```
import pytest
from unittest.mock import Mock, patch
from fastapi import FastAPI, HTTPException
from time import time
import logging
from swarms_cloud.utils.rate_limiter import rate_limiter


class MockClient:
    def __init__(self, host):
        self.host = host


class MockFunction:
    def __call__(self, *args, **kwargs):
        return f"Mock function called with args: {args} and kwargs: {kwargs}"


@pytest.fixture
def mock_time():
    with patch("time.time", Mock()) as mock_time:
        yield mock_time


@pytest.mark.parametrize(
    "max_requests, time_span",
```

```
    [(1, 10), (5, 1), (10, 60)],
)
def test_rate_limiter_allows_requests_within_limits(max_requests, time_span):
    # Arrange
    app = FastAPI()
    function = MockFunction()
    client1 = MockClient("client1")
    client2 = MockClient("client2")

    # Act
    @rate_limiter(max_requests, time_span)
    async def decorated_function(*args, **kwargs):
        return await function(*args, **kwargs)

    app.get("/")(decorated_function)

    # Assert
    for _ in range(max_requests):
        response = app.get("/")(client=client1)
        assert response.status_code == 200
        assert response.json() == "Mock function called with args: () and kwargs: {}"

    with pytest.raises(HTTPException) as error:
        app.get("/")(client=client1)

    assert error.value.status_code == 429
```

```
    assert error.value.detail == "Too many requests"

    # Check if limit resets after time_span
    time.sleep(time_span + 1)
    response = app.get("/")(client=client1)
    assert response.status_code == 200

    # Verify different clients are treated separately
    response = app.get("/")(client=client2)
    assert response.status_code == 200


def test_rate_limiter_logs_warnings(max_requests: int, time_span: int):
    # Arrange
    app = FastAPI()
    function = MockFunction()
    client = MockClient("client")

    # Act
    with patch.object(logging.getLogger(__name__), "warning") as mock_logger:

        @rate_limiter(max_requests, time_span)
        async def decorated_function(*args, **kwargs):
            return await function(*args, **kwargs)

        app.get("/")(decorated_function)
```

```
        for _ in range(max_requests + 1):
            app.get("/")(client=client)

    # Assert
    assert mock_logger.call_count == 1
    assert mock_logger.call_args[0][0] == f"Rate limit exceeded for IP: {client.host}"



def test_rate_limiter_catches_exceptions(max_requests: int, time_span: int, mock_time):
    # Arrange
    app = FastAPI()
    MockFunction()
    client = MockClient("client")

    # Act
    @rate_limiter(max_requests, time_span)
    async def decorated_function(*args, **kwargs):
        raise Exception("Test exception")

    app.get("/")(decorated_function)

    # Assert
    with pytest.raises(Exception) as error:
        app.get("/")(client=client)
```

```
assert error.value.args[0] == "Test exception"
```