Package: httptest2 (via r-universe)

September 14, 2024

Type Package

Title Test Helpers for 'httr2'

Description Testing and documenting code that communicates with remote servers can be painful. This package helps with writing tests for packages that use 'httr2'. It enables testing all of the logic on the R sides of the API without requiring access to the remote service, and it also allows recording real API responses to use as test fixtures. The ability to save responses and load them offline also enables writing vignettes and other dynamic documents that can be distributed without access to a live server.

Version 1.1.0.9000

URL https://enpiar.com/httptest2/, https://github.com/nealrichardson/httptest2

BugReports https://github.com/nealrichardson/httptest2/issues

License MIT + file LICENSE

Imports digest, httr2, jsonlite, rlang, stats, testthat, utils

Suggests curl, knitr, pkgload, rmarkdown, spelling, webfakes, xml2

Language en-US

RoxygenNote 7.3.1

Roxygen list(markdown = TRUE)

Encoding UTF-8

VignetteBuilder knitr

Config/testthat/edition 3

Repository https://nealrichardson.r-universe.dev

RemoteUrl https://github.com/nealrichardson/httptest2

RemoteRef HEAD

RemoteSha 5856ed57a5985b9ffede754601fe2bd501a19b09

Contents

`.mockPaths` *Set an alternate directory for mock API fixtures*

Description

By default, `with_mock_api()` will look for and `capture_requests()` will write mocks to your package's `tests/testthat` directory, or else the current working directory if that path does not exist. If you want to look in or write to other places, call `.mockPaths()` to add directories to the search path.

Usage

```
.mockPaths(new)
```

Arguments

`new`	Either a character vector of path(s) to add, or `NULL` to reset to the default.

Details

It works like `base::.libPaths()`: any directories you specify will be added to the list and searched first. The default directory will be searched last. Only unique values are kept: if you provide a path that is already found in `.mockPaths()`, the result effectively moves that path to the first position.

For finer-grained control, or to completely override the default behavior of searching in the current working directory, you can set the option "httptest2.mock.paths" directly.

Value

If `new` is omitted, the function returns the current search paths, a a character vector. If `new` is provided, the updated value will be returned invisibly.

Examples

```
.mockPaths()
.mockPaths("/var/somewhere/else")
.mockPaths()
.mockPaths(NULL)
.mockPaths()
```

`capture_requests` *Record API responses as mock files*

Description

`capture_requests()` is a context that collects the responses from requests you make and stores them as mock files. This enables you to perform a series of requests against a live server once and then build your test suite using those mocks, running your tests in `with_mock_api()`.

Usage

```
capture_requests(expr, simplify = TRUE)

start_capturing(simplify = TRUE)

stop_capturing()
```

Arguments

`expr`	Code to run inside the context
`simplify`	logical: if `TRUE` (default), plain-text responses with status 200 will be written as just the text of the response body. In all other cases, and when `simplify` is `FALSE`, the `httr2_response` object will be written out to a .R file using `base::dput()`.

Details

`start_capturing()` and `stop_capturing()` allow you to turn on/off request recording for more convenient use in an interactive session.

Recorded responses are written out as plain-text files. By storing fixtures as plain-text files, you can more easily confirm that your mocks look correct, and you can more easily maintain them without having to re-record them. If the API changes subtly, such as when adding an additional attribute to an object, you can just touch up the mocks.

If the response has status `200 OK` and the `Content-Type` maps to a supported file extension—currently `.json`, `.html`, `.xml`, `.txt`, `.csv`, and `.tsv`—just the response body will be written out, using the appropriate extension. `204 No Content` status responses will be stored as an empty file with extension `.204`. Otherwise, the response will be written as a `.R` file containing syntax that, when executed, recreates the `httr2_response` object.

