CIS 8398
Advanced AI Topics in Business

#API

Yu-Kai Lin

Agenda

- What are APIs?
- Accessing APIs from R
  - Without an API wrapper
  - With an API wrapper

[**Acknowledgements**] The materials in the following slides are based on the source(s) below:

- An introduction to APIs by Brian Cooksey
- What is an API? In English, please by Petr Gazarov
- purrr tutorial by Jennifer Bryan
- How to obtain a bunch of GitHub issues or pull requests with R by Jennifer Bryan

Prerequisites

- httr: Tools for Working with URLs and HTTP
- purrr: A complete and consistent functional programming toolkit for R (included in `tidyverse`)
- gh: Minimalistic GitHub API client in R
- devtools: devtools allows you to install R packages from GitHub. This is useful when the development version of a package fixes some bugs or offers additional features.

```
install.packages(c("httr", "gh", "devtools"))

#if you want to install the development version from GitHub
#devtools::install_github("r-lib/gh")

library(httr)
library(gh)
library(tidyverse)
```

What an API is and why it's valuable

APIs (application programming interfaces) are a big part of the web.

Most modern websites consume at least some third-party APIs.

- Saves time and efforts and makes developers more productive by easily mixing different services
- When a company offers an API to their customers, it just means that they've built a set of dedicated URLs that return **pure data responses**—meaning the responses won't contain the kind of presentational overhead that you would expect in a graphical user interface like a website.

The protocol of the web

Web APIs usually use HTTP to transfer data between client and server.

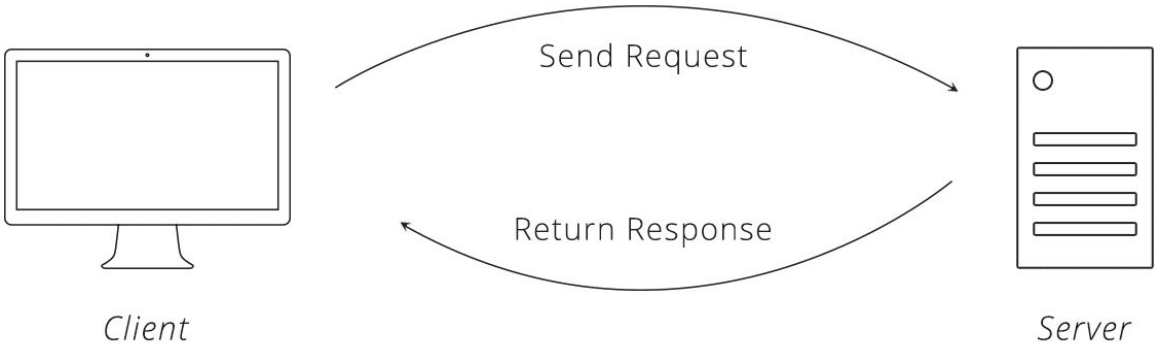
- When you call the API, you are making an HTTP request with some parameters and/or authentication information.
- Once server gets the request, it will send data back to you through an HTTP response. However, the HTTP body can be JSON, XML, or other formats depending on APIs.

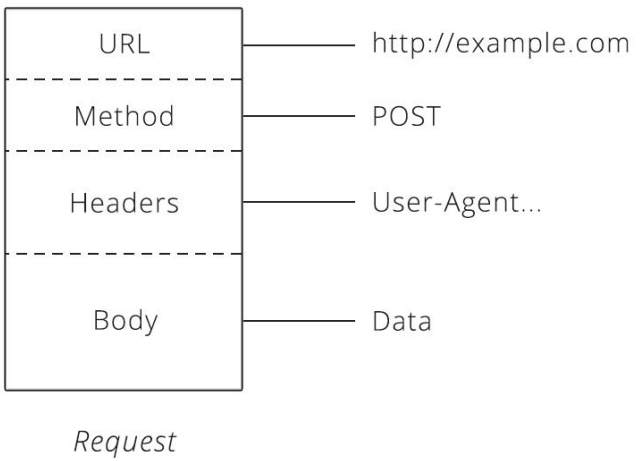
Why authentication?

