Package: nectar (via r-universe)

October 23, 2024

Title A Framework for Web API Packages

Version 0.0.0.9003

Description An opinionated framework for use within api-wrapping R
packages.

License MIT + file LICENSE

URL `https://nectar.api2r.org`, `https://github.com/jonthegeek/nectar`

BugReports `https://github.com/jonthegeek/nectar/issues`

Depends R (>= 3.5.0)

Imports cli, curl, fs, glue, httr2 (>= 1.0.0), jsonlite, purrr, rlang,
stbl, vctrs

Suggests covr, testthat (>= 3.0.0)

Remotes jonthegeek/stbl

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

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

RemoteUrl https://github.com/jonthegeek/nectar

RemoteRef HEAD

RemoteSha cf92bd8f0c6f1ecdcc44b3a55e05146af3efa84f

Contents

`call_api` *Send a request to an API*

Description

This function implements an opinionated framework for making API calls. It is intended to be used inside an API client package. It serves as a wrapper around the `req_` family of functions, such as `httr2::request()`, as well as `httr2::req_perform()` and `httr2::req_perform_iterative()`, and, by default, `httr2::resp_body_json()`.

Usage

```
call_api(
  base_url,
  ...,
  path = NULL,
  query = NULL,
  body = NULL,
  mime_type = NULL,
  method = NULL,
  security_fn = NULL,
  security_args = list(),
  response_parser = httr2::resp_body_json,
  response_parser_args = list(),
  next_req = NULL,
  max_reqs = Inf,
  max_tries_per_req = 3,
  user_agent = "nectar (https://nectar.api2r.org)"
)
```

Arguments

`base_url`	The part of the url that is shared by all calls to the API. In some cases there may be a family of base URLs, from which you will need to choose one.
`...`	These dots are for future extensions and must be empty.
`path`	The route to an API endpoint. Optionally, a list or character vector with the path as one or more unnamed arguments (which will be concatenated with "/") plus named arguments to `glue::glue()` into the path.

`query`	An optional list or character vector of parameters to pass in the query portion of the request. Can also include a `.multi` argument to pass to `httr2::req_url_query()` to control how elements containing multiple values are handled.
`body`	An object to use as the body of the request. If any component of the body is a path, pass it through `fs::path()` or otherwise give it the class "fs_path" to indicate that it is a path.
`mime_type`	A character scalar indicating the mime type of any files present in the body. Some APIs allow you to leave this as NULL for them to guess.
`method`	If the method is something other than GET or POST, supply it. Case is ignored.
`security_fn`	A function to use to authenticate the request. By default (`NULL`), no authentication is performed.
`security_args`	An optional list of arguments to the `security_fn` function.
`response_parser`	A function to parse the server response (`resp`). Defaults to `httr2::resp_body_json()`, since JSON responses are common. Set this to `NULL` to return the raw response from `httr2::req_perform()`.
`response_parser_args`	An optional list of arguments to pass to the `response_parser` function (in addition to `resp`).
`next_req`	An optional function that takes the previous response (`resp`) to generate the next request in a call to `httr2::req_perform_iterative()`. This function can usually be generated using one of the iteration helpers described in `httr2::iterate_with_offset()`.
`max_reqs`	The maximum number of separate requests to perform. Passed to the max_reqs argument of `httr2::req_perform_iterative()` when `next_req` is supplied. The default `2` should likely be changed to `Inf` after you validate the function.
`max_tries_per_req`	The maximum number of times to attempt each individual request. Passed to the `max_tries` argument of `httr2::req_retry()`.
`user_agent`	A string to identify where this request is coming from. It's polite to set the user agent to identify your package, such as "MyPackage (https://mypackage.com)".

Value

The response from the API, parsed by the `response_parser`.

See Also

`req_setup()`, `req_modify()`, `req_perform_opinionated()`, `resp_parse()`, and `do_if_fn_defined()` for finer control of the process.

`compact_nested_list` *Discard empty elements*

Description

Discard empty elements in nested lists.

Usage

```
compact_nested_list(lst)
```

Arguments

`lst`	A (nested) list to filter.

Value

The list, minus empty elements and branches.

Examples

```
x <- list(
  a = list(
    b = letters,
    c = NULL,
    d = 1:5
  ),
  e = NULL,
  f = 1:3
)
compact_nested_list(x)
```

`do_if_fn_defined` *Use a provided function*

Description

When constructing API calls programmatically, you may encounter situations where an upstream task should indicate which function to apply. For example, one endpoint might use a special security function that isn't used by other endpoints. This function exists to make coding such situations easier.

Usage

```
do_if_fn_defined(x, fn = NULL, ...)
```

Arguments

`x`	An object to potentially modify, such as a `httr2::request()` object.
`fn`	A function to apply to `x`. If `fn` is `NULL`, `x` is returned unchanged.
`...`	Additional arguments to pass to `fn`.

Value

The object, potentially modified.

