pytest-remote-response

Release 2.1.2

devanshshukla99

Apr 27, 2023

CONTENTS

CHAPTER

ONE

GETTING STARTED

1.1 Installation

1.1.1 PyPI

```
$ pip install pytest-remote-response
```

1.1.2 GitHub

```
$ git clone https://github.com/devanshshukla99/pytest-remote-response
$ cd pytest-remote-response
$ pip install .
```

or

```
$ pip install -e git+git://github.com/devanshshukla99/pytest-remote-response.git
↪#egg=pytest-remote-response
```

1.2 Usage

Table 1: Supported interceptors:

Library	Interceptors
urllib	*urllib* *_urllib*
urllib3	*urllib3* *_urllib3*
requests	*requests*
aiohttp	*aiohttp*

1.2.1 pytest plugin

Once, installed the plugin will register automatically with pytest with its configuration options available via `pytest --help`.

```
$ pytest --help
```

The plugin is only applicable to function wrapped with the *pytest_response.app.Response.activate()* decorator.

Handling connection requests

- **Block remote connections:**

```
$ pytest --remote-block
```

- **Capture remote requests:**

```
$ pytest --remote-capture
```

- **Mock remote requests:**

```
$ pytest --remote-response
```

- **Database dump:**

The database dump file can be specified using the `remote_response_database` ini-config option with pytest. These config options can be set through `pytest.ini`, `pyproject.toml`, `tox.ini` or `setup.cfg`. Follow the pytest_config docs for more info.

Examples

```
$ pytest --remote-block
$ pytest --remote-capture
$ pytest --remote-response
```

1.2.2 Standalone package

The tools implemented in this package can be easily ported to any other application, with mimial config required:

Basic usage:

```
from pytest_response import response

# Setup the database file
response.setup_database("database.json")

# Block outgoing connections
response.configure(remote=False, capture=False, response=False)
```

(continues on next page)

(continued from previous page)

```
# Capture outgoing connections
response.configure(remote=True, capture=True, response=False)

# Mock outgoing connections
response.configure(remote=True, capture=False, response=True)

# Applies the interceptor
response.post({INTERCEPTOR})

...
...

# Cleanup
response.unpost()
```

Note: Here {INTERCEPTORS} can be *str* or *list*

Examples:

Block connections in urllib

```
# Block outgoing connections for `urllib` library

from pytest_response import response

# Setting up a clean database
response.setup_database("database.json")

# Block outgoing connections
response.configure(remote=False, capture=False, response=False)

# Applies the `urllib` interceptor
response.post("urllib")

# It's important to import the function after setting up `response`,
# to make sure the monkey patched version is used
from urllib.request import urlopen  # noqa

# Since the connections are blocked, this request will
# error out with :class:`~pytest_response.exceptions.RemoteBlockedError`
r = urlopen("https://www.python.org")

# Cleanup
response.unpost()
```

Capture connections in requests

```
# Capture outgoing connections for `requests` library

from pytest_response import response

# Setting up a clean database
response.setup_database("database.json")

# Capture outgoing connections
response.configure(remote=True, capture=True, response=False)

# Applies the `requests` interceptor
response.post("requests")

# It's important to import the function after setting up `response`,
# to make sure the monkey patched version is used
import requests  # noqa

# Since the interceptors are in capture mode, the response data and headers
# will be dumped in the database file, i.e. database.json
r = requests.get("https://www.python.org")

# Cleanup
response.unpost()
```

Mock connections in urllib3

```
# Mock outgoing connections for `urllib3` library

from pytest_response import response

# Spoof outgoing connections
response.configure(remote=True, capture=False, response=True)

# Applies the `urllib3` interceptor
response.post("urllib3")

# It's important to import the function after setting up `response`,
# to make sure the monkey patched version is used
import urllib3  # noqa

http = urllib3.PoolManager()
url = "https://www.python.org"

# Since the interceptors are in response mode, the response data and headers
# will be spoofed with saved data in the database;
# if the query comes back empty, this request will
# error out with :class:`pytest_response.exceptions.ResponseNotFound`
res = http.request("GET", url)
assert res.status == 200
```

(continues on next page)

(continued from previous page)

```
# Cleanup
response.unpost()
```

Mock connections in urllib3 using decorator

```
import urllib3  # noqa

from pytest_response import response

response.configure(remote=True, capture=True, response=False)


@response.activate("urllib3")
def get_url():
    http = urllib3.PoolManager()
    url = "https://www.python.org"

    # Since the interceptors are in response mode, the response data and headers
    # will be spoofed with saved data in the database;
    # if the query comes back empty, this request will
    # error out with :class:`pytest_response.exceptions.ResponseNotFound`
    res = http.request("GET", url)
    assert res.status == 200
    assert res.data


get_url()
```

