```
import asyncio
import os
from unittest.mock import AsyncMock, Mock, mock_open, patch

import pytest
from aiohttp import ClientResponseError
from dotenv import load_dotenv
from requests.exceptions import RequestException

from swarm_models.gpt4_vision_api import GPT4VisionAPI

load_dotenv()

custom_api_key = os.environ.get("OPENAI_API_KEY")
img = "images/swarms.jpeg"


@pytest.fixture
def vision_api():
    return GPT4VisionAPI(openai_api_key="test_api_key")


def test_init(vision_api):
    assert vision_api.openai_api_key == "test_api_key"
```

```
def test_encode_image(vision_api):
    with patch(
        "builtins.open",
        mock_open(read_data=b"test_image_data"),
        create=True,
    ):
        encoded_image = vision_api.encode_image(img)
        assert encoded_image == "dGVzdF9pbWFnZV9kYXRh"



def test_run_success(vision_api):
    expected_response = {"This is the model's response."}
    with patch(
        "requests.post",
        return_value=Mock(json=lambda: expected_response),
    ) as mock_post:
        result = vision_api.run("What is this?", img)
        mock_post.assert_called_once()
        assert result == "This is the model's response."



def test_run_request_error(vision_api):
    with patch(
        "requests.post", side_effect=RequestException("Request Error")
    ):
        with pytest.raises(RequestException):
```

```
            vision_api.run("What is this?", img)



def test_run_response_error(vision_api):
    expected_response = {"error": "Model Error"}
    with patch(
        "requests.post",
        return_value=Mock(json=lambda: expected_response),
    ):
        with pytest.raises(RuntimeError):
            vision_api.run("What is this?", img)



def test_call(vision_api):
    expected_response = {
        "choices": [{"text": "This is the model's response."}]
    }
    with patch(
        "requests.post",
        return_value=Mock(json=lambda: expected_response),
    ) as mock_post:
        result = vision_api("What is this?", img)
        mock_post.assert_called_once()
        assert result == "This is the model's response."
```

```
@pytest.fixture
def gpt_api():
    return GPT4VisionAPI()


def test_initialization_with_default_key():
    api = GPT4VisionAPI()
    assert api.openai_api_key == custom_api_key


def test_initialization_with_custom_key():
    custom_key = custom_api_key
    api = GPT4VisionAPI(openai_api_key=custom_key)
    assert api.openai_api_key == custom_key


def test_run_with_exception(gpt_api):
    task = "What is in the image?"
    img_url = img
    with patch(
        "requests.post", side_effect=Exception("Test Exception")
    ):
        with pytest.raises(Exception):
            gpt_api.run(task, img_url)
```

```
def test_call_method_successful_response(gpt_api):
    task = "What is in the image?"
    img_url = img
    response_json = {
        "choices": [{"text": "Answer from GPT-4 Vision"}]
    }
    mock_response = Mock()
    mock_response.json.return_value = response_json
    with patch(
        "requests.post", return_value=mock_response
    ) as mock_post:
        result = gpt_api(task, img_url)
        mock_post.assert_called_once()
    assert result == response_json


def test_call_method_with_exception(gpt_api):
    task = "What is in the image?"
    img_url = img
    with patch(
        "requests.post", side_effect=Exception("Test Exception")
    ):
        with pytest.raises(Exception):
            gpt_api(task, img_url)
```

```
@pytest.mark.asyncio
async def test_arun_success(vision_api):
    expected_response = {
        "choices": [
            {"message": {"content": "This is the model's response."}}
        ]
    }
    with patch(
        "aiohttp.ClientSession.post",
        new_callable=AsyncMock,
        return_value=AsyncMock(
            json=AsyncMock(return_value=expected_response)
        ),
    ) as mock_post:
        result = await vision_api.arun("What is this?", img)
        mock_post.assert_called_once()
        assert result == "This is the model's response."



@pytest.mark.asyncio
async def test_arun_request_error(vision_api):
    with patch(
        "aiohttp.ClientSession.post",
        new_callable=AsyncMock,
        side_effect=Exception("Request Error"),
    ):
```

```
        with pytest.raises(Exception):
            await vision_api.arun("What is this?", img)



def test_run_many_success(vision_api):
    expected_response = {
        "choices": [
            {"message": {"content": "This is the model's response."}}
        ]
    }
    with patch(
        "requests.post",
        return_value=Mock(json=lambda: expected_response),
    ) as mock_post:
        tasks = ["What is this?", "What is that?"]
        imgs = [img, img]
        results = vision_api.run_many(tasks, imgs)
        assert mock_post.call_count == 2
        assert results == [
            "This is the model's response.",
            "This is the model's response.",
        ]



def test_run_many_request_error(vision_api):
    with patch(
```

```
        "requests.post", side_effect=RequestException("Request Error")
    ):
        tasks = ["What is this?", "What is that?"]
        imgs = [img, img]
        with pytest.raises(RequestException):
            vision_api.run_many(tasks, imgs)


@pytest.mark.asyncio
async def test_arun_json_decode_error(vision_api):
    with patch(
        "aiohttp.ClientSession.post",
        new_callable=AsyncMock,
        return_value=AsyncMock(
            json=AsyncMock(side_effect=ValueError)
        ),
    ):
        with pytest.raises(ValueError):
            await vision_api.arun("What is this?", img)


@pytest.mark.asyncio
async def test_arun_api_error(vision_api):
    error_response = {"error": {"message": "API Error"}}
    with patch(
        "aiohttp.ClientSession.post",
```

```
        new_callable=AsyncMock,
        return_value=AsyncMock(
            json=AsyncMock(return_value=error_response)
        ),
    ):
        with pytest.raises(Exception, match="API Error"):
            await vision_api.arun("What is this?", img)


@pytest.mark.asyncio
async def test_arun_unexpected_response(vision_api):
    unexpected_response = {"unexpected": "response"}
    with patch(
        "aiohttp.ClientSession.post",
        new_callable=AsyncMock,
        return_value=AsyncMock(
            json=AsyncMock(return_value=unexpected_response)
        ),
    ):
        with pytest.raises(Exception, match="Unexpected response"):
            await vision_api.arun("What is this?", img)


@pytest.mark.asyncio
async def test_arun_retries(vision_api):
    with patch(
```

```
        "aiohttp.ClientSession.post",
        new_callable=AsyncMock,
        side_effect=ClientResponseError(None, None),
    ) as mock_post:
        with pytest.raises(ClientResponseError):
            await vision_api.arun("What is this?", img)
        assert mock_post.call_count == vision_api.retries + 1



@pytest.mark.asyncio
async def test_arun_timeout(vision_api):
    with patch(
        "aiohttp.ClientSession.post",
        new_callable=AsyncMock,
        side_effect=asyncio.TimeoutError,
    ):
        with pytest.raises(asyncio.TimeoutError):
            await vision_api.arun("What is this?", img)
```