Package 'googleAuthR'

November 16, 2018

Type Package

Version 0.7.0

Title Authenticate and Create Google APIs

Description Create R functions that interact with OAuth2 Google APIs <https://developers.google.com/apis-explorer/> easily, with auto-refresh and Shiny compatibility.

URL http://code.markedmondson.me/googleAuthR/

BugReports https://github.com/MarkEdmondson1234/googleAuthR/issues

Depends R (>= 3.3.0)

Imports assertthat (>= 0.2.0), digest, httr (>= 1.3.1), jsonlite (>= 0.9.16), R6 (>= 2.1.0), memoise (>= 1.1.0), utils

Suggests bigQueryR, covr, devtools (>= 1.12.0), formatR (>= 1.4), googleAnalyticsR, httptest, knitr, miniUI (>= 0.1.1), rmarkdown, roxygen2 (>= 5.0.0), shiny (>= 0.13.2), testthat

License MIT + file LICENSE

LazyData true

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation no

Author Mark Edmondson [aut, cre] (<https://orcid.org/0000-0002-8434-3881>), Jennifer Bryan [ctb], Johann deBoer [ctb], Neal Richardson [ctb], David Kulp [ctb], Joe Cheng [ctb]

Maintainer Mark Edmondson <m@sunholo.com>

Repository CRAN

Date/Publication 2018-11-16 21:00:03 UTC

R topics documented:

`Authentication` *R6 environment to store authentication credentials*

Description

Used to keep persistent state.

Usage

```
Authentication
```

Format

An object of class `R6ClassGenerator` of length 24.

`gar_api_generator` *googleAuthR data fetch function generator*

Description

This function generates other functions for use with Google APIs

Usage

```
gar_api_generator(baseURI, http_header = c("GET", "POST", "PUT", "DELETE",
  "PATCH"), path_args = NULL, pars_args = NULL,
  data_parse_function = NULL, customConfig = NULL,
  simplifyVector = getOption("googleAuthR.jsonlite.simplifyVector"),
  checkTrailingSlash = TRUE)
```

Arguments

`baseURI`	The stem of the API call.
`http_header`	Type of http request.
`path_args`	A named list with name=folder in request URI, value=the function variable.
`pars_args`	A named list with name=parameter in request URI, value=the function variable.
`data_parse_function`	A function that takes a request response, parses it and returns the data you need.
`customConfig`	list of httr options such as use_proxy or add_headers that will be added to the request.
`simplifyVector`	Passed to fromJSON for response parsing
`checkTrailingSlash`	Default TRUE will append a trailing slash to baseURI if missing

Details

path_args and **pars_args** add default values to the baseURI. NULL entries are removed. Use "" if you want an empty argument.

You don't need to supply access_token for OAuth2 requests in pars_args, this is dealt with in gar_auth()

Add custom configurations to the request in this syntax: `customConfig = list(httr::add_headers("From" = "mark@exa`

Value

A function that can fetch the Google API data you specify

Examples

```
## Not run:
library(googleAuthR)
## change the native googleAuthR scopes to the one needed.
options("googleAuthR.scopes.selected" =
  c("https://www.googleapis.com/auth/urlshortener"))

shorten_url <- function(url){

  body = list(
    longUrl = url
    )

  f <- gar_api_generator("https://www.googleapis.com/urlshortener/v1/url",
                      "POST",
                      data_parse_function = function(x) x$id)

   f(the_body = body)
 }

To use the above functions:
library(googleAuthR)
# go through authentication flow
gar_auth()
s <- shorten_url("http://markedmondson.me")
s

## End(Not run)
```

---

gar_api_page *Takes a generated API function and lets you page through results*

Description

A helper function to help with the common task of paging through large API results.

Usage

```
gar_api_page(f, page_f = function(x) x$nextLink, page_method = c("url",
  "param"), page_arg = NULL)
```

Arguments

f	a function created by gar_api_generator
page_f	A function that will extract the next page information from f()
page_method	Method of paging: url will fetch by changing the fetch URL; param will fetch the next page via a parameter set in page_arg
page_arg	If page_method="param", you need to set this to the parameter that will change for each API page.

Details

The page_f function operates on the object returned from the data_parse_function of the function f

If using page_method="url" then then page_f function needs to return the URL that will fetch the next page of results. The default finds this via x$nextLink. This is the easiest to implement if available and is recommended.

If using page_method = "param", then page_f needs to extract the parameter specified in page_arg that will fetch the next page of the results, or NULL if no more pages are required. e.g. if response is x, page_f should extract the next value for the parameter of page_arg that fetches the next results. It should also return NULL if no (more) paging is necessary. See examples. Remember to add the paging argument (e.g. start-index) to the generated function too, so it can be modified.

Value

A list of the API page responses, that you may need to process further into one object.