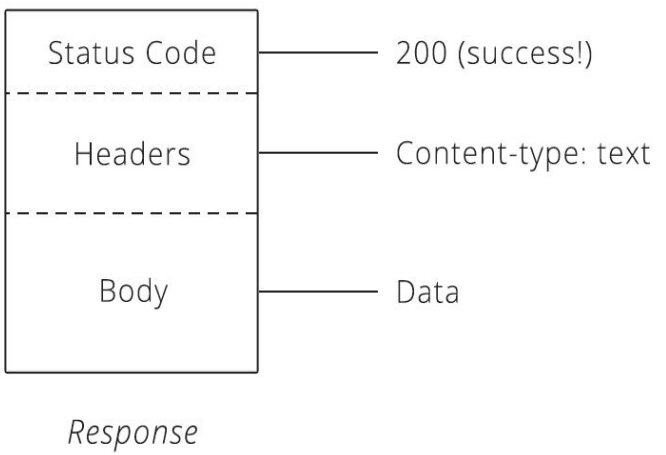
Many APIs require some sort of Authentication before you can use it.

- To provide access to, or control over, sensitive data
- To limit the number of API calls per user

HTTP request and response







HTTP request methods

The four methods most commonly seen in APIs are:

- **GET** - Asks the server to retrieve a resource
- **POST** - Asks the server to create a new resource
- **PUT** - Asks the server to edit/update an existing resource
- **DELETE** - Asks the server to delete a resource

Take the GitHub API for example. You can use the

- **GET** method to retrieve a list of public repositories in someone's account
- **POST** method to create a new repository in your account
- **PUT** method to update a file in your repository
- **DELETE** method to delete a file in your repository

Obviously, you cannot use **POST**, **PUT**, and **DELETE** on other users' repository since you do not have the authentication.

Accessing APIs from R

Let's take a look at this API call: https://api.github.com/repos/tidyverse/ggplot2

You can access this on your browser.

Accessing APIs from R is essentially asking R to issue some HTTP requests for you.

To make HTTP requests, the easiest way is to use the `httr` package. `httr` contains one function for every HTTP verb. The functions have the same names as the verbs (e.g. `GET()`, `POST()`).

```
#install.packages("httr")
library(httr)
response = GET(url = "https://api.github.com/repos/tidyverse/ggplot2")
class(response)
```

```
## [1] "response"
```

```
str(response, max.level = 1) # use max.level = 1 to avoid excessive output
```

```
## List of 10
##  $ url        : chr "https://api.github.com/repos/tidyverse/ggplot2"
##  $ status_code: int 200
##  $ headers    :List of 26
##   ..- attr(*, "class")= chr [1:2] "insensitive" "list"
##  $ all_headers:List of 1
##  $ cookies    :'data.frame':    0 obs. of  7 variables:
##  $ content    : raw [1:6713] 7b 0a 20 20 ...
##  $ date       : POSIXct[1:1], format: "2023-10-18 15:43:03"
##  $ times      : Named num [1:6] 0 0.0716 1.1056 1.1341 1.248 ...
##   ..- attr(*, "names")= chr [1:6] "redirect" "namelookup" "connect" "pretransfer"
##  $ request    :List of 7
##   ..- attr(*, "class")= chr "request"
##  $ handle     :Class 'curl_handle' <externalptr>
##  - attr(*, "class")= chr "response"
```

```
response_content = content(response)
```

```
str(response_content,
    max.level = 1, nchar.max=17)
```

```
## List of 83
##  $ id                    : int 19438
##  $ node_id               : chr ""| __truncated__
##  $ name                  : chr "ggplot2"
##  $ full_name             : chr ""| __truncated__
##  $ private               : logi FALSE
##  $ owner                 :List of 18
##  $ html_url              : chr ""| __truncated__
##  $ description           : chr ""| __truncated__
##  $ fork                  : logi FALSE
##  $ url                   : chr ""| __truncated__
##  $ forks_url             : chr ""| __truncated__
##  $ keys_url              : chr ""| __truncated__
##  $ collaborators_url     : chr ""| __truncated__
##  $ teams_url             : chr ""| __truncated__
##  $ hooks_url             : chr ""| __truncated__
##  $ issue_events_url      : chr ""| __truncated__
##  $ events_url            : chr ""| __truncated__
##  $ assignees_url         : chr ""| __truncated__
##  $ branches_url          : chr ""| __truncated__
##  $ tags_url              : chr ""| __truncated__
##  $ blobs_url             : chr ""| __truncated__
##  $ git_tags_url          : chr ""| __truncated__
##  $ git_refs_url          : chr ""| __truncated__
##  $ trees_url             : chr ""| __truncated__
##  $ statuses_url          : chr ""| __truncated__
##  $ languages_url         : chr ""| __truncated__
##  $ stargazers_url        : chr ""| __truncated__
##  $ contributors_url      : chr ""| __truncated__
```

```
response_content
```

```
## $id
## [1] 19438
##
## $node_id
## [1] "MDEwOlJlcG9zaXRvcnkxOTQzOA=="
## $name
## [1] "ggplot2"
## $full_name
## [1] "tidyverse/ggplot2"
## $private
## [1] FALSE
## $owner
## [1] "tidyverse"
## $owner$id
## [1] 22032646
## $owner$node_id
## [1] "MDEyOk9yZ2FuaXphdGlvbjIyMDMyNjQ2"
## $owner$avatar_url
## [1] "https://avatars.githubusercontent.com/u/22032
## $owner$gravatar_id
## [1] ""
```