Using `ResponseDB`

```
# Import the database interface :class:`pytest_response.database.ResponseDB`
from base64 import b64encode

from pytest_response.database import ResponseDB

# Setup the database
db = ResponseDB(path="database.json")

# Insert new element
url = "https://www.python.org"
status = 200
headers = {}
data = b""
db.insert(url=url, status=status, response=data, headers=headers)

# Verify the index
assert b64encode(url.encode()).decode() in db.index()
```

(continues on next page)

(continued from previous page)

```
# Query for an URL
url = "https://www.python.org"
get_status, get_data, get_headers = db.get(url)

# Verify the data is unchanged
assert status == get_status
assert data == get_data
assert headers == get_headers
```

CHAPTER

TWO

WHY PYTEST-REMOTE-RESPONSE?

Nowadays, many of our tests fetch some data or are in some way connected to an online service.

But, what if the service is down? or modified its response?

? It's a lot more challenging to ascertain the fault in such cases.

This package attempts to mock some external requests to make the tests more self-contained and reduce the requests to the outside world!

Perhaps solve some HTTP 429's too.

Note: Due to some limitations, it is not advisible to run the tests in completely offline-environments.

Get in touch if interested in testing such cases!

CHAPTER

THREE

HOW IT WORKS?

pytest-remote-response works on top of interceptors specific for each library.

In a nutshell, interceptors wrap the library calls, thereby attaching additional mock/capture logic.

Note: For some nerds, interceptors uses `MonkeyPatch` to monkey patch the original library with a wrapped one.

This approach certainly has some benefits and limitations; for instance, it's surprisingly easy to write new interceptors as it's only a wrapper method but it's only applicable when that method is called explicitly; for countering this, *pytest-remote-response* ships with two more deep interceptors *_urllib* and *_urllib3*.

Warning: *_urllib* and *_urllib3* are more low-level but are plagued with threading issues; use them carefully!

3.1 Interceptor

Note: Any custom intercept must have at-least `install` and `uninstall` methods and should return a *BaseMockResponse*

3.1.1 Example

```
from functools import warps

# Import `response` instance to get access to :class:`~pytest.MonkeyPatch` and
# :class:`~pytest_response.database.ResponseDB`
from pytest_response import response

# Library wrapper
def wrapper(func):
    @wraps(func)
    def inner_func(url, *args, **kwargs):
        # Check if response is `True`, if `True` spoof the request from the database
        if response.response:
            status, data, headers = response.get(url=url)
            return MockResponse(status, data, headers)
```

(continues on next page)

(continued from previous page)

```
        # Actual library call
        lib_response = func(url, *args, **kwargs)

        # Check if capture is `True`, if `True` then extract and dump the response into␣
↪the database.
        if not response.capture:
            return _

        # Extract data, headers and status from `lib_response`

        response.insert(url=url, response=data, headers=dict(headers), status=_.status)
        return _
    return inner_func

class MockResponse(BaseMockResponse):
    # A basic Mock Response for spoofing data, headers and status
    def __init__(self, status, data, headers={}):
        super().__init__(status, data, headers)
    pass


def install():
    # A basic install method for Monkey Patching the library
    lib_call = library.call
    wrapped_lib_call = urlopen_wrapper(lib_call)
    response.mpatch.setattr("library.call", wrapped_lib_call)
    return


def uninstall():
    # A basic uninstall method
    response.mpatch.undo()
```

3.2 Database

`pytest-remote-response` has *ResponseDB* to facilitate talking to the database. Internally, its actually a wrapper for `sqlite3`.

Primary communication methods are *insert()* and *get()*.

Note: Some fields such as `headers` and `data` are compressed using `zlib` to reduce the over-all footprint.

Table 1: Database fields

Fields	Dumped as
url	`str` – `base64.b64encode()` serialized
cache_date	`str`
status	`str`
headers	`str` – `base64.b64encode()` serialized after `zlib.compress()`
response	`str` – `base64.b64encode()` serialized after `zlib.compress()`

3.2.1 Example

```
# Import the database interface :class:`pytest_response.database.ResponseDB`
from base64 import b64encode

from pytest_response.database import ResponseDB

# Setup the database
db = ResponseDB(path="database.json")

# Insert new element
url = "https://www.python.org"
status = 200
headers = {}
data = b""
db.insert(url=url, status=status, response=data, headers=headers)

# Verify the index
assert b64encode(url.encode()).decode() in db.index()

# Query for an URL
url = "https://www.python.org"
get_status, get_data, get_headers = db.get(url)

# Verify the data is unchanged
assert status == get_status
assert data == get_data
assert headers == get_headers
```

CHAPTER

FOUR

API DOCUMENTATION

4.1 pytest plugin

pytest_response.plugin.**pytest_addoption**(*parser*)
Pytest hook for adding cmd-line options.