Examples

```
## Not run:
# demos the two methods for the same function.
# The example is for the Google Analytics management API,
#  you need to authenticate with that to run them.


# paging by using nextLink that is returned in API response
ga_segment_list1 <- function(){

  # this URL will be modified by using the url_override argument in the generated function
  segs <- gar_api_generator("https://www.googleapis.com/analytics/v3/management/segments",
                            "GET",
                            pars_args = list("max-results"=10),
                            data_parse_function = function(x) x)


  gar_api_page(segs,
               page_method = "url",
               page_f = function(x) x$nextLink)

}
```

```
# paging by looking for the next start-index parameter

## start by creating the function that will output the correct start-index
paging_function <- function(x){
  next_entry <- x$startIndex + x$itemsPerPage

  # we have all results e.g. 1001 > 1000
  if(next_entry > x$totalResults){
    return(NULL)
  }

  next_entry
  }

## remember to add the paging argument (start-index) to the generated function too,
##  so it can be modified.
ga_segment_list2 <- function(){

  segs <- gar_api_generator("https://www.googleapis.com/analytics/v3/management/segments",
                            "GET",
                             pars_args = list("start-index" = 1,
                                              "max-results"=10),
                             data_parse_function = function(x) x)

  gar_api_page(segs,
               page_method = "param",
               page_f = paging_function,
               page_arg = "start-index")

  }


identical(ga_segment_list1(), ga_segment_list2())



## End(Not run)
```

---

gar_attach_auto_auth *Auto Authentication function for use within .onAttach*

Description

To be placed within .onAttach to auto load an authentication file from an environment variable.

Usage

```
gar_attach_auto_auth(required_scopes, environment_var = "GAR_AUTH_FILE",
  travis_environment_var = NULL)
```

Arguments

required_scopes
: A character vector of minimum required scopes for this API library

environment_var
: The name of the environment variable where the file path to the authentication file is kept

travis_environment_var
: Defunct; now does nothing

This function works with gar_auto_auth. It is intended to be placed within the .onAttach hook so that it loads when you load your library.

For auto-authentication to work, the environment variable needs to hold a file path to an existing auth file such as created via gar_auth or a JSON file file download from the Google API console.

Value

Invisible, used for its side effects of calling auto-authentication.

See Also

Other authentication functions: gar_auth_service, gar_auth, gar_auto_auth, gar_gce_auth, get_google_token, token_exists

Examples

```
## Not run:

.onAttach <- function(libname, pkgname){

  googleAuthR::gar_attach_auto_auth("https://www.googleapis.com/auth/urlshortener", "US_AUTH_FILE")

}

## will only work if you have US_AUTH_FILE environment variable pointing to an auth file location
## .Renviron example
US_AUTH_FILE="/home/mark/auth/urlshortnerauth.json"


## End(Not run)
```

---

gar_auth *Authorize* googleAuthR

Description

Authorize googleAuthR to access your Google user data. You will be directed to a web browser, asked to sign in to your Google account, and to grant googleAuthR access to user data for Google Search Console. These user credentials are cached in a file named .httr-oauth in the current working directory, from where they can be automatically refreshed, as necessary.

Usage

```
gar_auth(token = NULL, new_user = FALSE)
```

Arguments

token	an actual token object or the path to a valid token stored as an .rds file
new_user	logical, defaults to FALSE. Set to TRUE if you want to wipe the slate clean and re-authenticate with the same or different Google account. This deletes the .httr-oauth file in current working directory.

Details

These arguments are controlled via options, which, if undefined at the time googleAuthR is loaded, are defined like so:

key Set to option googleAuthR.client_id, which defaults to a client ID that ships with the package

secret Set to option googleAuthR.client_secret, which defaults to a client secret that ships with the package

cache Set to option googleAuthR.httr_oauth_cache, which defaults to TRUE

scopes Set to option googleAuthR.scopes.selected, which defaults to demo scopes.

To override these defaults in persistent way, predefine one or more of them with lines like this in a .Rprofile file:

```
options(googleAuthR.client_id = "FOO",
        googleAuthR.client_secret = "BAR",
        googleAuthR.httr_oauth_cache = FALSE)
```

See Startup for possible locations for this file and the implications thereof.

More detail is available from Using OAuth 2.0 to Access Google APIs. This function executes the "installed application" flow.

Value

an OAuth token object, specifically a Token2.0, invisibly

See Also

Other authentication functions: gar_attach_auto_auth, gar_auth_service, gar_auto_auth, gar_gce_auth, get_google_token, token_exists

gar_auth_js *Shiny JavaScript Google Authorisation [Server Module]*

Description

Shiny Module for use with gar_auth_jsUI

Usage

```
gar_auth_js(...)

googleAuth_js(input, output, session,
  message = "Authenticate with your Google account")
```

Arguments

...	Arguments passed to googleAuth_js
input	shiny input
output	shiny output
session	shiny session
message	The message to show when not authenticated

Details

Call via `shiny::callModule(gar_auth_js, "your_id")`

Value

A httr reactive OAuth2.0 token

gar_auth_jsUI *Shiny JavaScript Google Authorisation [UI Module]*

Description

A Javascript Google authorisation flow for Shiny apps.