Examples

```
build_api_req <- function(endpoint, security_fn = NULL, ...) {
  req <- httr2::request("https://example.com")
  req <- httr2::req_url_path_append(req, endpoint)
  do_if_fn_defined(req, security_fn, ...)
}

# Most endpoints of this API do not require authentication.
unsecure_req <- build_api_req("unsecure_endpoint")
unsecure_req$headers

# But one endpoint requires
secure_req <- build_api_req(
  "secure_endpoint", httr2::req_auth_bearer_token, "secret-token"
)
secure_req$headers$Authorization
```

`req_auth_api_key` *Authenticate with an API key*

Description

Many APIs provide API keys that can be used to authenticate requests (or, often, provide other information about the user). This function helps to apply those keys to requests.

Usage

```
req_auth_api_key(req, ..., location = "header")
```

Arguments

`req`	A `httr2::request()` object.
`...`	Additional parameters depending on the location of the API key. • `parameter_name` ("header" or "query" only) The name of the parameter to use in the header or query. • `api_key` ("header" or "query" only) The API key to use. • `path` ("cookie" only) The location of the cookie.
`location`	Where the API key should be passed. One of `"header"` (default), `"query"`, or `"cookie"`.

Value

A `httr2::request()` object.

`req_modify` *Modify an API request for a particular endpoint*

Description

Modify the basic request for an API by adding a path and any other path-specific properties.

Usage

```
req_modify(
  req,
  ...,
  path = NULL,
  query = NULL,
  body = NULL,
  mime_type = NULL,
  method = NULL
)
```

Arguments

`req`	A `httr2::request()` object.
`...`	These dots are for future extensions and must be empty.
`path`	The route to an API endpoint. Optionally, a list or character vector with the path as one or more unnamed arguments (which will be concatenated with "/") plus named arguments to `glue::glue()` into the path.
`query`	An optional list or character vector of parameters to pass in the query portion of the request. Can also include a `.multi` argument to pass to `httr2::req_url_query()` to control how elements containing multiple values are handled.
`body`	An object to use as the body of the request. If any component of the body is a path, pass it through `fs::path()` or otherwise give it the class "fs_path" to indicate that it is a path.
`mime_type`	A character scalar indicating the mime type of any files present in the body. Some APIs allow you to leave this as NULL for them to guess.
`method`	If the method is something other than GET or POST, supply it. Case is ignored.

Value

A `httr2::request()` object.

Examples

```
req_base <- req_setup(
  "https://example.com",
  user_agent = "my_api_client (https://my.api.client)"
)
req <- req_modify(req_base, path = c("specific/{path}", path = "endpoint"))
req
req <- req_modify(req, query = c("param1" = "value1", "param2" = "value2"))
req
```

`req_perform_opinionated` *Perform a request with opinionated defaults*

Description

This function ensures that a request has `httr2::req_retry()` applied, and then performs the request, using either `httr2::req_perform_iterative()` (if a `next_req` function is supplied) or `httr2::req_perform()` (if not).

Usage

```
req_perform_opinionated(
  req,
  ...,
  next_req = NULL,
  max_reqs = 2,
  max_tries_per_req = 3
)
```

Arguments

`req`	The first request to perform.
`...`	These dots are for future extensions and must be empty.
`next_req`	An optional function that takes the previous response (`resp`) to generate the next request in a call to `httr2::req_perform_iterative()`. This function can usually be generated using one of the iteration helpers described in `httr2::iterate_with_offset()`.
`max_reqs`	The maximum number of separate requests to perform. Passed to the max_reqs argument of `httr2::req_perform_iterative()` when `next_req` is supplied. The default 2 should likely be changed to `Inf` after you validate the function.
`max_tries_per_req`	The maximum number of times to attempt each individual request. Passed to the `max_tries` argument of `httr2::req_retry()`.

Value

A list of `httr2::response()` objects, one for each request performed.

req_prepare *Prepare a request for an API*

Description

This function implements an opinionated framework for preparing an API request. It is intended to be used inside an API client package. It serves as a wrapper around the `req_` family of functions, such as `httr2::request()`.

Usage

```
req_prepare(
  base_url,
  ...,
  path = NULL,
  query = NULL,
  body = NULL,
  mime_type = NULL,
  method = NULL,
  user_agent = "nectar (https://nectar.api2r.org)"
)
```

Arguments

`base_url`	The part of the url that is shared by all calls to the API. In some cases there may be a family of base URLs, from which you will need to choose one.
`...`	These dots are for future extensions and must be empty.
`path`	The route to an API endpoint. Optionally, a list or character vector with the path as one or more unnamed arguments (which will be concatenated with "/") plus named arguments to `glue::glue()` into the path.
`query`	An optional list or character vector of parameters to pass in the query portion of the request. Can also include a `.multi` argument to pass to `httr2::req_url_query()` to control how elements containing multiple values are handled.
`body`	An object to use as the body of the request. If any component of the body is a path, pass it through `fs::path()` or otherwise give it the class "fs_path" to indicate that it is a path.
`mime_type`	A character scalar indicating the mime type of any files present in the body. Some APIs allow you to leave this as NULL for them to guess.
`method`	If the method is something other than GET or POST, supply it. Case is ignored.
`user_agent`	A string to identify where this request is coming from. It's polite to set the user agent to identify your package, such as "MyPackage (https://mypackage.com)".