Files are saved to the first directory in `.mockPaths()`, which if not otherwise specified is either "tests/testthat" if it exists (as it should if you are in the root directory of your package), else the current working directory. If you have trouble when recording responses, or are unsure where the files are being written, set `options(httptest2.verbose = TRUE)` to print a message for every file that is written containing the absolute path of the file.

Value

`capture_requests()` returns the result of `expr`. `start_capturing()` invisibly returns the destination directory. `stop_capturing()` returns nothing; it is called for its side effects.

See Also

`build_mock_url()` for how requests are translated to file paths. And see `vignette("redacting", package = "httptest2")` for details on how to prune sensitive content from responses when recording.

Examples

```
# Setup so that our examples clean up after themselves
tmp <- tempfile()
.mockPaths(tmp)
on.exit(unlink(tmp, recursive = TRUE))

library(httr2)
capture_requests({
  request("http://httpbin.org/get") %>% req_perform()
  request("http://httpbin.org/response-headers") %>%
    req_headers(`Content-Type` = "application/json") %>%
    req_perform()
})
# Or:
start_capturing()
request("http://httpbin.org/get") %>% req_perform()
request("http://httpbin.org/response-headers") %>%
  req_headers(`Content-Type` = "application/json") %>%
  req_perform()
stop_capturing()
```

---

change_state *Handle a change of server state*

Description

In a vignette, put a call to `change_state()` before any code block that makes a change on the server, or rather, before any code block that might repeat the same request previously done and expect a different result.

Usage

```
change_state()
```

Details

`change_state()` works by layering a new directory on top of the existing `.mockPaths()`, so fixtures are recorded/loaded there, masking rather than overwriting previously recorded responses for the same request. In vignettes, these mock layers are subdirectories with integer names.

Value

Invisibly, the return of `.mockPaths()` with the new path added.

See Also

`start_vignette()`; `vignette("vignettes", package = "httptest2")` for an overview of all

---

expect_request_header *Test that an HTTP request is made with a header*

Description

This expectation checks that HTTP headers (and potentially header values) are present in a request. It works both in the mock HTTP contexts and on "live" HTTP requests.

Usage

```
expect_request_header(
  expr,
  ...,
  fixed = FALSE,
  ignore.case = FALSE,
  perl = FALSE,
  useBytes = FALSE
)
```

Arguments

`expr`	Code to evaluate
`...`	Named headers to match. Values should either be a string (length-1 character), which will be passed to `testthat::expect_match()`, or `NULL` to assert that the named header is not present in the request. To assert that a header is merely present in the request, without asserting anything about its contents, provide an empty string (`""`). Header names are always case-insensitive; header values will be matched using the following parameters:
`fixed`	logical. If `TRUE`, `pattern` is a string to be matched as is. Overrides all conflicting arguments.

ignore.case	if FALSE, the pattern matching is *case sensitive* and if TRUE, case is ignored during matching.
perl	logical. Should Perl-compatible regexps be used?
useBytes	logical. If TRUE the matching is done byte-by-byte rather than character-by-character. See 'Details'.

Value

The value of expr if there are no expectation failures

Examples

```
library(httr2)
expect_request_header(
  request("http://httpbin.org") %>%
    req_headers(Accept = "image/png") %>%
    req_perform(),
  accept = "image/png",
  `x-fake-header` = NULL
)
expect_request_header(
  request("http://httpbin.org") %>%
    req_headers(Accept = "image/png") %>%
    req_perform(),
  accept = ""
)
```

---

expect_verb *Expectations for mocked HTTP requests*

Description

The mock contexts in httptest2 can raise errors or messages when requests are made, and those (error) messages have three elements, separated by space: (1) the request method (e.g. "GET"); (2) the request URL; and (3) the request body, if present. These verb-expectation functions look for this message shape. expect_PUT, for instance, looks for a request message that starts with "PUT".