Usage

```
gar_auth_jsUI(...)

googleAuth_jsUI(id, login_class = "btn btn-primary",
  logout_class = "btn btn-danger", login_text = "Log In",
  logout_text = "Log Out", approval_prompt_force = TRUE,
  scopes = getOption("googleAuthR.scopes.selected", "email"))
```

Arguments

...	Arguments passed to googleAuth_jsUI
id	Shiny id
login_class	CSS class of login button
logout_class	CSS class of logout button
login_text	Text to show on login button
logout_text	Text to show on logout button
approval_prompt_force	Whether to force a login each time
scopes	Set the scopes, minimum needs is "email"

Details

Shiny Module for use with gar_auth_js

Value

Shiny UI

---

gar_auth_service *JSON service account authentication*

Description

As well as OAuth2 authentication, you can authenticate without user interaction via Service accounts. This involves downloading a secret JSON key with the authentication details.

To use, go to your Project in the https://console.developers.google.com/apis/credentials/serviceaccountkey

and select JSON Key type. Save the file to your computer and call it via supplying the file path to the `json_file` parameter.

Navigate to it via: Google Dev Console > Credentials > New credentials > Service account Key > Select service account > Key type = JSON

Usage

```
gar_auth_service(json_file, scope = getOption("googleAuthR.scopes.selected"))
```

Arguments

json_file	the JSON file downloaded from Google Developer Console
scope	Scope of the JSON file auth if needed

Value

(Invisible) Sets authentication token

See Also

https://developers.google.com/identity/protocols/OAuth2ServiceAccount

https://developers.google.com/identity/protocols/OAuth2ServiceAccount

Other authentication functions: gar_attach_auto_auth, gar_auth, gar_auto_auth, gar_gce_auth, get_google_token, token_exists

---

`gar_auto_auth` *Perform auto authentication*

Description

This helper function lets you use environment variables to auto-authenticate on package load, intended for calling by gar_attach_auto_auth

Usage

```
gar_auto_auth(required_scopes, new_user = FALSE, no_auto = FALSE,
  environment_var = "GAR_AUTH_FILE", travis_environment_var = NULL)
```

Arguments

`required_scopes`
: Required scopes needed to authenticate - needs to match at least one

`new_user`
: If TRUE, reauthenticate via Google login screen

`no_auto`
: If TRUE, ignore auto-authentication settings

`environment_var`
: Name of environment var that contains auth file path

`travis_environment_var`
: No longer supported

The authentication file can be a `.httr-oauth` file created via gar_auth or a Google service JSON file downloaded from the Google API credential console, with file extension `.json`.

You can use this in your code to authenticate from a file location specified in file, but it is mainly intended to be called on package load via gar_attach_auto_auth.

`environment_var` This is the name that will be called via Sys.getenv on library load. The environment variable will contain an absolute file path to the location of an authentication file.

Value

an OAuth token object, specifically a `Token2.0`, invisibly

See Also

Help files for .onAttach

Other authentication functions: gar_attach_auto_auth, gar_auth_service, gar_auth, gar_gce_auth, get_google_token, token_exists

gar_batch *Turn a list of gar_fetch_functions into batch functions*

Description

Turn a list of gar_fetch_functions into batch functions

Usage

```
gar_batch(call_list, ...,
  batch_endpoint = getOption("googleAuthR.batch_endpoint", default =
  "https://www.googleapis.com/batch"))
```

Arguments

call_list	a list of functions from gar_api_generator
...	further arguments passed to the data parse function of f
batch_endpoint	the batch API endpoint to send to

Details

This function will turn all the individual Google API functions into one POST request to /batch.

If you need to pass multiple data parse function arguments its probably best to do it in separate batches to avoid confusion.

Value

A list of the Google API responses

See Also

https://developers.google.com/webmaster-tools/v3/how-tos/batch

Documentation on doing batch requests for the search console API. Other Google APIs are similar.

