## R and Restful APIs

jim rothstein

API	and Restful APIs
	API is way to open a close software system in specific ways and to specific
	users.

## Restful Examples

List of youtube videos in my Documentaries playlist

 $"000\_httr\_youtube\_playlist\_TALK.pdf"$ 

[documentaries] (000\_httr\_youtube\_playlist\_TALK.pdf)

► Check politican's donors

- Create Github Gist To obtain my gists (only) "'zsh export token=\$(Rscript -e "cat(Sys.getenv('GITHUB\_PAT'))") curl -s -H "Authorization: token \$token" -H "Accept: application/vnd.github.v3+json" https://api.github.com/gists ## This message is from <!-Cursory, limited explanation of Restful **APIs** 

links for reader.

#### **DEFINITION**

(from Wikipedia)https:

- //en.wikipedia.org/wiki/Representational\_state\_transfer#Applied\_to\_web\_services
  Web service APIs that adhere to the REST architectural constraints are called
  RESTful APIs.[12] HTTP-based RESTful APIs are defined with the following
  aspects:[13]
  - a base URI, such as http://api.example.com/;
  - standard HTTP methods (e.g., GET, POST, PUT, and DELETE);
  - a media type that defines state transition data elements (e.g., Atom, microformats, application/vnd.collection+json,[13]: 91–99 etc.). The current representation tells the client how to compose requests for transitions to all the next available application states. This could be as simple as a URI or as complex as a Java applet.[14]



No R libraries (wrapper) for Github API

Goal: Use R to programmatically retrieve data for analysis, securely.

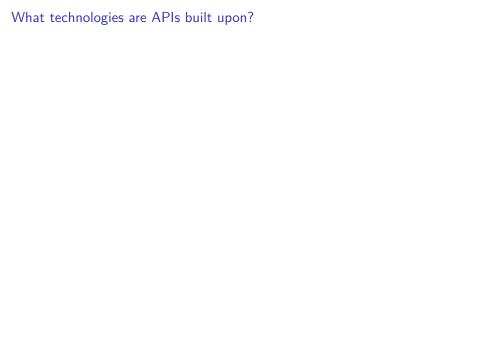
#### First, understand some of the models

- ► HTTP messages, GET, POST
- headers, body, URI
- Diagram: HTTP
- Tools: curl, Postman and many others
- EXAMPLES: Using CURL
- https://docs.github.com/en/rest/guides/getting-started-with-the-rest-api

I appear before a chunk!

curl -s https://api.github.com/zen
## Responsive is better than fast.

I am after a chunk...



## **Technologies**

- ► HTTP and related (JSON, ssl ...)
- ► OAUTH2 and related

HTTP Diagram

Diagram for GET, POST messages, with HEADERS  $\,$ 

#### API documenation

#### Endpoints:

 $https://docs.github.com/en/rest/overview/endpoints-available-for-github-apps \\ https://developers.google.com/youtube/v3/docs/playlists/list —$ 

CURL intro (using httpbin.org) do in browser, then in curl compare: curl -is httpbin.org vs httpbin.org/get

browser: show curl

curl -sv http://httpbin.org



## We know some of the plumbing: HTTP, API documentation, CURL ...

#### Finally, Do this in R, httr2:

```
https://httr2.r-lib.org show httpbin.org show github.com ## EXAMPLE: Basic GET in R
```

#### Simple GET

```
## return what is sent
library(httr2)
req <- request("https://httpbin.org")
req
    ## <httr2_request>
    ## GET https://httpbin.org
    ## Body: empty
## Dry run
```

```
req |> req_dry_run()
    ## GET / HTTP/1.1
    ## Host: httpbin.org
    ## User-Agent: httr2/0.1.1 r-curl/4.3.2 libcurl/7.58.0
    ## Accept: */*
    ## Accept-Encoding: deflate, gzip
```

#### Run it

```
resp <- req |> req_perform()
resp

## <httr2_response>
## GET https://httpbin.org/
## Status: 200 OK
## Content-Type: text/html
## Body: In memory (9593 bytes)
```