Adds relevent cmd-line and ini-config options.

pytest_response.plugin.**pytest_configure**(*config*)
Pytest hook for setting up *pytest_response.app.Response*

pytest_response.plugin.**pytest_unconfigure**(*config*)
Pytest hook for cleaning up.

4.2 Response

class pytest_response.app.**BaseMockResponse**(*status: int*, *data: bytes*, *headers: dict = {}*)
Bases: object

Basic response for mocking requests.

Parameters

- **status** (*int*) – Status code of the response.
- **data** (*bytes*) – Response data.
- **headers** (*dict*, optional) – Default to *{}*.

close() → None

flush()

getcode() → int

info() → dict

read(**args*, ***kwargs*) → bytes
Wrapper for _io.BytesIO.read

readinto(**args*, ***kwargs*) → bytes
Wrapper for _io.BytesIO.readinto

readline(**args*, ***kwargs*) → bytes
Wrapper for _io.BytesIO.readline

class pytest_response.app.**Response**(*path: str = 'interceptors'*, *capture: bool = False*, *remote: bool = False*, *response: bool = False*, *log_level: str = 'debug'*, *database: str = 'database.db'*)

Bases: object

Controlling and configuration application for `pytest-remote-response`

Parameters

- **path** (*str*, optional) – Path for the interceptors. Defaults to *pytest_response.interceptors*
- **capture** (*bool*, optional) – if *True* captures data and headers in the database. Defaults to *False*
- **remote** (*bool*, optional) – if *False* blocks connection requests. Defaults to *False*
- **response** (*bool*, optional) – if *True* responds with data and headers from the database. Defaults to *False*
- **log_level** (*str*, optional) – Log level. Defaults to *debug*

Examples

```
>>> from pytest_response import response
>>> response.setup_database({{ Path to the database}})
>>> response.post({{ Interceptor }})
>>> ...
>>> response.unpost()
```

activate(*interceptors: Union[str, list]*)

Wrapper to apply the interceptor decorator.

Parameters **interceptor** (*str*, *list*) – interceptors to apply

Examples

```
>>> @response.activate("urllib_quick")
>>> def test_urllib():
>>>     url = "https://www.python.org"
>>>     r = urlopen(url)
>>>     assert r.status == 200
```

apply(*mock: Optional[Union[str, list]] = None*) → bool

Activates intercepter module provided in *mock* otherwise activates all.

Parameters **mock** (*str*, optional) – Applies the mock.

property available: List[str]

property capture: bool

configure(*remote: bool = False*, *capture: bool = False*, *response: bool = False*) → None

Helper method for configuring interceptors.

Parameters

- **remote** (*bool*, optional) – If *False* blocks connection requests. Defaults to *False*.
- **capture** (*bool*, optional) – If *True* captures data and headers in the database. Defaults to *False*.

- **response** (*bool*, optional) – If *True* responds with data and headers from the database. Defaults to *False*.

get(*url: str*, **args*, ***kwargs*)
Wrapper function for *pytest_response.database.ResponseDB.get()*

Parameters **url** (*str*) – URL to be queried.

Returns

- **status** (*int*) – Status code
- **data** (*bytes*) – Response data.
- **headers** (*dict*) – Response header.

insert(*url: str*, *response: bytes*, *headers: str*, *status: int*, **args*, ***kwargs*) → bool
Wrapper function for *pytest_response.database.ResponseDB.insert()*

Parameters

- **url** (*str*) – URL of the dump.
- **response** (*bytes*) – Data captured.
- **headers** (*str*) – Headers captured.
- **status** (*int*) – Status code of the response.
- ****kwargs** (*dict*) – Any additional parameter to be dumped.

post(*mock: Union[str, list]*) → None
Registers and applies the mock under the same hood.

Internally uses *Response.register()* followed by *Response.apply()*

Parameters **mock** (*str*) – Registers and applies the mock.

register(*mock: Union[str, list]*) → None
Registers interceptor modules; applies using *Response.apply()*

Parameters **mock** (*str*, *list*) – Interceptor; check *Response.available()* for more info.

registered() → Dict[str, module]
Returns registered modules.