Walk through API calls changing parameters using gar_batch_walk

Other batch functions: gar_batch_walk

Examples

```
## Not run:

## usually set on package load
options(googleAuthR.batch_endpoint = "https://www.googleapis.com/batch/urlshortener/v1")

## from goo.gl API
shorten_url <- function(url){
  body = list(longUrl = url)
```

```
  f <- gar_api_generator("https://www.googleapis.com/urlshortener/v1/url",
                         "POST",
                          data_parse_function = function(x) x$id)

  f(the_body = body)
}


## from goo.gl API
user_history <- function(){
  f <- gar_api_generator("https://www.googleapis.com/urlshortener/v1/url/history",
                       "GET",
                       data_parse_function = function(x) x$items)

  f()
}

gar_batch(list(shorten_url("http://markedmondson.me"), user_history()))


## End(Not run)
```

---

gar_batch_walk *Walk data through batches*

Description

Convenience function for walking through data in batches

Usage

```
gar_batch_walk(f, walk_vector, gar_pars = NULL, gar_paths = NULL,
  the_body = NULL, pars_walk = NULL, path_walk = NULL, body_walk = NULL,
  batch_size = 10, batch_function = NULL, data_frame_output = TRUE, ...,
  batch_endpoint = getOption("googleAuthR.batch_endpoint", default =
  "https://www.googleapis.com/batch"))
```

Arguments

f	a function from gar_api_generator
walk_vector	a vector of the parameter or path to change
gar_pars	a list of parameter arguments for f
gar_paths	a list of path arguments for f
the_body	a list of body arguments for f
pars_walk	a character vector of the parameter(s) to modify for each walk of f
path_walk	a character vector of the path(s) to modify for each walk of f
body_walk	a character vector of the body(s) to modify for each walk of f

batch_size	size of each request to Google /batch API
batch_function	a function that will act on the result list of each batch API call
data_frame_output	if the list of lists are dataframes, you can bind them all by setting to TRUE
...	further arguments passed to the data parse function of f
batch_endpoint	the batch API endpoint to send

Details

You can modify more than one parameter or path arg, but it must be the same walked vector e.g. `start = end = x`

Many Google APIs have `batch_size` limits greater than 10, 1000 is common.

The 'f' function needs to be a 'gar_api_generator()' function that uses one of 'path_args', 'pars_args' or 'body_args' to construct the URL (rather than say using 'sprintf()' to create the API URL).

You don't need to set the headers in the Google docs for batching API functions - those are done for you.

The argument 'walk_vector' needs to be a vector of the values of the arguments to walk over, which you indicate will walk over the pars/path or body arguments on the function via on of the '*_walk' arguments e.g. if walking over id=1, id=2, for a path argument then it would be 'path_walk="id"' and 'walk_vector=c(1,2,3,4)'

The 'gar_*' parameter is required to pass intended for other arguments to the function 'f' you may need to pass through.

'gar_batch_walk()' only supports changing one value at a time, for one or multiple arguments (I think only changing the 'start-date', 'end-date' example would be the case when you walk through more than one per call)

'batch_size' should be over 1 for batching to be of any benefit at all

The 'batch_function' argument gives you a way to operate on the parsed output of each call

Value

if data_frame_output is FALSE: A list of lists. Outer list the length of number of batches required, inner lists the results from the calls

if data_frame_output is TRUE: The list of lists will attempt to rbind all the results

See Also

Other batch functions: gar_batch

Examples

```
## Not run:


# get a webproperty per account
getAccountInfo <- gar_api_generator(
```

```
  "https://www.googleapis.com/analytics/v3/management/accounts",
  "GET", data_parse_function = function(x) unique(x$items$id))

getWebpropertyInfo <- gar_api_generator(
 "https://www.googleapis.com/analytics/v3/management/", # don't use sprintf to construct this
  "GET",
  path_args = list(accounts = "default", webproperties = ""),
  data_parse_function = function(x) x$items)

walkData <- function(){

  # here due to R lazy evaluation
  accs <- getAccountInfo()
  gar_batch_walk(getWebpropertyInfo,
                 walk_vector = accs,
                 gar_paths = list("webproperties" = ""),
                 path_walk = "accounts",
                 batch_size = 100, data_frame_output = FALSE)
                 }

# do the walk
walkData()


## End(Not run)
```

---

gar_cache_get_loc *Setup where to put cache*

Description

To cache to a file system use `memoise::cache_filesystem("cache_folder")`, suitable for unit testing and works between R sessions.

The cached API calls do not need authentication to be active, but need this function to set caching first.

Usage

```
gar_cache_get_loc()

gar_cache_empty()

gar_cache_setup(mcache = memoise::cache_memory(),
  invalid_func = function(req) {     tryCatch(req$status_code == 200, error =
  function(x) FALSE) })
```

Arguments

mcache	A cache method from memoise.
invalid_func	A function that takes API response, and returns TRUE or FALSE whether caching takes place. Default cache everything.

Value

TRUE if successful.