#### Body

## [2] < body > n <a href="https://qithub.com/requests/httpbin" class="qi

#### Add header

```
req |>
    req_headers(name = "jim", location = quote(eugene) ) |>
    req_headers("ACCEPT" = "application/json") |>
    req_dry_run()
    ## GET / HTTP/1.1
```

#### Add query string

```
req2 <- request("https://httpbin.org/get?q=joe")</pre>
req2
req %>% req_dry_run()
```

#### Add a body, in json

```
req |>
  req_body_json(list(x=1, friend="joe")) |>
  req_dry_run()
    ## POST /get HTTP/1.1
    ## Host: httpbin.org
    ## User-Agent: httr2/0.1.1 r-curl/4.3.2 libcurl/7.58.0
    ## Accept: */*
    ## Accept-Encoding: deflate, gzip
```

▶ But how to include security?

## GITHUB Personal Access Token (PAT)

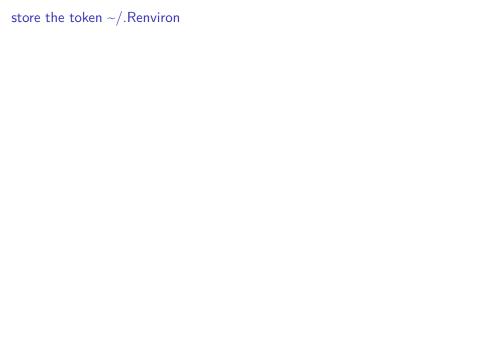
#### Github Personal Access Token (PAT)

- ▶ Replacement for password
- ▶ Scope: controls what resources , time limits
- EXAMPLE: Github token (Personal Access Token) (walk through)
- ▶ Read the Documentation
- ▶ SCOPE
- Back to CURL

# Github PAT token Why a token?

settings | developer settings (bottom left)

```
more info: get token:
https://docs.github.com/en/rest/guides/getting-started-with-the-rest-api
https://docs.github.com/en/rest
export token=$(Rscript -e "cat(Sys.getenv('GITHUB_PAT'))")
curl -si -u jimrothstein: $token https://api.github.com/users/jimrothstein
head
      ## etag: "cb16c57e3b7c5de05e16d618199090630c76791353f00960ab424cec3f9e12dd
      ## x-oauth-scopes: gist, notifications, read:discussion, read:org, read:pa
```



## token, works export token=\$(Rscript -e "cat(Sys.getenv('GITHUB\_PAT'))") echo \$token

```
curl -Hs "Authorization: token $token" https://api.github.com/users/codertocat
                                                                          Time
                                                                          Left
```

## etag: W/"b9387c3fdac0d6bc9d6055f6e84379cdc2cdb8bfbd67bdd727eb1e69<mark>9bd585</mark> ## access-control-expose-headers: ETag, Link, Location, Retry-After, X-Git

## strict-transport-security: max-age=31536000; includeSubdomains; preload

## referrer-policy: origin-when-cross-origin, strict-origin-when-cross-ori

There is another side to all this: the Creater of APIs.

R has a tool plumber. Not for today.

## One issue not to overlook: 3rd party Security

- Rough analogy: Valet Key
- Specifically, \*\*Authentication\*\* and "Authorization"
- 3 device model
- notion of 1 party trusted by other two, without revealing passwords.

So far, the user has been you - the Developer. What if you want others to

use app? They are going to give their username/password to your app.

▶ OAUTH2

EXAMPLE Use Github uses token (PAT) in R.

New problem: each API server is different!

(Quote: httr2:: docs)

EXAMPLES: Many (follow httr2:: vignette) + Google, Youtube ...

How to WRITE an API: openAPI

## Attempt to simplify and standardize how the developer determines the $\ensuremath{\mathsf{API}}$

structure.

https://oai.github.io/Documentation/start-here.html

The OAS defines a standard, programming languageagnostic interface description for REST APIs, which allows both humans and comp level programming, the OAS removes guesswork in calling a service.

### R Packages

- httr2 (rewrite of httr)
- curl
- plumber (server in R)

R Packages you may want to evaluate (but I did not use)

- gargle