Package 'googleAuthR'

March 31, 2017

Type Package

Version 0.5.1

Title Easy Authentication with Google OAuth2 API

Description Create R functions that interact with OAuth2 Google APIs easily, with auto-refresh and Shiny compatibility.

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

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

Depends R (>= 3.2.0)

Imports devtools (>= 1.12.0), httr (>= 1.1.0), jsonlite (>= 0.9.16), R6 (>= 2.1.0), shiny (>= 0.13.2), testthat (>= 1.0.2), formatR (>= 1.4), miniUI (>= 0.1.1)

Suggests roxygen2 (>= 5.0.0), covr, rmarkdown, knitr

License MIT + file LICENSE

LazyData true

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation no

Author Mark Edmondson [aut, cre], Jennifer Bryan [ctb], Johann deBoer [ctb]

Maintainer Mark Edmondson <m@sunholo.com>

Repository CRAN

Date/Publication 2017-03-31 10:11:27 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.

check_cached_scopes *Check token scopes*

Description

Check token scopes

Usage

```
check_cached_scopes()
```

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"))
```

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 `httr::use_proxy` or `httr::add_headers` that will be added to the request.
simplifyVector	Passed to jsonlite::fromJSON for response parsing

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_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 = "TRAVIS_GAR_AUTH_FILE")
```

Arguments

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

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

`travis_environment_var`
: Name of Travis environment var that contains auth file path

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.

Travis

If you are using Travis to make tests, then a specific environment name for a Travis auth file is also needed, that should be relative to the home directory of your Github repository.

You should then also encrypt the auth file and include the encrypted file in your Github repository. See googlesheets vignette on managing auth tokens and Travis encrypting files how-to for background.

To work on travis, you will need one auth token for the library load in the home folder, and another in the `testthat` folder. You can achieve this by using the same encrypted file command twice within your `.travis.yml` configuration, writing out to the two different folders:

```
openssl aes-256-cbc -K $encrypted_0a6446eb3ae3_key -iv $encrypted_0a6446eb3ae3_key -in auth.json.enc
openssl aes-256-cbc -K $encrypted_0a6446eb3ae3_key -iv $encrypted_0a6446eb3ae3_key -in auth.json.enc
```

See Also

Other authentication functions: gar_auth_service, gar_auth, gar_auto_auth, gar_gce_auth, get_google_token, is_legit_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, is_legit_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(input, output, session)
```

Arguments

input	shiny input
output	shiny output
session	shiny session

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(id, login_class = "btn btn-primary",
  logout_class = "btn btn-danger", login_text = "Log In",
  logout_text = "Log Out")
```

Arguments

id	Shiny id
login_class	CSS class of login button
logout_class	CSS class ofr logout button
login_text	Text to show on login button
logout_text	Text to show on logout button

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, is_legit_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 = "TRAVIS_GAR_AUTH_FILE")
```

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

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

You can use this in your code to authenticate from a filelocation specified in 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.

`travis_environment_var` Name that will be called via Sys.getenv in Travis tests. As the working directory is different on travis, this environment variable should contain a relative path to your Github repository home folder.

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, is_legit_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, ...)
```

Arguments

call_list	a list of functions from gar_api_generator
...	further arguments passed to the data parse function of f

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 seperate 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: applyDataParseFunction, doBatchRequest, gar_batch_walk, makeBatchRequest, parseBatchResponse

gar_batch_walk *Walk data through batches*

Description

Convienience 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, ...)
```

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

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.

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: applyDataParseFunction, doBatchRequest, gar_batch, makeBatchRequest, parseBatchResponse

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

Doesn't 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

Doesn't 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_gadget *Gadget for easy authentication*

Description

Gadget for easy authentication

Usage

```
gar_gadget()
```

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	From the Google API console.
client.secret	From the Google API console.

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

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

---

gar_shiny_getUrl *Get the Shiny Apps URL.*

Description

Needed for the redirect URL in Google Auth flow

Usage

```
gar_shiny_getUrl(session)
```

Arguments

session	The shiny session object.

Value

The URL of the Shiny App its called from.

See Also

Other shiny auth functions: authReturnCode, createCode, gar_shiny_getAuthUrl, gar_shiny_getToken, loginOutput, reactiveAccessToken, renderLogin, revokeEventObserver, with_shiny

---

gar_token_info *Get current token summary*

Description

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

Usage