Your turn

Take a closer look at the GitHub API reference documentation.

Try to find the right API URLs for the following data:

- User `hadley`'s profile
- List of `hadley`'s repositories
- List of users who are following `hadley`
- Members of `Google` on GitHub

Once you have these URLs, use `httr` to collect these data.

Tip: you can actually test/verify whether you have a right API URL by going to that URL in your web browser.

Adding parameters to an API query

Most APIs allow you to include parameters in your API calls. You need to review the API documentation to understand what parameters are allowed.

Example: In GitHub's repositories endpoint, we can list list repositories for a user with the following query parameters:

Parameter	Options	Default
`type`	`all`, `owner`, `member`	`owner`
`sort`	`created`, `updated`, `pushed`, `full_name`	`full_name`
`direction`	`asc`, `desc`	`asc` for `full_name`, otherwise `desc`
`per_page`	How many results per page (max 100)	`30`
`page`	Which page to fetch	`1`

Suppose we want to list the 50 most recently created repos from `hadley`, our API query would be: https://api.github.com/users/hadley/repos?sort=created&direction=desc&per_page=50

APIs with an R library wrapper

When using `httr` to make API calls, you are constructing the API URLs by yourself

Many popular APIs have a dedicated R library that wraps the API calls so that it is even easier to use and more user-friendly.

- gh: Minimalistic GitHub API client in R
- tuber: Client for the YouTube API
- twitteR: R Based Twitter Client
- Rfacebook: Access to Facebook API via R
- meetupapi: Access 'Meetup' API
- spotifyr: Pull Track Audio Features from the 'Spotify' Web API
- RedditExtractoR: Reddit Data Extraction Toolkit
- ZillowR: R Interface to Zillow Real Estate and Mortgage Data API
- ...

The gh package

I will use the `gh` package to demonstrate how to use an R library wrapper to make APIs calls. The logic can be applied to all other API libraries. However, you should read the library manual in order to learn how to properly use each of these libraries.

```
library(gh)

# Need a token. Get it from https://github.com/settings/tokens
# Otherwise, you may get the error message: "Github API Rate limit exceeded"
my_token = "123_very_long_random_string_321"
Sys.setenv(GITHUB_TOKEN = my_token)
```

GitHub API Rate limiting

- For authenticated API requests, you can make up to **5000 requests per hour**. Authenticated requests are associated with the authenticated user based on the token.
- For unauthenticated requests, the rate limit allows for up to **60 requests per hour**. Unauthenticated requests are associated with the originating IP address, and not the user making requests.

Your turn

Let's take a minute for you to get your token (you will need it for this lecture as well as assignment 2):

https://github.com/settings/tokens

1. GitHub may ask you what privileges to be granted to the token. **Uncheck all of them**. This will make it a read-only token, which is sufficient for our lecture/assignment.

2. Save the token as you will need it for the later part of this lab and our next assignment.

3. Revoke the token once the assignment is graded (in 2 weeks).

User `hadley`'s profile: https://github.com/hadley

```
hadley <- gh("/users/hadley")
class(hadley)
```

```
## [1] "gh_response" "list"
```

```
length(hadley)
```

```
## [1] 32
```

```
names(hadley)
```

```
##  [1] "login"               "id"                  "node_id"
##  [4] "avatar_url"          "gravatar_id"         "url"
##  [7] "html_url"            "followers_url"       "following_url"
## [10] "gists_url"           "starred_url"         "subscriptions_url"
## [13] "organizations_url"   "repos_url"           "events_url"
## [16] "received_events_url" "type"                "site_admin"
## [19] "name"                "company"             "blog"
## [22] "location"            "email"               "hireable"
## [25] "bio"                 "twitter_username"    "public_repos"
## [28] "public_gists"        "followers"           "following"
## [31] "created_at"          "updated_at"
```