Returns Returns the list of registered interceptors.

Return type *list* of *pathlib.Path*

property remote: bool

property response: bool

setup_database(*path: str*) → bool
Method to setup-up database.

Parameters **path** (*str*) – Path for the database.

unapply(**args*, ***kwargs*) → bool
Un-applies interceptor modules.

unpost() → bool
Unapplied and unregisters mocks under the same hood.

Internally uses *Response.unapply()* followed by *Response.unregister()*

unregister() → bool
Deactivates interceptor modules.

4.3 Database

class pytest_response.database.**ResponseDB**(*path: str*)
Bases: object

Wrapper class for sqlite3

Parameters **path** (*str*) – Path for the database

Examples

```
>>> db = ResponseDB("database.db")
```

all() → dict
Method to return all records in the database.

Returns

Return type Return list of all elements.

close() → None

get(*url: str*, ***kwargs*) → Tuple[int, bytes, dict]
Method for getting response and header for a perticular query *url*. Currently working by locating *url* only; multi-query to be implemented later.

Parameters **url** (*str*) – URL to be queried.

Returns

- **status** (*int*) – Status code
- **data** (*bytes*) – Response data.
- **headers** (*dict*) – Response header.

index(*index: Optional[str] = 'url'*) → List[str]
Returns all occurances of the column *index*.

Parameters **index** (*str*) – Defaults to *url*.

Returns **_occurances** – All occurances of the selected column *index*.

Return type *list*

insert(*url: str*, *response: bytes*, *headers: dict*, *status: Optional[int] = 200*, ***kwargs*) → None
Method for dumping url, headers and responses to the database. All additonal kwargs are dumped as well.

Parameters

- **url** (*str*) – URL of the dump.
- **response** (*bytes*) – Data captured.
- **headers** (*str*) – Headers captured.
- **status** (*int*) – Status code of the response.

- **kwargs** (*dict*) – Any additional parameter to be dumped.

setup() → bool
Function to setup the database table.

today = '2023-04-27'

truncate() → bool
Method to purge all records in the database.

4.4 Interceptors

4.4.1 *urllib* Interceptor module

Functions

urlopen_wrapper(func)	Wrapper for urllib.request
install()	Method to monkey patch the library call with the wrapped one.
uninstall()	Method to undo all monkey patches.

urlopen_wrapper

pytest_response.interceptors.urllib.**urlopen_wrapper**(*func*)
Wrapper for urllib.request

install

pytest_response.interceptors.urllib.**install**()
Method to monkey patch the library call with the wrapped one.

uninstall

pytest_response.interceptors.urllib.**uninstall**()
Method to undo all monkey patches.

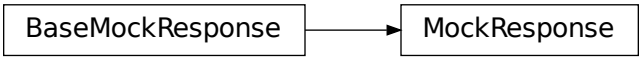
Classes

MockResponse(status, data[, headers])	

MockResponse

class pytest_response.interceptors.urllib.**MockResponse**(*status*, *data*, *headers={}*)
Bases: *pytest_response.app.BaseMockResponse*

Class Inheritance Diagram



4.4.2 *urllib* Interceptor module (*unstable*)

Functions

install()	
uninstall()	

install

pytest_response.interceptors._urllib.**install**()

uninstall

pytest_response.interceptors._urllib.**uninstall**()

Classes

ResponseSocketIO(sock, mode)	Provides a method to write the value of the buffer into another var for dumping.
ResponseSocket(host, port, *args, **kwargs)	Socket implementation of Pytest-Response
Response_SSLSocket(*args, **kwargs)	SSLSocket implementation of Pytest-Response
ResponseHTTPConnection(host[, port, ...])	Wrapper for *~http.client.HTTPConnection*
ResponseHTTPHandler([debuglevel])	Override the default HTTPHandler class with one that uses the ResponseHTTPConnection class to open HTTP URLs.
ResponseHTTPResponse(sock[, debuglevel, ...])	Provides a way to capture or respond with a saved response.

continues on next page

Table 4 – continued from previous page

ResponseHTTPSConnection(host[, port, ...])	Override the default *~HTTPSConnection* to use *~ResponseSocket*
ResponseHTTPSHandler([debuglevel, context, ...])	Override the default HTTPSHandler class with one that uses the ResponseHTTPSConnection class to open HTTP URLs.

