```
import base64
import concurrent.futures
import time
from io import BytesIO

import pytest
import requests
from PIL import Image
import os


img = os.environ.get("TEST_IMG")


# Utility function to convert image to Base64
def image_to_base64(image_path):
    with Image.open(image_path) as image:
        buffered = BytesIO()
        image.save(buffered, format="JPEG")
        img_str = base64.b64encode(buffered.getvalue()).decode("utf-8")
    return img_str


@pytest.fixture
def base64_image():
    return image_to_base64(img)
```

```
@pytest.fixture
def request_data(base64_image):
    text_data = {"type": "text", "text": "Describe what is in the image"}
    image_data = {
        "type": "image_url",
        "image_url": {"url": f"{base64_image}"},
    }
    return {
        "model": "cogvlm-chat-17b",
        "messages": [{"role": "user", "content": [text_data, image_data]}],
        "temperature": 0.8,
        "top_p": 0.9,
        "max_tokens": 1024,
    }


@pytest.mark.functionality
def test_api_response_structure(request_data):
    url = "https://api.swarms.world/v1/chat/completions"
    response = requests.post(url, json=request_data)
    assert response.status_code == 200
    response_data = response.json()
    assert "id" in response_data
    assert "created" in response_data
```

```
    assert isinstance(response_data["choices"], list)


@pytest.mark.error_handling
def test_api_with_invalid_image(request_data):
    request_data["messages"][0]["content"][1]["image_url"][
        "url"
    ] = ""
    url = "https://api.swarms.world/v1/chat/completions"
    response = requests.post(url, json=request_data)
    assert response.status_code == 400


@pytest.mark.speed
def test_api_response_time(request_data):
    url = "https://api.swarms.world/v1/chat/completions"
    start = time.time()
    requests.post(url, json=request_data)
    end = time.time()
    assert (
        end - start < 2
    )  # Example threshold: response time should be less than 2 seconds


@pytest.mark.concurrency
@pytest.mark.parametrize("n", [1, 2, 5])
```

```
def test_concurrent_requests(n, request_data):
    url = "https://api.swarms.world/v1/chat/completions"
    responses = [requests.post(url, json=request_data) for _ in range(n)]
    assert all(response.status_code == 200 for response in responses)


@pytest.mark.security
def test_sql_injection_vulnerability(request_data):
    malicious_input = "'; DROP TABLE users; --"
    request_data["messages"][0]["content"][0]["text"] = malicious_input
    url = "https://api.swarms.world/v1/chat/completions"
    response = requests.post(url, json=request_data)
    assert (
        response.status_code == 400
    )  # Assuming that the API properly handles SQL injection attempts


def send_request(request_data):
    url = "https://api.swarms.world/v1/chat/completions"
    response = requests.post(url, json=request_data)
    return response


@pytest.mark.load
def test_load_performance(request_data):
    number_of_requests = 10
```

```
    with concurrent.futures.ThreadPoolExecutor(
        max_workers=number_of_requests
    ) as executor:
        futures = [
            executor.submit(send_request, request_data)
            for _ in range(number_of_requests)
        ]
        start_time = time.time()
        concurrent.futures.wait(futures)
        total_time = time.time() - start_time

    assert total_time < 20, "Handling 10 concurrent requests took too long"



@pytest.mark.stress
def test_stress_system(request_data):
    number_of_requests = 50
    responses = []
    with concurrent.futures.ThreadPoolExecutor(max_workers=10) as executor:
        futures = [
            executor.submit(send_request, request_data)
            for _ in range(number_of_requests)
        ]
        for future in concurrent.futures.as_completed(futures):
            responses.append(future.result())
```

```
    success_responses = [
        response for response in responses if response.status_code == 200
    ]
    assert (
        len(success_responses) > 0.9 * number_of_requests
    ), "Less than 90% of requests were successful under stress"


@pytest.mark.integration
def test_integration_with_text_and_image(request_data):
    # This test assumes the API returns a specific part of the response that we can assert on
    response = send_request(request_data)
    assert response.status_code == 200, "Failed to get a successful response"
    response_data = response.json()
    # Example assertion, the actual key/value to check will depend on the API's response structure
    assert (
        "description" in response_data
    ), "Response data does not include the expected 'description' key"


@pytest.mark.timeout(5)
def test_timeout_behavior(request_data):
    # This test assumes that the API should handle requests within a specified timeout
    with pytest.raises(requests.exceptions.ReadTimeout):
        requests.post(
            "https://api.swarms.world/v1/chat/completions",
```

```
            json=request_data,
            timeout=0.01,
        )



@pytest.mark.security
def test_header_injection(request_data):
    url = "https://api.swarms.world/v1/chat/completions"
    # Injecting a malicious header
    headers = {
        "User-Agent": "python-requests/2.25.1",
        "X-Custom-Inject": "test; curl http://example.com",
    }
    response = requests.post(url, json=request_data, headers=headers)
    assert (
        response.status_code == 400
    ), "API did not reject a request with a malicious header injection"
```