Usage

```
expect_GET(object, url = "", ...)

expect_POST(object, url = "", ...)

expect_PATCH(object, url = "", ...)

expect_PUT(object, url = "", ...)
```

```
expect_DELETE(object, url = "", ...)

expect_no_request(object, ...)
```

Arguments

`object`	Code to execute that may cause an HTTP request
`url`	character: the URL you expect a request to be made to. Default is an empty string, meaning that you can just assert that a request is made with a certain method without asserting anything further.
`...`	character segments of a request payload you expect to be included in the request body, to be joined together by `paste0()`. You may also pass any of the following named logical arguments, which will be passed to `base::grepl()`: • `fixed`: Should matching take the pattern as is or treat it as a regular expression. Default: `TRUE`, and note that this default is the opposite of the default in `grepl`. (The rest of the arguments follow its defaults.) • `ignore.case`: Should matching be done case insensitively? Default: `FALSE`, meaning matches are case sensitive. • `perl`: Should Perl-compatible regular expressions be used? Default: `FALSE` • `useBytes`: Should matching be done byte-by-byte rather than character-by-character? Default: `FALSE`

Value

A `testthat` 'expectation'.

Examples

```
library(httr2)
without_internet({
  expect_GET(
    request("http://httpbin.org/get") %>% req_perform(),
    "http://httpbin.org/get"
  )
  expect_GET(
    request("http://httpbin.org/get") %>% req_perform(),
    "http://httpbin.org/[a-z]+",
    fixed = FALSE # For regular expression matching
  )
  expect_PUT(
    request("http://httpbin.org/put") %>%
      req_method("PUT") %>%
      req_body_json(list(a = 1)) %>%
      req_perform(),
    "http://httpbin.org/put",
    '{"a":1}'
  )
  # Don't need to assert the request body, or even the URL
  expect_PUT(
```

```
    request("http://httpbin.org/put") %>%
      req_method("PUT") %>%
      req_body_json(list(a = 1)) %>%
      req_perform()
  )
  expect_no_request(rnorm(5))
})
```

---

gsub_response *Find and replace within a response object*

Description

This function passes its arguments to `base::gsub()` in order to find and replace string patterns (regular expressions) within the URL and the response body of `httr2_response` objects.

Usage

```
gsub_response(response, pattern, replacement, ...)
```

Arguments

`response`	An `httr2_response` or `http2_request` object to sanitize.
`pattern`	From `base::gsub()`: "character string containing a regular expression (or character string for `fixed = TRUE`) to be matched in the given character vector." Passed to `gsub()`. See the docs for `gsub()` for further details.
`replacement`	A replacement for the matched pattern, possibly including regular expression backreferences. Passed to `gsub()`. See the docs for `gsub()` for further details.
`...`	Additional logical arguments passed to `gsub()`: `ignore.case`, `perl`, `fixed`, and `useBytes` are the possible options.

Details

Note that, unlike `gsub()`, the first argument of the function is `response`, not `pattern`, while the equivalent argument in `gsub()`, `"x"`, is placed third. This difference is to maintain consistency with the other redactor functions in `httptest2`, which all take `response` as the first argument.

This function also can be applied to an `http2_request` object to replace patterns inside the request URL.

Value

An `httr2_response` object, same as was passed in, with the pattern replaced in the URLs and bodies.

redact *Remove sensitive content from HTTP responses*

Description

When recording requests for use as test fixtures, you don't want to include secrets like authentication tokens and personal ids. These functions provide a means for redacting this kind of content, or anything you want, from responses that `capture_requests()` saves.

Usage

```
redact_cookies(response)

redact_headers(response, headers = c())

within_body_text(response, FUN)
```

Arguments

`response`	An `httr2_response` or `httr2_request` object to sanitize.
`headers`	For `redact_headers()`, a character vector of header names to sanitize.
`FUN`	For `within_body_text()`, a function that takes as its argument a character vector and returns a modified version of that. This function will be applied to the text of the response's body.