```
gar_token_info()
```

googleAuth *Server side google auth (Shiny 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 a startup guide by viewing the vignette: `vignette("googleAuthR")`

Details

There are two main functions in the googleAuthR package:

- `gar_auth` provides local authentication token.
- `gar_api_generator` A function factory for easy enabling of Google API functions.

Batching

If you have many API calls, you can save a lot of time by using batching. This takes your many calls and sends them in one POST request to `www.googleapis.com/batch`, returning a list of any responses. See `gar_batch` for details.

Another common batch task is to call the same function with one parameter changing each call. This is supported using the `gar_batch_walk` function.

Shiny functions

If you need Shiny authentication, then these functions work together to give a smooth authentication flow.

- `googleAuthUI` provides the Shiny UI via Shiny modules
- `googleAuth` provides the Shiny server via Shiny modules
- `with_shiny` Wrap the functions you created with gar_api_generator with this so you can pass the `reactiveAccessToken`

Default options

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

- `googleAuthR.rawResponse = FALSE`
- `googleAuthR.httr_oauth_cache = TRUE`
- `googleAuthR.verbose = 3`
- `googleAuthR.client_id = "201908948134-rm1ij8ursrfcbkv9koc0aqver84b04r7.apps.googleusercontent.c`
- `googleAuthR.client_secret = "nksRJZ5K3nm9FUWsAtBoBArz"`

- `googleAuthR.webapp.client_id = "201908948134-cjjs89cffh3k429vi7943ftpk3jg36ed.apps.googleuserco`
- `googleAuthR.webapp.client_secret = "mE7rHl0-iNtzyI1MQia-mg1o"`
- `googleAuthR.webapp.port = 1221`
- `googleAuthR.jsonlite.simplifyVector = TRUE`
- `googleAuthR.scopes.selected = c("https://www.googleapis.com/auth/webmasters",`
- `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` *A Login button (Shiny 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`

loginOutput *Login/logout Shiny output*

Description

USe within a ui.R to render the login button generated by `renderLogin`

Usage

```
loginOutput(output_name)
```

Arguments

`output_name`	Name of what output object was assigned in `renderLogin`

Value

A login/logout button in a Shiny app

See Also

Other shiny auth functions: `authReturnCode`, `createCode`, `gar_shiny_getAuthUrl`, `gar_shiny_getToken`, `gar_shiny_getUrl`, `reactiveAccessToken`, `renderLogin`, `revokeEventObserver`, `with_shiny`

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)
```

---

myMessage *Customer message log level*

Description

Customer message log level

Usage

```
myMessage(..., level = 2)
```

Arguments

...	The message(s)
level	The severity

Details

0 = everything, 1 = debug, 2=normal, 3=important

---

reactiveAccessToken *Create a reactive Google OAuth2 token*

Description

Use within a Shiny server.R session to create the access token passed to all Google API functions using with_shiny

Usage

```
reactiveAccessToken(session)
```

Arguments

session	A Shiny session object.

Value

A reactive Google auth token

See Also

Other shiny auth functions: authReturnCode, createCode, gar_shiny_getAuthUrl, gar_shiny_getToken, gar_shiny_getUrl, loginOutput, renderLogin, revokeEventObserver, with_shiny

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)
```

`renderLogin` *Render a Google API Authentication Login/logout button*

Description

Use within a Shiny server.R to assign to an output for ui.R. The login button carries an ActionLink with value "signed_in" but as Shiny reloads on pushing it can't be used for detection of login state. Use `!is.null(access_token())` instead.

Usage

```
renderLogin(session, access_token, 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

`session`	A Shiny session object
`access_token`	A token generated by `reactiveAccessToken`
`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 Bootstrap class for the login link
`logout_class`	The Bootstrap 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.

Value

An object to assign to output e.g. output$login

See Also

revokeEventObserver

Other shiny auth functions: `authReturnCode`, `createCode`, `gar_shiny_getAuthUrl`, `gar_shiny_getToken`, `gar_shiny_getUrl`, `loginOutput`, `reactiveAccessToken`, `revokeEventObserver`, `with_shiny`

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
  ## revoke=TRUE means upon logout a user will need to reauthenticate
  output$loginButton <- renderLogin(session, access_token(), revoke=TRUE)

  ## Needed if revoke=TRUE above
  revokeEventObserver(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)
```

`revokeEventObserver` *Listens for a user revoking authentication*

Description

If the parameter `revoke` is set to TRUE for renderLogin then this observer is also required in the Shiny server to do the revoking.

Usage

```
revokeEventObserver(access_token, input)
```

Arguments

`access_token`	A token generated by `reactiveAccessToken`.
`input`	the input object from a shinyServer function.

See Also

renderLogin

Other shiny auth functions: `authReturnCode`, `createCode`, `gar_shiny_getAuthUrl`, `gar_shiny_getToken`, `gar_shiny_getUrl`, `loginOutput`, `reactiveAccessToken`, `renderLogin`, `with_shiny`

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
  ## revoke=TRUE means upon logout a user will need to reauthenticate
  output$loginButton <- renderLogin(session, access_token(), revoke=TRUE)

  ## Needed if revoke=TRUE above
  revokeEventObserver(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)
```

`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 token generated within a `gar_shiny_getToken`.
`...`	Other arguments passed to f.

Value

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

See Also

Other shiny auth functions: `authReturnCode`, `createCode`, `gar_shiny_getAuthUrl`, `gar_shiny_getToken`, `gar_shiny_getUrl`, `loginOutput`, `reactiveAccessToken`, `renderLogin`, `revokeEventObserver`

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