Examples

```
## Not run:

# demo function to cache within
shorten_url_cache <- function(url){
  body = list(longUrl = url)
  f <- gar_api_generator("https://www.googleapis.com/urlshortener/v1/url",
                      "POST",
                      data_parse_function = function(x) x)
 f(the_body = body)

 }

 ## only cache if this URL
 gar_cache_setup(invalid_func = function(req){
      req$content$longUrl == "http://code.markedmondson.me/"
 })

 # authentication
 gar_auth()
 ## caches
 shorten_url_cache("http://code.markedmondson.me")

 ## read cache
 shorten_url("http://code.markedmondson.me")

 ## ..but dont cache me
 shorten_url_cache("http://blahblah.com")


## End(Not run)
```

---

gar_check_existing_token

Check a token vs options

Description

Useful for debugging authentication issues

Usage

```
gar_check_existing_token(token = Authentication$public_fields$token)
```

Arguments

token	A token to check, default current live session token

Details

Will compare the passed token's settings and compare to set options. If these differ, then reauthentication may be needed.

Value

FALSE if the options and current token do not match, TRUE if they do.

---

gar_create_api_objects

Create the API objects from the Discovery API

Description

Create the API objects from the Discovery API

Usage

```
gar_create_api_objects(filename, api_json, format = TRUE)
```

Arguments

filename	File to write the objects to
api_json	The json from gar_discovery_api
format	If TRUE will use tidy_eval on content

Value

TRUE if successful, side-effect creating filename

See Also

Other Google Discovery API functions: gar_create_api_skeleton, gar_create_package, gar_discovery_apis_list, gar_discovery_api

gar_create_api_skeleton *Create an API library skeleton*

Description

This will create a file with the skeleton of the API functions for the specified library

Usage

```
gar_create_api_skeleton(filename, api_json, format = TRUE)
```

Arguments

filename	R file to write skeleton to
api_json	The json from gar_discovery_api
format	If TRUE will use tidy_eval on content

Value

TRUE if successful, side effect will write a file

See Also

Other Google Discovery API functions: gar_create_api_objects, gar_create_package, gar_discovery_apis_list, gar_discovery_api

gar_create_package *Create a Google API package*

Description

Create a Google API package

Usage

```
gar_create_package(api_json, directory, rstudio = TRUE, check = TRUE,
  github = TRUE, format = TRUE, overwrite = TRUE)
```

Arguments

api_json	json from gar_discovery_api
directory	Where to build the package
rstudio	Passed to create, creates RStudio project file
check	Perform a check on the package once done
github	If TRUE will upload package to your github
format	If TRUE will use tidy_eval on content
overwrite	Whether to overwrite an existing directory if it exists

Details

For github upload to work you need to have your github PAT setup. See use_github.

Uses devtools' create to create a package structure then gar_create_api_skeleton and gar_create_api_objects to create starting files for a Google API package.

Value

If check is TRUE, the results of the CRAN check, else FALSE

See Also

https://developers.google.com/discovery/v1/reference/apis/list

A Github repository with https://github.com/MarkEdmondson1234/autoGoogleAPI generated by this function.

Other Google Discovery API functions: gar_create_api_objects, gar_create_api_skeleton, gar_discovery_apis_list, gar_discovery_api

---

gar_discovery_api *Get meta data details for specified Google API*

Description

Does not require authentication

Usage

```
gar_discovery_api(api, version)
```

Arguments

api	The API to fetch
version	The API version to fetch

Value

Details of the API

See Also

https://developers.google.com/discovery/v1/reference/apis/getRest

Other Google Discovery API functions: gar_create_api_objects, gar_create_api_skeleton, gar_create_package, gar_discovery_apis_list

---

gar_discovery_apis_list *Get a list of Google API libraries*

Description

Does not require authentication

Usage

```
gar_discovery_apis_list()
```

Value

List of Google APIs and their resources

See Also

https://developers.google.com/discovery/v1/reference/apis/list

Other Google Discovery API functions: gar_create_api_objects, gar_create_api_skeleton, gar_create_package, gar_discovery_api

---

gar_gce_auth *Authenticate on Google Compute Engine*

Description

This takes the metadata auth token in a Google Compute Engine instance as authentication source

Usage

```
gar_gce_auth(service_account = "default",
  client.id = getOption("googleAuthR.webapp.client_id"),
  client.secret = getOption("googleAuthR.webapp.client_secret"))
```

Arguments

service_account	Specify a different service account from the default
client.id	The Google Project API console's client Id
client.secret	The Google Project API console's client secret

Details

service_account is default or the service account email e.g. "service-account-key-json@projectname.iam.gservic

Google Compute Engine instances come with their own authentication tokens.

It has no refresh token so you need to call for a fresh token after approx. one hour. The metadata token will refresh itself when it has about 60 seconds left.

You can only use for scopes specified when creating the instance.

If you want to use them make sure their service account email is added to accounts you want to get data from.