ResponseSocketIO

class pytest_response.interceptors._urllib.**ResponseSocketIO**(*sock*, *mode*)

Bases: socket.SocketIO

Provides a method to write the value of the buffer into another var for dumping. Wrapper for *~socket.SocketIO*.

Methods Summary

readinto(b)	Wrapper function for *socket.SocketIO.readinto*

Methods Documentation

readinto(*b*)

Wrapper function for *socket.SocketIO.readinto*

ResponseSocket

class pytest_response.interceptors._urllib.**ResponseSocket**(*host*, *port*, **args*, ***kwargs*)

Bases: _socket.socket

Socket implementation of Pytest-Response

Provides *ResponseSocket.makefile* method to return a buffer built with *ResponseSocketIO*

Methods Summary

close()	Close the socket.
connect(*args, **kwargs)	Connects to host in capturing mode otherwise passes.
makefile([mode, buffering, encoding, ...])	Provides makefile() method which returns a Buffered IO built with *ResponseSocketIO*
sendall(data, *args, **kwargs)	Wrapper for *_socket.socket.sendall*

Methods Documentation

close()
Close the socket. It cannot be used after this call.

connect(**args*, ***kwargs*)
Connects to host in capturing mode otherwise passes.

Wrapper for *_socket.socket.connect*

makefile(*mode='r'*, *buffering=None*, *encoding=None*, *errors=None*, *newline=None*, **args*, ***kwargs*)
Provides makefile() method which returns a Buffered IO built with *ResponseSocketIO*

sendall(*data*, **args*, ***kwargs*)
Wrapper for *_socket.socket.sendall*

Response_SSLSocket

class pytest_response.interceptors._urllib.**Response_SSLSocket**(**args*, ***kwargs*)
Bases: ssl.SSLSocket

SSLSocket implementation of Pytest-Response

Provides a wrapper *recv_into* for capturing the response.

ResponseHTTPConnection

class pytest_response.interceptors._urllib.**ResponseHTTPConnection**(*host*, *port=None*, *timeout=<object object>*, *source_address=None*, *blocksize=8192*)
Bases: http.client.HTTPConnection

Wrapper for *~http.client.HTTPConnection*

Methods Summary

connect()	Override the connect() function to intercept calls.
request(method, url[, body, headers, ...])	Send a complete request to the server.

Methods Documentation

connect()
Override the connect() function to intercept calls.

request(*method*, *url*, *body=None*, *headers={}*, ***, *encode_chunked=False*)
Send a complete request to the server.

ResponseHTTPHandler

class pytest_response.interceptors._urllib.**ResponseHTTPHandler**(*debuglevel=0*)

Bases: urllib.request.HTTPHandler

Override the default HTTPHandler class with one that uses the ResponseHTTPConnection class to open HTTP URLs.

Methods Summary

http_open(req)

Methods Documentation

http_open(*req*)

ResponseHTTPResponse

class pytest_response.interceptors._urllib.**ResponseHTTPResponse**(*sock*, *debuglevel=0*, *method=None*, *headers=None*)

Bases: http.client.HTTPResponse

Provides a way to capture or respond with a saved response.

Methods Summary

begin(*args, **kwargs)

Methods Documentation

begin(**args*, ***kwargs*)

ResponseHTTPSConnection

class pytest_response.interceptors._urllib.**ResponseHTTPSConnection**(*host*, *port=None*, *key_file=None*, *cert_file=None*, *timeout=<object object>*, *source_address=None*, ***, *context=None*, *check_hostname=None*, *blocksize=8192*)

Bases: http.client.HTTPSConnection, *pytest_response.interceptors._urllib.ResponseHTTPConnection*

Override the default *~HTTPSConnection* to use *~ResponseSocket*

Methods Summary

connect()	Connect to a host on a given (SSL) port.
request(method, url[, body, headers, ...])	Send a complete request to the server.

Methods Documentation

connect()
Connect to a host on a given (SSL) port.

request(*method*, *url*, *body=None*, *headers={}*, ***, *encode_chunked=False*)
Send a complete request to the server.

ResponseHTTPSHandler

class pytest_response.interceptors._urllib.**ResponseHTTPSHandler**(*debuglevel=0*, *context=None*, *check_hostname=None*)

Bases: urllib.request.HTTPSHandler

Override the default HTTPSHandler class with one that uses the ResponseHTTPSConnection class to open HTTP URLs.