Details

`redact_cookies()` removes cookies from `httr2_response` objects and is the default redactor in `capture_requests()`. `redact_headers()` lets you target selected request and response headers for redaction. `within_body_text()` lets you manipulate the text of the response body and manages the parsing of the raw (binary) data in the `httr_response` object.

Note that if you set a redacting function, it will also be applied to requests when loading mocks. This allows you to sanitize and/or shorten URLs in your mock files.

Value

All redacting functions return a well-formed `httr2_response` or `httr2_request` object.

See Also

`vignette("redacting", package = "httptest2")` for a detailed discussion of what these functions do and how to customize them. `gsub_response()` is another redactor.

`set_redactor` *Set a response redactor*

Description

A redactor is a function that alters the response content being written out in the `capture_requests()` context, allowing you to remove sensitive values, such as authentication tokens, as well as any other modification or truncation of the response body. By default, the `redact_cookies()` function will be used to purge standard auth methods, but `set_redactor()` allows you to provide a different one.

Usage

```
set_redactor(FUN)
```

Arguments

`FUN`	A function or expression that modifies `httr2_response` objects. Specifically, a valid input is one of: • A function taking a single argument, the `httr2_response`, and returning a valid `httr2_response` object. • A formula as shorthand for an anonymous function with `.` as the "response" argument, as in the `purrr` package. That is, instead of `function (response) redact_headers(response, "X-Custom-Header")`, you can use `~ redact_headers(., "X-Custom-Header")` • A list of redacting functions/formulas, which will be executed in sequence on the response • `NULL`, to override the default `redact_cookies()`.

Details

Alternatively, you can put a redacting function in `inst/httptest2/redact.R` in your package, and any time your package is loaded (as in when running tests or building vignettes), the function will be used automatically.

Value

Invisibly, the redacting function, validated and perhaps modified. Formulas and function lists are turned into proper functions. `NULL` as input returns the `force()` function.

See Also

For further details on how to redact responses, see `vignette("redacting", package = "httptest2")`.

Examples

```
# Shorten UUIDs in response body/URLs to their first 6 digits:
set_redactor(function(resp) gsub_response(resp, "([0-9a-f]{6})[0-9a-f]{26}", "\\1"))
# Restore the default
set_redactor(redact_cookies)
```

---

start_vignette *Set mocking/capturing state for a vignette*

Description

Use `start_vignette()` to either use previously recorded responses, if they exist, or capture real responses for future use.

Usage

```
start_vignette(dir, ...)

end_vignette()
```

Arguments

`dir`	Root file path for the mocks for this vignette. A good idea is to use the file name of the vignette itself.
`...`	Optional arguments passed to `start_capturing()`

Details

In a vignette or other R Markdown or Sweave document, place `start_vignette()` in an R code block at the beginning, before the first API request is made, and put `end_vignette()` in a R code chunk at the end. You may want to make those R code chunks have `echo=FALSE` in order to hide the fact that you're calling them.

As in `with_mock_dir()`, the behavior changes based on the existence of the `dir` directory. The first time you build the vignette, the directory won't exist yet, so it will make real requests and record them inside of `dir`. On subsequent runs, the mocks will be used. To record fresh responses from the server, delete the `dir` directory, and the responses will be recorded again the next time the vignette runs.

If you have additional setup code that you'd like available across all of your package's vignettes, put it in `inst/httptest2/start-vignette.R` in your package, and it will be called in `start_vignette()` before the mock/record context is set. Similarly, teardown code can go in `inst/httptest2/end-vignette.R`, evaluated in `end_vignette()` after mocking is stopped.

Value

Nothing; called for its side effect of starting/ending response recording or mocking.

See Also

`start_capturing()` for how requests are recorded; `use_mock_api()` for how previously recorded requests are loaded; `change_state()` for how to handle recorded requests when the server state is changing; `vignette("vignettes", package = "httptest2")` for an overview of all

---

`use_httptest2` *Use 'httptest2' in your tests*

Description

This function adds `httptest2` to Suggests in the package DESCRIPTION and loads it in `tests/testthat/setup.R`. Call it once when you're setting up a new package test suite.