If this function is called on a non-Google Compute Engine instance it will return NULL

Value

A token

See Also

gar_gce_auth_email

Other authentication functions: gar_attach_auto_auth, gar_auth_service, gar_auth, gar_auto_auth, get_google_token, token_exists

---

gar_gce_auth_email *Get the service email via GCE metadata*

Description

Get the service email via GCE metadata

Usage

```
gar_gce_auth_email(service_account = "default")
```

Arguments

service_account	Specify a different service account from the default Useful if you don't know the default email and need it for other uses

Value

the email address character string

See Also

gar_gce_auth

`gar_set_client` *Setup the clientId, clientSecret and scopes*

Description

Help setup the client ID and secret with the OAuth 2.0 clientID. Do not confuse with Service account keys.

Usage

```
gar_set_client(json = Sys.getenv("GAR_CLIENT_JSON"),
  web_json = Sys.getenv("GAR_CLIENT_WEB_JSON"), scopes = NULL)
```

Arguments

`json`	The file location of an OAuth 2.0 client ID json file
`web_json`	The file location of client ID json file for web applications
`scopes`	A character vector of scopes to set

Details

This function helps set the `options(googleAuthR.client_id)`, `options(googleAuthR.client_secret)` and `options(googleAuthR.scopes.selected)` for you.

You can also set the web application client IDs that are used in Shiny authentication, that are set via the options `options(googleAuthR.webapp.client_id)`, `options(googleAuthR.webapp.client_secret)`

Note that if you authenticate with a cache token with different values it will overwrite them.

For successful authentication, the API scopes can be browsed via the googleAuthR RStudio addin or the Google API documentation.

Do not confuse this JSON file with the service account keys, that are used to authenticate a service email. This JSON only sets up which app you are going to authenticate with - use gar_auth_service with the Service account keys JSON to perform the actual authentication.

By default the JSON file will be looked for in the location specified by the `"GAR_CLIENT_JSON"` environment argument, or via `"GAR_CLIENT_WEB_JSON"` for webapps.

Value

The `project-id` the app has been set for

Author(s)

Idea via @jennybc and @jimhester from `gargle and gmailr` libraries.

See Also

https://console.cloud.google.com/apis/credentials

Examples

```
## Not run:

gar_set_client("google-client.json", scopes = "http://www.googleapis.com/auth/webmasters")
gar_auth_service("google-service-auth.json")

## End(Not run)
```

---

gar_shiny_auth *Create Authentication within Shiny's server.R*

Description

This can be used at the top of the server function for authentication when you have used gar_shiny_ui to create a login page for your ui function.

Usage

```
gar_shiny_auth(session)
```

Arguments

session	Shiny session argument

Details

If using gar_shiny_ui, put this at the top of your server.R function

Author(s)

Based on a gist by Joe Cheng, RStudio

See Also

Other pre-load shiny authentication: gar_shiny_auth_url, gar_shiny_login_ui, gar_shiny_ui, silent_auth

Examples

```
## Not run:
library(shiny)
library(googleAuthR)
gar_set_client()

fileSearch <- function(query) {
  googleAuthR::gar_api_generator("https://www.googleapis.com/drive/v3/files/",
                                 "GET",
                                 pars_args=list(q=query),
                                 data_parse_function = function(x) x$files)()
}

## ui.R
ui <- fluidPage(title = "googleAuthR Shiny Demo",
                textInput("query",
                label = "Google Drive query",
                value = "mimeType != 'application/vnd.google-apps.folder'"),
                tableOutput("gdrive")
                )

## server.R
server <- function(input, output, session){

# this is not reactive, no need as you only reach here authenticated
gar_shiny_auth(session)

output$gdrive <- renderTable({
  req(input$query)

  # no need for with_shiny()
  fileSearch(input$query)

  })
  }

# gar_shiny_ui() needs to wrap the ui you have created above.
shinyApp(gar_shiny_ui(ui), server)

## End(Not run)
```

`gar_shiny_auth_url` *Make a Google Authorisation URL for Shiny*

Description

Set this within your login_ui where you need the Google login.

Usage

```
gar_shiny_auth_url(req, state = getOption("googleAuthR.securitycode"),
  client.id = getOption("googleAuthR.webapp.client_id"),
  client.secret = getOption("googleAuthR.webapp.client_secret"),
  scope = getOption("googleAuthR.scopes.selected"),
  access_type = c("online", "offline"), approval_prompt = c("auto",
  "force"))
```

Arguments

req	a Rook request, do not set as this will be used by Shiny to generate URL
state	URL state
client.id	client.id
client.secret	client.secret
scope	API scopes
access_type	whether to keep the token
approval_prompt	Auto-login if user is recognised or always force signin

See Also

Other pre-load shiny authentication: gar_shiny_auth, gar_shiny_login_ui, gar_shiny_ui, silent_auth

---

gar_shiny_login_ui *A login page for Shiny*

Description

An alternative to the immediate login provided by default by gar_shiny_ui

Usage

```
gar_shiny_login_ui(req, title = "googleAuthR Login Demo")
```

Arguments

req	Passed to gar_shiny_auth_url to generate login URL
title	The title of the page

Details

Use gar_shiny_auth_url to create the login URL. You must leave the first argument free as this is used to generate the login, but you can pass other arguments to customise your UI.