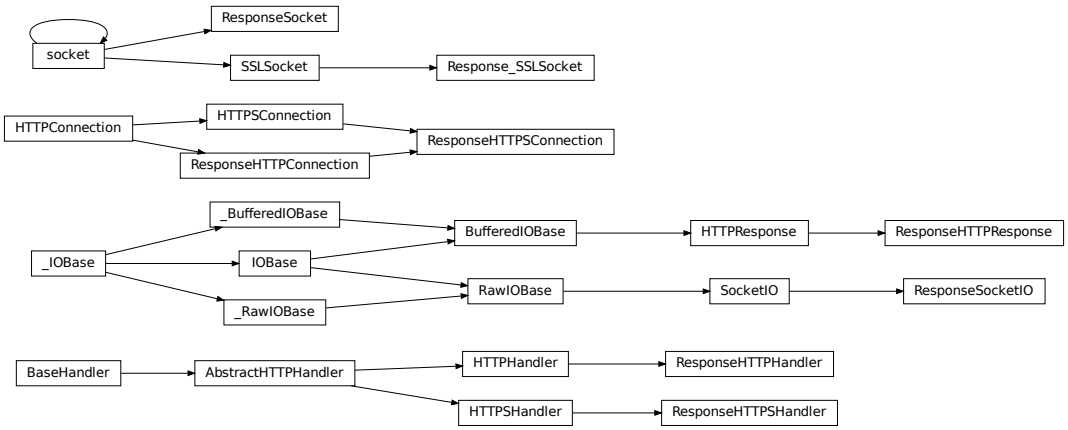
Methods Summary

https_open(req)

Methods Documentation

https_open(*req*)

Class Inheritance Diagram



4.4.3 *urllib3* Interceptor module

Functions

urlopen_wrapper(func)	Wrapper for urllib3.HTTPConnectionPool.urlopen()
install()	Method to monkey patch the library call with the wrapped one.
uninstall()	Method to undo all monkey patches.

urlopen_wrapper

pytest_response.interceptors.urllib3.**urlopen_wrapper**(*func*)

Wrapper for urllib3.HTTPConnectionPool.urlopen()

install

pytest_response.interceptors.urllib3.**install**()
Method to monkey patch the library call with the wrapped one.

uninstall

pytest_response.interceptors.urllib3.**uninstall**()
Method to undo all monkey patches.

Classes

MockResponse(status, data[, headers])	

MockResponse

class pytest_response.interceptors.urllib3.**MockResponse**(*status*, *data*, *headers={}*)
Bases: *pytest_response.app.BaseMockResponse*

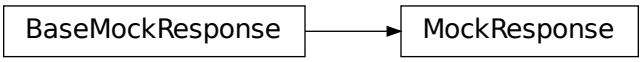
Methods Summary

get_redirect_location(*args, **kwargs)	

Methods Documentation

get_redirect_location(**args*, ***kwargs*)

Class Inheritance Diagram



4.4.4 *urllib3* Interceptor module (*unstable*)

Functions

install()	
uninstall()	

install

pytest_response.interceptors._urllib3.**install**()

uninstall

pytest_response.interceptors._urllib3.**uninstall**()

Classes

Response_HTTPU3_Intercepter(*args, **kwargs)	
Response_HTTPSU3_Intercepter(*args, **kwargs)	

Response_HTTPU3_Intercepter

class pytest_response.interceptors._urllib3.**Response_HTTPU3_Intercepter**(**args*, ***kwargs*)

Bases: *pytest_response.interceptors._urllib.ResponseHTTPConnection*, urllib3.connection.HTTPConnection

Response_HTTPSU3_Intercepter

class pytest_response.interceptors._urllib3.**Response_HTTPSU3_Intercepter**(**args*, ***kwargs*)

Bases: *pytest_response.interceptors._urllib.ResponseHTTPSConnection*, urllib3.connection.HTTPSConnection

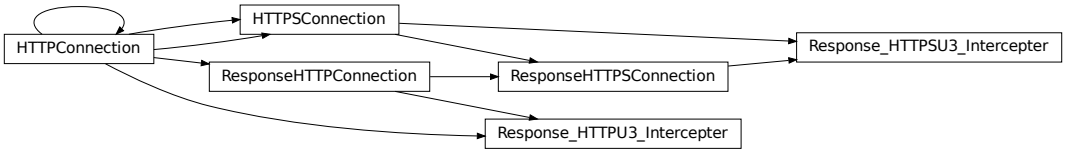
Attributes Summary

is_verified	Whether this connection verifies the host's certificate.

Attributes Documentation

is_verified = True
Whether this connection verifies the host's certificate.