Usage

```
use_httptest2(path = ".")
```

Arguments

`path`	character path to the package

Details

The function is idempotent: if `httptest2` is already added to these files, no additional changes will be made.

Value

Nothing: called for file system side effects.

`without_internet` *Make all HTTP requests raise an error*

Description

`without_internet()` simulates the situation when any network request will fail, as in when you are without an internet connection. Any HTTP request through `httr2` will raise an error.

Usage

```
without_internet(expr)

block_requests()
```

Arguments

expr	Code to run inside the mock context

Details

`block_requests()` and `stop_mocking()` allow you to turn on/off request blocking for more convenient use in an interactive session.

The error message raised has a well-defined shape, made of three elements, separated by space: (1) the request method (e.g. "GET"); (2) the request URL; and (3) the request body, if present. The verb-expectation functions, such as `expect_GET()` and `expect_POST()`, look for this shape.

Value

The result of `expr`

Examples

```
library(httr2)
library(testthat, warn.conflicts = FALSE)
without_internet({
  expect_error(
    request("http://httpbin.org/get") %>% req_perform(),
    "GET http://httpbin.org/get"
  )
  expect_error(
    request("http://httpbin.org/put") %>%
      req_method("PUT") %>%
      req_body_json(list(a = 1)) %>%
      req_perform(),
    'PUT http://httpbin.org/put {"a":1}',
    fixed = TRUE
  )
})
```

`with_mock_api` *Serve a mock API from files*

Description

In this context, HTTP requests attempt to load API response fixtures from files. This allows test code to proceed evaluating code that expects HTTP requests to return meaningful responses. Requests that do not have a corresponding fixture file raise errors, like how `without_internet()` does.

Usage

```
with_mock_api(expr)

use_mock_api()

stop_mocking()
```

Arguments

expr	Code to run inside the mock context

Details

use_mock_api() and stop_mocking() allow you to turn on/off request mocking for more convenient use in an interactive session.

Requests are translated to mock file paths according to several rules that incorporate the request method, URL, query parameters, and body. See build_mock_url() for details.

File paths for API fixture files may be relative to the 'tests/testthat' directory, i.e. relative to the .R test files themselves. This is the default location for storing and retrieving mocks, but you can put them anywhere you want as long as you set the appropriate location with .mockPaths().

Value

with_mock_api() returns the result of expr. use_mock_api() and stop_mocking() return nothing.

Examples

```
library(httr2)
with_mock_api({
  # There are no mocks recorded in this example, so catch this request with
  # expect_GET()
  expect_GET(
    request("https://cran.r-project.org") %>% req_perform(),
    "https://cran.r-project.org"
  )
  # For examples with mocks, see the tests and vignettes
})
```

---

with_mock_dir *Use or create mock files depending on their existence*

Description

This context will switch the .mockPaths() to tests/testthat/dir (and then resets it to what it was before). If the tests/testthat/dir folder doesn't exist, capture_requests() will be run to create mocks. If it exists, with_mock_api() will be run. To re-record mock files, simply delete tests/testthat/dir and run the test.

Usage

```
with_mock_dir(dir, expr, simplify = TRUE, replace = TRUE)
```

Arguments

dir	character string, unique folder name that will be used or created under tests/testthat/
expr	Code to run inside the context
simplify	logical: if TRUE (default), plain-text responses with status 200 will be written as just the text of the response body. In all other cases, and when simplify is FALSE, the httr2_response object will be written out to a .R file using base::dput().
replace	Logical: should dir replace the contents of .mockPaths() (default) or be added in front of the existing paths? The default behavior here is the opposite of .mockPaths() so that the tests inside of with_mock_dir() are fully isolated.

See Also

vignette("httptest2") for usage examples.

Index