See Also

Other pre-load shiny authentication: gar_shiny_auth_url, gar_shiny_auth, gar_shiny_ui, silent_auth

gar_shiny_ui *Create a Google login before your Shiny UI launches*

Description

A function that will turn your ui object into one that will look for Google authentication before loading the main app. Use together with gar_shiny_auth

Usage

```
gar_shiny_ui(ui, login_ui = silent_auth)
```

Arguments

ui	A Shiny ui object
login_ui	A UI or HTML template that is seen before the main app and contains a login in link generated by gar_shiny_auth_url

Details

Put this at the bottom of your ui.R or pass into shinyApp wrapping your created ui.

Author(s)

Based on a gist by Joe Cheng, RStudio

See Also

Other pre-load shiny authentication: gar_shiny_auth_url, gar_shiny_auth, gar_shiny_login_ui, silent_auth

Examples

```
## Not run:
library(shiny)
library(googleAuthR)
gar_set_client()

fileSearch <- function(query) {
  googleAuthR::gar_api_generator("https://www.googleapis.com/drive/v3/files/",
                                 "GET",
                                 pars_args=list(q=query),
                                 data_parse_function = function(x) x$files)()
}

## ui.R
ui <- fluidPage(title = "googleAuthR Shiny Demo",
                textInput("query",
```

```
                label = "Google Drive query",
                value = "mimeType != 'application/vnd.google-apps.folder'"),
                tableOutput("gdrive")
                )

## server.R
server <- function(input, output, session){

# this is not reactive, no need as you only reach here authenticated
gar_shiny_auth(session)

output$gdrive <- renderTable({
  req(input$query)

  # no need for with_shiny()
  fileSearch(input$query)

  })
  }

# gar_shiny_ui() needs to wrap the ui you have created above.
shinyApp(gar_shiny_ui(ui), server)

## End(Not run)
```

`gar_token_info` *Get current token summary*

Description

Get details on the current active auth token to help debug issues

Usage

```
gar_token_info(detail_level = getOption("googleAuthR.verbose", default = 3))
```

Arguments

`detail_level`	How much info to show

`googleAuth` *Shiny Google Authorisation [Server Module]*

Description

Server part of shiny module, use with googleAuthUI

Usage

```
googleAuth(input, output, session, login_text = "Login via Google",
  logout_text = "Logout", login_class = "btn btn-primary",
  logout_class = "btn btn-default", access_type = c("online", "offline"),
  approval_prompt = c("auto", "force"), revoke = FALSE)
```

Arguments

input	shiny input
output	shiny output
session	shiny session
login_text	What the login text will read on the button
logout_text	What the logout text will read on the button
login_class	The CSS class for the login link
logout_class	The CSS class for the logout link
access_type	Online or offline access for the authentication URL
approval_prompt	Whether to show the consent screen on authentication
revoke	If TRUE a user on logout will need to re-authenticate

Details

Call via `shiny::callModule(googleAuth, "your_ui_name", login_text = "Login")`

Value

A reactive authentication token

See Also

Other shiny module functions: googleAuthUI

Examples

```
## Not run:
options("googleAuthR.scopes.selected" =
  c("https://www.googleapis.com/auth/urlshortener"))

shorten_url <- function(url){
  body = list(
    longUrl = url
 )

 f <-
   gar_api_generator("https://www.googleapis.com/urlshortener/v1/url",
                     "POST",
                     data_parse_function = function(x) x$id)
```

```
 f(the_body = body)

 }

server <- function(input, output, session){

  ## Create access token and render login button
  access_token <- callModule(googleAuth,
                             "loginButton",
                             login_text = "Login1")

  short_url_output <- eventReactive(input$submit, {
    ## wrap existing function with_shiny
    ## pass the reactive token in shiny_access_token
    ## pass other named arguments
    with_shiny(f = shorten_url,
               shiny_access_token = access_token(),
               url=input$url)
  })

  output$short_url <- renderText({

    short_url_output()

  })

}

## ui
ui <- fluidPage(
  googleAuthUI("loginButton"),
  textInput("url", "Enter URL"),
  actionButton("submit", "Shorten URL"),
  textOutput("short_url")
)

shinyApp(ui = ui, server = server)

## End(Not run)
```

---

googleAuthR *googleAuthR: Easy Authentication with Google OAuth2 APIs*

Description

Get more details on the googleAuthR website.

