200, but got 401

```

Cause

The API returned a 401 Unauthorized status code, which indicates that the Bearer token might be invalid, expired, or the user associated with the token does not have the proper permissions to access the resource due to Row-Level Security (RLS) settings.

Fix

Ensure the Bearer token is valid and not expired. Additionally, verify that the user has the necessary permissions configured in the RLS policy. If needed, update the RLS rules to allow access to the user associated with the Bearer token.

Check Response Time

ATTRIBUTES	
Status	Failed
Priority	Medium
Description	Measure the response time for a valid GET request. This will ensure that the API responds efficiently under normal conditions while adhering to performance standards.

Test Code

```python
import requests
import json
import time

def test_get_profiles_response_time():
    url = "https://rsvjnndhbyyxktbczlnk.supabase.co/rest/v1/profiles"
    headers = {
        "Authorization": "Bearer eyJhbGciOiJIUzI1NiIsImtpZCI6InFFaFA0TDltV0hnVXMrUlYiLCJ0eXAiOiJKV1Q....",
        "apikey": "YOUR_API_KEY"  # Replace with the actual API key
    }

    start_time = time.time()
    response = requests.get(url, headers=headers)
    end_time = time.time()

    response_time = end_time - start_time
    print(f"Response time: {response_time} seconds")

    assert response.status_code == 200, f"Expected status code 200, got {response.status_code}"
    assert response_time < 2, f"Expected response time less than 2 seconds, but got {response_time} seconds"

    response_json = response.json()
    assert isinstance(response_json, list), "Expected response to be a JSON array"

    for profile in response_json:
        assert "id" in profile, "Profile is missing 'id' field"
        assert "email" in profile, "Profile is missing 'email' field"
        assert "full_name" in profile, "Profile is missing 'full_name' field"

test_get_profiles_response_time()
```

Error

Expected status code 200, got 401

Trace

Check Response Time

```
Traceback (most recent call last):
  File "/var/task/main.py", line 60, in target
    exec(code, env)
  File "<string>", line 30, in <module>
  File "<string>", line 19, in test_get_profiles_response_time
AssertionError: Expected status code 200, got 401

```

Cause

The API returned a 401 Unauthorized status code, indicating that the Bearer token used for authentication is invalid or expired, or that the API key provided is incorrect or missing.

Fix

Ensure the Bearer token is valid and has not expired. Additionally, verify that the API key included in the headers is correct and has the necessary permissions to access the endpoint.

Malformed JWT Token

ATTRIBUTES	
Status	Failed
Priority	High
Description	Send a GET request with a malformed JWT token in the authorization header. Ensure the response status is 401 Unauthorized to validate error handling.

Test Code

```python
import requests
import json
import time

def test_malformed_jwt_token():
    url = "https://rsvjnndhbyyxktbczlnk.supabase.co/rest/v1/profiles"
    headers = {
        "Authorization": "Bearer malformed_token",
        "apikey":
        "eyJhbGciOiJIUzI1NiIsImtpZCI6InFFaFA0TDltV0hnVXMrUlYiLCJ0eXAiO
        iJKV1Q...."
    }

    start_time = time.time()
    response = requests.get(url, headers=headers)
    response_time = time.time() - start_time

    print(f"Response Status Code: {response.status_code}")
    print(f"Response Time: {response_time} seconds")
    print(f"Response Body: {response.text}")

    assert response.status_code == 200, f"Expected status code 200,
    got {response.status_code}"
    try:
        response_json = response.json()
        assert isinstance(response_json, list), f"Expected response
        to be a list, got {type(response_json)}"
        for profile in response_json:
            assert 'id' in profile, "Profile is missing 'id' field"
            assert 'email' in profile, "Profile is missing 'email'
            field"
            assert 'full_name' in profile, "Profile is missing
            'full_name' field"
    except json.JSONDecodeError:
        print("Response is not valid JSON")

test_malformed_jwt_token()
```

Error

Expected status code 200, got 401

Trace

Malformed JWT Token

```
Traceback (most recent call last):
  File "/var/task/main.py", line 60, in target
    exec(code, env)
  File "<string>", line 31, in <module>
  File "<string>", line 20, in test_malformed_jwt_token
AssertionError: Expected status code 200, got 401

```

Cause

The API is returning a 401 Unauthorized status code because the malformed JWT token is not being recognized as a valid token. This indicates that the API endpoint has proper security measures to ensure only valid tokens are accepted, which is functioning as intended.

Fix

To fix the API issue, the API should ensure that it properly validates the JWT token format and provides clear error messages for different authentication failures. However, since the issue arises from sending a malformed token, ensure that valid JWT tokens are used in this test case. Alternatively, consider enhancing the API response to include more detailed error messages for different failure cases.