Value

A `httr2::request()` object.

`req_setup` *Setup a basic API request*

Description

For a given API, the `base_url` and `user_agent` will almost always be the same. Use this function to prepare that piece of the request once for easy reuse.

Usage

```
req_setup(base_url, ..., user_agent = "nectar (https://nectar.api2r.org)")
```

Arguments

`base_url`	The part of the url that is shared by all calls to the API. In some cases there may be a family of base URLs, from which you will need to choose one.
`...`	These dots are for future extensions and must be empty.
`user_agent`	A string to identify where this request is coming from. It's polite to set the user agent to identify your package, such as "MyPackage (https://mypackage.com)".

Value

A `httr2::request()` object.

Examples

```
req_setup("https://example.com")
req_setup(
  "https://example.com",
  user_agent = "my_api_client (https://my.api.client)"
)
```

`resp_parse` *Parse one or more responses*

Description

`httr2` provides two methods for performing requests: `httr2::req_perform()`, which returns a single `httr2::response()` object, and `httr2::req_perform_iterative()`, which returns a list of `httr2::response()` objects. This function automatically determines whether a single response or multiple responses have been returned, and parses the responses appropriately.

Usage

```
resp_parse(resp, ...)

## Default S3 method:
resp_parse(
  resp,
  ...,
  arg = rlang::caller_arg(resp),
  call = rlang::caller_env()
)

## S3 method for class 'httr2_response'
resp_parse(resp, ..., response_parser = httr2::resp_body_json)
```

Arguments

`resp`	A single `httr2::response()` object (as returned by `httr2::req_perform()`) or a list of such objects (as returned by `httr2::req_perform_iterative()`).
`...`	Additional arguments passed on to the `response_parser` function (in addition to `resp`).
`arg`	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
`call`	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.
`response_parser`	A function to parse the server response (`resp`). Defaults to `httr2::resp_body_json()`, since JSON responses are common. Set this to `NULL` to return the raw response from `httr2::req_perform()`.

Value

The response parsed by the `response_parser`. If `resp` was a list, the parsed responses are concatenated when possible. Unlike httr2::resps_data, this function does not concatenate raw vector responses.

`stabilize_string` *Ensure an argument is a length-1 character*

Description

Calls to APIs often require a string argument. This function ensures that those arguments are length-1, non-`NA` character vectors, or length-1, non-`NA` vectors that can be coerced to character vectors. This is intended to ensure that calls to the API will not fail with predictable errors, thus avoiding unnecessary internet traffic.

Usage

```
stabilize_string(
  x,
  ...,
  regex = NULL,
  arg = rlang::caller_arg(x),
  call = rlang::caller_env()
)
```

Arguments

`x`	The argument to stabilize.
`...`	Arguments passed on to `stbl::stabilize_chr_scalar` `x_class` Character. The class name of `x` to use in error messages. Use this if you remove a special class from `x` before checking its coercion, but want the error message to match the original class.
`regex`	Character scalar. An optional regex pattern to compare the value(s) of `x` against. If a complex regex pattern throws an error, try installing the stringi package with `install.packages("stringi")`.
`arg`	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
`call`	The execution environment of a currently running function, e.g. `caller_env()`. The function will be mentioned in error messages as the source of the error. See the `call` argument of `abort()` for more information.

Value

`x` coerced to a length-1 character vector, if possible.

Examples

```
stabilize_string("a")
stabilize_string(1.1)
x <- letters
try(stabilize_string(x))
x <- NULL
try(stabilize_string(x))
x <- character()
try(stabilize_string(x))
x <- NA
try(stabilize_string(x))
```

`url_normalize` *Normalize a URL*

Description

This function normalizes a URL by adding a trailing slash to the base if it is missing. It is mainly for testing and other comparisons.

Usage

```
url_normalize(url)
```

Arguments

`url`	A URL to normalize.

Value

A normalized URL

Examples

```
identical(
  url_normalize("https://example.com"),
  url_normalize("https://example.com/")
)
identical(
  url_normalize("https://example.com?param=value"),
  url_normalize("https://example.com/?param=value")
)
```

`url_path_append` *Add path elements to a URL*

Description

Append zero or more path elements to a URL without duplicating "/" characters. Based on `httr2::req_url_path_append()`

Usage

```
url_path_append(url, ...)
```

Arguments

`url`	A URL to modify.
`...`	Path elements to append, as strings.

Value

A modified URL.

Examples

```
url_path_append("https://example.com", "api", "v1", "users")
url_path_append("https://example.com/", "/api", "/v1", "/users")
url_path_append("https://example.com/", "/api/v1/users")
```

Index