Package 'googleAuthR'

August 19, 2015

Type Package

Version 0.1.1

Title Easy Authentication with Google OAuth2 APIs

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

URL https://github.com/MarkEdmondson1234/googleAuthR

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

Depends R (>= 3.2.0)

Imports httr (>= 1.0.0), jsonlite (>= 0.9.16), R6 (>= 2.1.0)

Suggests shiny (>= 0.12.1), knitr

License MIT + file LICENSE

LazyData true

VignetteBuilder knitr

NeedsCompilation no

Author Mark Edmondson [aut, cre]

Maintainer Mark Edmondson <m@sunholo.com>

Repository CRAN

Date/Publication 2015-08-19 17:28:18

R topics documented:

Authentication *R6 environment to store authentication credentials*

Description

Used to keep persistent state.

Usage

```
Authentication
```

Format

```
Class 'R6ClassGenerator' <environment: 0x10b913460>
 - attr(*, "name")= chr "Authentication_generator"
```

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)
```

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. **path_args** and **pars_args** add default values to the baseURI. You don't need to supply access_token for OAuth2 requests in pars_args, this is dealt with in gar_auth()

Value

A function that can fetch the Google API data you specify

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, verbose = TRUE)
```

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.
verbose	Increase feedback messages of the function.

Details

Most users, most of the time, do not need to call this function explicitly – it will be triggered by the first action that requires authorization. Even when called, the default arguments will often suffice. However, when necessary, this function allows the user to

- store a token – the token is invisibly returned and can be assigned to an object or written to an .rds file
- read the token from an .rds file or pre-existing object in the workspace
- provide your own app key and secret – this requires setting up a new project in Google Developers Console
- prevent caching of credentials in .httr-oauth

In a call to scr_auth, the user can provide the token, app key and secret explicitly and can dictate whether credentials will be cached in .httr_oauth. If unspecified, 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

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

`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.

Shiny functions

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

- `reactiveAccessToken` provides the Shiny authentication token.
- `renderLogin` Creates the login button server.R side.
- `loginOutput` Creates the login button ui.R side.
- `with_shiny` Wrap the functions you created with gar_api_generator with this so you can pass the `reactiveAccessToken`

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`; `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)
```

---

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; 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".

Usage

```
renderLogin(session, access_token, login_text = "Login via Google",
  logout_text = "Logout", login_class = "btn btn-primary",
  logout_class = "btn btn-default")
```

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

Value

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

See Also

Other shiny auth functions: authReturnCode; createCode; gar_shiny_getAuthUrl; gar_shiny_getToken; gar_shiny_getUrl; loginOutput; reactiveAccessToken; 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)
```

`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

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