```
from unittest.mock import patch

import pytest
from fastapi import FastAPI, Request
from fastapi.testclient import TestClient
from starlette.middleware.base import BaseHTTPMiddleware

from swarms_cloud.func_api_wrapper import SwarmCloud


# Create a fixture for an instance of SwarmCloud
@pytest.fixture
def func_api_wrapper():
    return SwarmCloud()


# Create a fixture for a mock of the uvicorn module
@pytest.fixture
def mock_uvicorn():
    with patch("swarms_cloud.func_api_wrapper.uvicorn") as mock:
        yield mock


def test_init(func_api_wrapper):
    assert func_api_wrapper.host == "0.0.0.0"
    assert func_api_wrapper.port == 8000
```

```
    assert isinstance(func_api_wrapper.app, FastAPI)


def test_add(func_api_wrapper):
    @func_api_wrapper.add("/test", method="get")
    def test_endpoint():
        return {"message": "test"}

    client = TestClient(func_api_wrapper.app)
    response = client.get("/test")
    assert response.status_code == 200
    assert response.json() == {"message": "test"}


def test_add_invalid_method(func_api_wrapper):
    with pytest.raises(ValueError):

        @func_api_wrapper.add("/test", method="invalid")
        def test_endpoint():
            return {"message": "test"}


def test_run(func_api_wrapper, mock_uvicorn):
    func_api_wrapper.run()
    mock_uvicorn.run.assert_called_once_with(
        func_api_wrapper.app, host="0.0.0.0", port=8000
```

```
    )


def test_call(func_api_wrapper, mock_uvicorn):
    func_api_wrapper()
    mock_uvicorn.run.assert_called_once_with(
        func_api_wrapper.app, host="0.0.0.0", port=8000
    )


def test_add_post(func_api_wrapper):
    @func_api_wrapper.add("/test_post", method="post")
    def test_post_endpoint():
        return {"message": "test_post"}

    client = TestClient(func_api_wrapper.app)
    response = client.post("/test_post")
    assert response.status_code == 200
    assert response.json() == {"message": "test_post"}


def test_add_put(func_api_wrapper):
    @func_api_wrapper.add("/test_put", method="put")
    def test_put_endpoint():
        return {"message": "test_put"}
```

```
    client = TestClient(func_api_wrapper.app)
    response = client.put("/test_put")
    assert response.status_code == 200
    assert response.json() == {"message": "test_put"}


def test_add_delete(func_api_wrapper):
    @func_api_wrapper.add("/test_delete", method="delete")
    def test_delete_endpoint():
        return {"message": "test_delete"}

    client = TestClient(func_api_wrapper.app)
    response = client.delete("/test_delete")
    assert response.status_code == 200
    assert response.json() == {"message": "test_delete"}


def test_error_handling(func_api_wrapper):
    @func_api_wrapper.add("/test_error", method="get")
    def test_error_endpoint():
        raise Exception("Test exception")

    client = TestClient(func_api_wrapper.app)
    response = client.get("/test_error")
    assert response.status_code == 500
    assert "Test exception" in response.text
```

```
def test_rate_limiting(func_api_wrapper):
    @func_api_wrapper.add("/test_rate_limit", method="get")
    def test_rate_limit_endpoint():
        return {"message": "test_rate_limit"}

    client = TestClient(func_api_wrapper.app)
    for _ in range(6):
        response = client.get("/test_rate_limit")
    assert response.status_code == 429
    assert "Too Many Requests" in response.text


# Custom middleware for testing
class CustomMiddleware(BaseHTTPMiddleware):
    async def dispatch(self, request, call_next):
        response = await call_next(request)
        response.headers["X-Custom-Header"] = "Test"
        return response


def test_add_patch(func_api_wrapper):
    @func_api_wrapper.add("/test_patch", method="patch")
    def test_patch_endpoint():
        return {"message": "test_patch"}
```

```
    client = TestClient(func_api_wrapper.app)
    response = client.patch("/test_patch")
    assert response.status_code == 200
    assert response.json() == {"message": "test_patch"}


def test_request_data(func_api_wrapper):
    @func_api_wrapper.add("/test_data", method="post")
    async def test_data_endpoint(request: Request):
        data = await request.json()
        return data

    client = TestClient(func_api_wrapper.app)
    response = client.post("/test_data", json={"key": "value"})
    assert response.status_code == 200
    assert response.json() == {"key": "value"}


def test_query_params(func_api_wrapper):
    @func_api_wrapper.add("/test_params", method="get")
    def test_params_endpoint(key: str):
        return {"key": key}

    client = TestClient(func_api_wrapper.app)
    response = client.get("/test_params?key=value")
```

```
    assert response.status_code == 200
    assert response.json() == {"key": "value"}


def test_custom_middleware(func_api_wrapper):
    func_api_wrapper.app.middleware("http")(CustomMiddleware)

    @func_api_wrapper.add("/test_middleware", method="get")
    def test_middleware_endpoint():
        return {"message": "test_middleware"}

    client = TestClient(func_api_wrapper.app)
    response = client.get("/test_middleware")
    assert response.status_code == 200
    assert response.headers["X-Custom-Header"] == "Test"


def test_add_endpoints(func_api_wrapper):
    endpoints = [
        ("/test1", "get", lambda: {"message": "test1"}),
        ("/test2", "post", lambda: {"message": "test2"}),
    ]
    func_api_wrapper.add_endpoints(endpoints)

    client = TestClient(func_api_wrapper.app)
    response = client.get("/test1")
```

```
    assert response.status_code == 200
    assert response.json() == {"message": "test1"}

    response = client.post("/test2")
    assert response.status_code == 200
    assert response.json() == {"message": "test2"}


def test_add_endpoints_invalid_method(func_api_wrapper):
    endpoints = [
        ("/test_invalid", "invalid", lambda: {"message": "test_invalid"}),
    ]
    with pytest.raises(ValueError):
        func_api_wrapper.add_endpoints(endpoints)


def test_add_endpoints_exception(func_api_wrapper):
    endpoints = [
        ("/test_exception", "get", lambda: 1 / 0),
    ]
    func_api_wrapper.add_endpoints(endpoints)

    client = TestClient(func_api_wrapper.app)
    response = client.get("/test_exception")
    assert response.status_code == 500
    assert "division by zero" in response.text
```