Class Inheritance Diagram



4.4.5 *requests* Interceptor Module

Functions

requests_wrapper(func)	Wrapper for `requests.get()`
install()	Method to monkey patch the library call with the wrapped one.
uninstall()	Method to undo all monkey patches.

requests_wrapper

pytest_response.interceptors.requests.**requests_wrapper**(*func*)
Wrapper for `requests.get()`

install

pytest_response.interceptors.requests.**install**()
Method to monkey patch the library call with the wrapped one.

uninstall

pytest_response.interceptors.requests.**uninstall**()
Method to undo all monkey patches.

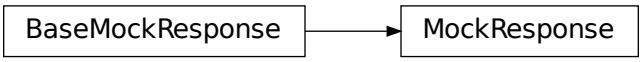
Classes

MockResponse(status, data[, headers])	

MockResponse

class pytest_response.interceptors.requests.**MockResponse**(*status*, *data*, *headers={}*)
Bases: *pytest_response.app.BaseMockResponse*

Class Inheritance Diagram



4.4.6 *aiohttp* Interceptor Module

Functions

create_wrapper(func)	Wrapper for `aiohttp.ClientResponse.text()`
get_wrapper(func)	Wrapper for `aiohttp.ClientSession.get()`
install()	Method to monkey patch the library call with the wrapped one.
uninstall()	Method to undo all monkey patches.

create_wrapper

pytest_response.interceptors.aiohttp.**create_wrapper**(*func*)
Wrapper for aiohttp.ClientResponse.text()

get_wrapper

pytest_response.interceptors.aiohttp.**get_wrapper**(*func*)
Wrapper for aiohttp.ClientSession.get()

install

pytest_response.interceptors.aiohttp.**install**()
Method to monkey patch the library call with the wrapped one.

uninstall

pytest_response.interceptors.aiohttp.**uninstall**()
Method to undo all monkey patches.

Classes

MockResponse(status, data[, headers])

MockResponse

class pytest_response.interceptors.aiohttp.**MockResponse**(*status*, *data*, *headers={}*)
Bases: *pytest_response.app.BaseMockResponse*

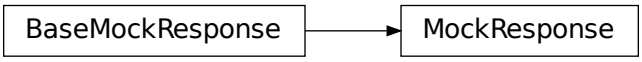
Methods Summary

text()

Methods Documentation

async text()

Class Inheritance Diagram



4.5 Exceptions

RemoteBlockedError([reason])	Exception raised when Remote connections are blocked.
ResponseNotFound([reason])	Exception raised when response is not locally available.
MalformedUrl([reason])	Exception raised when a malformed URL is encountered.
InterceptorNotFound([reason])	Exception raised when the requested interceptor is not available.
DatabaseNotFound([reason])	Exception raised when database is not initialized properly.

4.5.1 RemoteBlockedError

exception pytest_response.exceptions.**RemoteBlockedError**(*reason='A test tried to connect to internet.'*, **args*, ***kwargs*)

Exception raised when Remote connections are blocked.

4.5.2 ResponseNotFound

exception pytest_response.exceptions.**ResponseNotFound**(*reason='Response is not available; try capturing first.'*, **args*, ***kwargs*)

Exception raised when response is not locally available.

4.5.3 MalformedUrl

exception pytest_response.exceptions.**MalformedUrl**(*reason='Malformed URL encountered'*, **args*, ***kwargs*)

Exception raised when a malformed URL is encountered.

4.5.4 InterceptorNotFound

exception pytest_response.exceptions.**InterceptorNotFound**(*reason='Interceptor not available; check `Response.available `'*, **args*, ***kwargs*)

Exception raised when the requested interceptor is not available.

4.5.5 DatabaseNotFound

exception pytest_response.exceptions.**DatabaseNotFound**(*reason='Database not initialized; use ``Response.setup_database ``'*, **args*, ***kwargs*)

Exception raised when database is not initialized properly.

CHAPTER

FIVE

CHANGELOG

5.1 2.1 (2023-03-27)

5.1.1 Trivial/Internal Changes

- Migrating to towncrier for changelogs and added a workflow to check changelog entry for consistency in pull requests. (#33)

5.2 2.0.0 (2022-06-04)

5.2.1 Added/Improved Documentation

- Added documentation and examples for `activate()`. (#16)

5.2.2 Backwards Incompatible Changes

- Renamed `urllib_full` to `_urllib` and `urllib3_full` to `_urllib3`, to indicate their threading instability. (#16)
- Moving away from commandline arguments `--remote-db` / `--remote-database` in favour of ini-config option `remote_response_database` set through pytest with default value `database.db`. (#30)