List of `hadley`'s repositories: https://github.com/hadley?tab=repositories

```
hadley_repos <- gh("/users/hadley/repos", .limit = Inf) # get all repos
length(hadley_repos)
```

```
## [1] 331
```

```
hadley_repos[[1]]
```

```
## $id
## [1] 40423928
##
## $node_id
## [1] "MDEwOlJlcG9zaXRvcnk0MDQyMzkyOA=="
##
## $name
## [1] "15-state-of-the-union"
##
## $full_name
## [1] "hadley/15-state-of-the-union"
##
## $private
## [1] FALSE
##
## $owner
## $owner$login
## [1] "hadley"
```

List of users who are following `hadley`: https://github.com/hadley?tab=followers

```
#Over 20k! Get the first 100. Set .limit = Inf if you want to get all
hadley_followers <- gh("/users/hadley/followers", .limit = 100)
length(hadley_followers)
```

```
## [1] 100
```

```
hadley_followers[[1]]
```

```
## $login
## [1] "topfunky"
##
## $id
## [1] 26
##
## $node_id
## [1] "MDQ6VXNlcjI2"
##
## $avatar_url
## [1] "https://avatars.githubusercontent.com/u/26?v=4"
##
## $gravatar_id
## [1] ""
##
## $url
## [1] "https://api.github.com/users/topfunky"
```

Members of `Google` on GitHub: https://github.com/orgs/google/people

```
#Get the first 100. Set .limit = Inf if you want to get all
google_members <- gh("/orgs/google/members", .limit = 100)
length(google_members)
```

```
## [1] 100
```

```
google_members[[1]]
```

```
## $login
## [1] "44past4"
##
## $id
## [1] 6388530
##
## $node_id
## [1] "MDQ6VXNlcjYzODg1MzA="
##
## $avatar_url
## [1] "https://avatars.githubusercontent.com/u/6388530?v=4"
##
## $gravatar_id
## [1] ""
##
## $url
## [1] "https://api.github.com/users/44past4"
```

How to turn list into data.frame?

You would notice that `gh_response` is a list of objects? How do you convert these objects in a list to a data.frame?

Once the data is in a data.frame, it is much easier to analyze.

Functional programming using `purrr`:

```
df_google_members = map_df(
  google_members,
  magrittr::extract,
  c("login", "id", "type") # what values to extract from the list
)

df_google_members
```

```
## # A tibble: 100 × 3
##    login                id type
##    <chr>             <int> <chr>
##  1 44past4         6388530 User
##  2 aabmass         1510004 User
##  3 aaronj1335       787066 User
##  4 aarontp         2667195 User
##  5 acmcarther      1660129 User
##  6 acozzette       1115459 User
##  7 adamvduke         94930 User
##  8 adarob          1088232 User
##  9 AdrianAtGoogle 26070065 User
## 10 advaitjain      2789958 User
## # ℹ 90 more rows
```

Finally! A data frame! Hallelujah!

Use of variables in `gh` API calls

It may be convenient to use variables/placeholders when making your API calls.

To do this in `gh`:

```
# gh("/users/hadley") =
hadley <- gh("/users/{username}", username = "hadley")

# gh("/users/hadley/repos") =
hadley_repos <- gh("/users/{username}/repos", username = "hadley")

# gh("/orgs/google/members?page=2") =
google_members <- gh("/orgs/{org}/members", org = "google", page = 2)

# gh("/repos/google/guava") =
google_guava <- gh("/repos/{owner}/{repo}", owner = "google", repo = "guava")
```

Your turn

Find facebook members on GitHub

For each of the facebook members on GitHub, retrieve the number of his/her followers.

Create a data frame like the following:

```
## # A tibble: 105 × 2
##    login            followers
##    <chr>                <int>
##  1 aaronabramov          1417
##  2 adamgross42            595
##  3 afterdusk               68
##  4 ahmadi18                32
##  5 ahmed-shehata          324
##  6 aigoncharov            318
##  7 alexholdenmiller      1054
##  8 amyreese               444
##  9 anankervis             606
## 10 anshulverma            269
## # ℹ 95 more rows
```