Default options

These are the default options that you can override via `options()`

- `googleAuthR.batch_endpoint = "https://www.googleapis.com/batch"`
- `googleAuthR.rawResponse = FALSE`
- `googleAuthR.httr_oauth_cache = ".httr-oauth"`
- `googleAuthR.verbose = 3`
- `googleAuthR.client_id = NULL`
- `googleAuthR.client_secret = NULL`
- `googleAuthR.webapp.client_id = NULL`
- `googleAuthR.webapp.client_secret = NULL`
- `googleAuthR.webapp.port = 1221`
- `googleAuthR.jsonlite.simplifyVector = TRUE`
- `googleAuthR.scopes.selected = NULL`
- `googleAuthR.ok_content_types=c("application/json; charset=UTF-8", ("text/html; charset=UTF-8"))`
- `googleAuthR.securitycode = paste0(sample(c(1:9, LETTERS, letters), 20, replace = T), collapse='')`
- `googleAuthR.tryAttempts = 5`

---

googleAuthUI *Shiny Google Authorisation [UI Module]*

Description

UI part of shiny module, use with googleAuth

Usage

```
googleAuthUI(id)
```

Arguments

id	shiny id

Value

A shiny UI for logging in

See Also

Other shiny module functions: googleAuth

googleSignIn *Google SignIn [Server Module]*

Description

Shiny Module for use with googleSignInUI. Use when you don't need to call APIs, but would like a login to Shiny.

Usage

```
googleSignIn(input, output, session)
```

Arguments

input	shiny input
output	shiny output
session	shiny session

Details

Call via `shiny::callModule(googleSignIn, "your_id")`

Value

A reactive list with values $g_id, $g_name, $g_email and $g_image

Author(s)

Based on original code by David Kulp

googleSignInUI *Google SignIn [UI Module]*

Description

Shiny Module for use with googleSignIn. If you just want a login to a Shiny app, without API tokens.

Usage

```
googleSignInUI(id)
```

Arguments

id	Shiny id

Value

Shiny UI

Author(s)

Based on original code by David Kulp

See Also

https://github.com/dkulp2/Google-Sign-In

---

silent_auth *Silent auth*

Description

The default for logging in via gar_shiny_ui, this creates no login page and just takes you straight to authentication on Shiny app load.

Usage

```
silent_auth(req)
```

Arguments

req	What Shiny uses to check the URL parameters

See Also

Other pre-load shiny authentication: gar_shiny_auth_url, gar_shiny_auth, gar_shiny_login_ui, gar_shiny_ui

---

skip_if_no_env_auth *Skip test if not authenticated*

Description

Use within tests to skip if a local authentication file isn't available through an environment variable.

Usage

```
skip_if_no_env_auth(env_arg)
```

Arguments

env_arg	The name of the environment argument pointing to the auth file

`with_shiny` *Turn a googleAuthR data fetch function into a Shiny compatible one*

Description

Turn a googleAuthR data fetch function into a Shiny compatible one

Usage

```
with_shiny(f, shiny_access_token = NULL, ...)
```

Arguments

`f`	A function generated by `googleAuth_fetch_generator`.
`shiny_access_token`	A reactive object that resolves to a token.
`...`	Other arguments passed to f.

Value

the function f with an extra parameter, shiny_access_token=NULL.

Examples

```
## Not run:
## in global.R

## create the API call function, example with goo.gl URL shortner
library(googleAuthR)
options("googleAuthR.scopes.selected" = c("https://www.googleapis.com/auth/urlshortener"))

shorten_url <- function(url){

  body = list(
    longUrl = url
 )

 f <- gar_api_generator("https://www.googleapis.com/urlshortener/v1/url",
                        "POST",
                        data_parse_function = function(x) x$id)

 f(the_body = body)

 }


## in server.R
library(shiny)
library(googleAuthR)
```

```
source('global.R')

shinyServer(function(input, output, session)){

  ## Get auth code from return URL
  access_token  <- reactiveAccessToken(session)

  ## Make a loginButton to display using loginOutput
  output$loginButton <- renderLogin(session, access_token())

  short_url_output <- eventReactive(input$submit, {
  ## wrap existing function with_shiny
  ## pass the reactive token in shiny_access_token
  ## pass other named arguments
    short_url <- with_shiny(f = shorten_url,
                            shiny_access_token = access_token(),
                            url=input$url)

   })

   output$short_url <- renderText({

     short_url_output()

   })
 }

## in ui.R
library(shiny)
library(googleAuthR)

shinyUI(
  fluidPage(
    loginOutput("loginButton"),
    textInput("url", "Enter URL"),
    actionButton("submit", "Shorten URL"),
    textOutput("short_url")
    ))

## End(Not run)
```

Index