5.2.3 Deprecations and Removals

- Ability to activate interceptors via command-line argument `--remote={INTERCEPTOR}` has been removed in favour of activating via `activate()` decorator. (#16)

5.2.4 Features

- pytest-remote-response now has a decorator `activate()` to apply interceptors on individual functions. It supports string, regex pattern or list of interceptors as argument. (#16)

5.2.5 Bug Fixes

- Fixed the documentation examples. (#26)

5.2.6 Trivial/Internal Changes

- Moving from `tinydb` to `sqlite3` for better threading support and reliability; no changes in public API. (#28)

5.3 1.0.0 (2021-06-30)

5.3.1 Bug Fixes

- Fixed *pytest_response.app.Response* and added another exception *DatabaseNotFound*. (#13)

5.3.2 Added/Improved Documentation

- More simplified `README.rst` page. (#7)
- Added more informative doc-strings. (#10)
- Added a supported clients list in the documentation. (#18)
- Documentation now hosts *Getting started*, *Why pytest-remote-response?* and *How it works?* pages! (#21)
- Added documentation and example usage of *ResponseDB* in the *How it works?* page. (#23)

5.3.3 Backwards Incompatible Changes

- Renamed package from `pytest-response` to `pytest-remote-response`. (#9)
- Renamed interceptors to a more clear norm. (#15)

5.3.4 Features

- Instead of moving through entire pipleline, now the interceptors will return a *pytest_response.app.BaseMockResponse* directly. (#1)
- Added a *pytest_response.app.Response.configure()* method for setting values of remote, capture and response. (#3)
- Added interceptor for `aiohttp` library. (#4)

5.3.5 Trivial/Internal Changes

- Similified `pytest_response.logger.log` use. (#3)
- Now the *pytest_response.database.ResponseDB.get()* method will automatically rstrip "/", useful in comparing URLs. (#4)
- Now `status` code will also be dumped/responded with along with `data` and `headers`. (#6)
- Simplified GitHub actions as a two step process. (#8)

CHAPTER

SIX

LICENSE

```
MIT License

Copyright (c) 2023 Devansh Shukla

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

CHAPTER

SEVEN

PYTEST-REMOTE-RESPONSE

This package provides a plugin for pytest framework for capturing and mocking connection requests during the test run.

Inspired by pook and pytest-responses.

Get started using the documentation and getting-started.

7.1 Installation

```
$ pip install pytest-remote-response
```

or

```
$ git clone https://github.com/devanshshukla99/pytest-remote-response
$ cd pytest-remote-response
$ pip install .
```

The plugin will register automatically with `pytest` framework and will be ready to use.

7.2 Supported Clients

Currently, *pytest-remote-response* supports,

- ✓ urllib
- ✓ urllib3
- ✓ requests
- ✓ aiohttp

7.3 Usage

7.3.1 Pytest plugin

The plugin works by applying monkeypatches of interceptors for different libraries using a wrapper `response.activate`. The interceptors when applied can capture, prevent or mock the connection request.

The available interceptors are listed in `response.available` method.

Example of using the decorator:

```
import urllib3
from pytest_response import response

response.configure(remote=True, capture=True, response=False)

@response.activate("urllib3")
def get_url():
    http = urllib3.PoolManager()
    url = "https://www.python.org"

    # Since the interceptors are in response mode, the response data and headers
    # will be spoofed with saved data in the database;
    # if the query comes back empty, this request will
    # error out with :class:`pytest_response.exceptions.ResponseNotFound`
    res = http.request("GET", url)
    assert res.status == 200
    assert res.data
```

Handling requests:

- **Block remote requests:** all requests are allowed by default; one can disable them using `--remote-block` flag

```
$ pytest --remote-block
```

- **Capture remote requests:** the requests can be captured in a `sqlite3` database using `--remote-capture` arg

```
$ pytest --remote-capture
```

- **Mock remote requests:** the requests can be mocked using `--remote-response`

```
$ pytest --remote-response
```

7.3.2 Standalone package

The tools implemented in this package can be easily ported to any other application, with mimial config required.

Configuration:

```
from pytest_response import response

response.setup_database({DUMP FILE})
response.post({INTERCEPTOR})
...
response.unpost()
```

7.4 Testing

Use `tox` to make sure the plugin is working:

```
$ git clone https://github.com/devanshshukla99/pytest-remote-response
$ cd pytest-remote-response
$ tox -e py38
```

See tox for more info.

7.5 Licence

This plugin is licenced under a MIT licence - see the `LICENCE` file.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

P

R

S

T

U