MY472 - Data for Data Scientists
Week 8: APIs

Daniel de Kadt
14 November 2023

Introduction

- We've now learned how to **take** data from structured, unstructured, and even dynamic webpages.

- However, many sources prefer to simply **give** us (some of) their data!

- This week we will learn about the modal way of doing this – web APIs

Plan for today

- JSON
- APIs
- API Examples
- Coding

JSON

JSON

- API responses are very often delivered in JSON format (**J**avaScript **O**bject **N**otation)
- JSON is a lightweight, flexible, easy-to-parse format to store and transmit data
- JSON data can be read/parsed into R with the `fromJSON` function from the `jsonlite` package
- Yet, many packages have their own functions to read data in JSON format into R, e.g. the `content(r, ...)` function from the `httr` package which we will use a little later.

JSON

- JSON objects are key-value pairs: "someKey": [someValue]
- Many key-value pairs can be in a single JSON object, separated with ","
- Keys have to be strings with double quotes
- Values can be one of the following types:
    - String ("hello")
    - Number (42, 3.141)
    - Array ([][])
    - Boolean (true, false)
    - null
- Often follow a nested structure

Reference: https://www.w3schools.com/js/js_json_syntax.asp

JSON, Example 1

```
{
    "name": "Bob",
    "courseWork": [
        "Assignment",
        "Final"
    ],
    "grades": [
        65,
        73
    ],
    "supervisor": {
        "name": "Alice",
        "department": "Mathematics"
    },
    "currentlyEnrolled": false
}
```

JSON, Example 2


```
{
    "date": [
        "2020-10-01",
        "2020-10-17",
        "2020-10-24"
    ],
    "section": [
        "Economics",
        "Politics",
        "Sports"
    ],
    "headline": [
        "Covid recession",
        "New polls",
        "Liverpool wins"
    ],
    "lead_paragraph": [
        "The recession triggered by the pandemic ...",
        null,
        "In their game on Saturday, Liverpool FC ..."
    ]
}
```

JSON, Example 3


```
{
    "MT": [
        {
            "code": "MY472",
            "title": "Data for Data Scientists",
            "description": "A course about collecting, processing, and storing data.",
            "units": 0.5,
        },
        {
            "code": "MY470",
            "title": "Computer Programming",
            "description": "An introduction to programming.",
            "units": 0.5,
        }
    ],
    "LT": [
        {
            "code": "MY459",
            "title": "Special Topics in Quantitative Analysis: Quantitative Text Analysis",
            "description": "A course about text analysis.",
            "units": 0.5,
        }
    ]
}
```

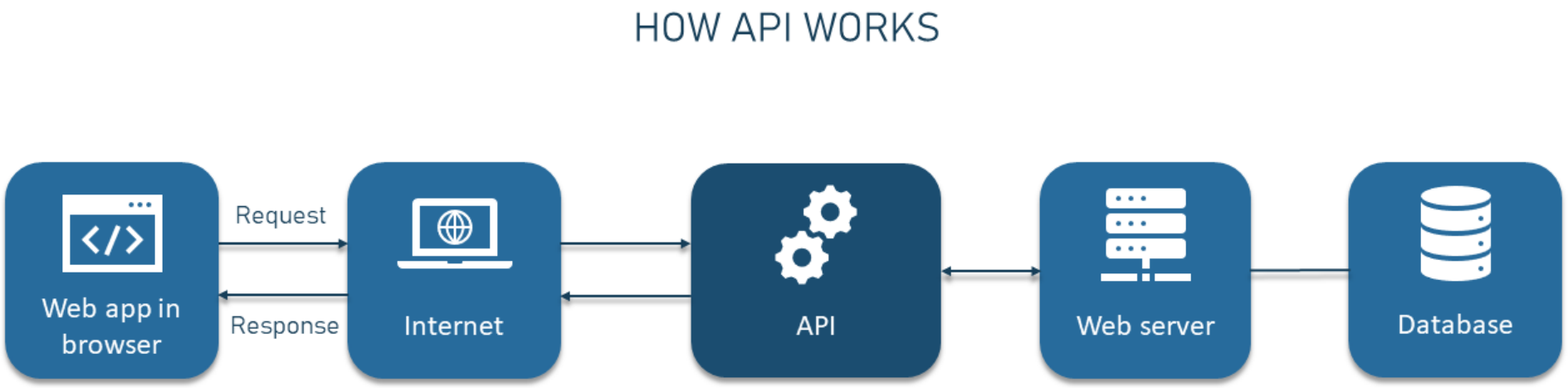
APIs

APIs

- API: **A**pplication **P**rogramming **I**nterface
- In web APIs, a set of structured HTTP/S requests can return data in a lightweight format e.g. JSON or XML
- The API user sends a request to the API (e.g. with a software such as R) and the API returns data from the API provider's database, in accordance with the provider's permissions
- APIs are widely used to communicate between applications
- APIs are now also widely available for data-curious scientists

See also e.g. Munzert et al., 2014, Chapter 9

How APIs Work



Source: https://www.altexsoft.com/blog/what-is-api-definition-types-specifications-documentation/

More on APIs

Types of APIs:

- **RESTful APIs**: Queries for static information at current moment (e.g. user profiles, posts, etc.)
- **Streaming APIs**: Changes in users' data in real time (e.g. new tweets, weather alerts…)

APIs generally have extensive **documentation**:

- Written for developers
- What to look for: **Endpoints** and **parameters**

Endpoints: A web location that receives requests and sends responses

Parameters: Allow you to send (very) specific requests

More on APIs

Many APIs require a **key** or **tokens** (often generated via a developer account)

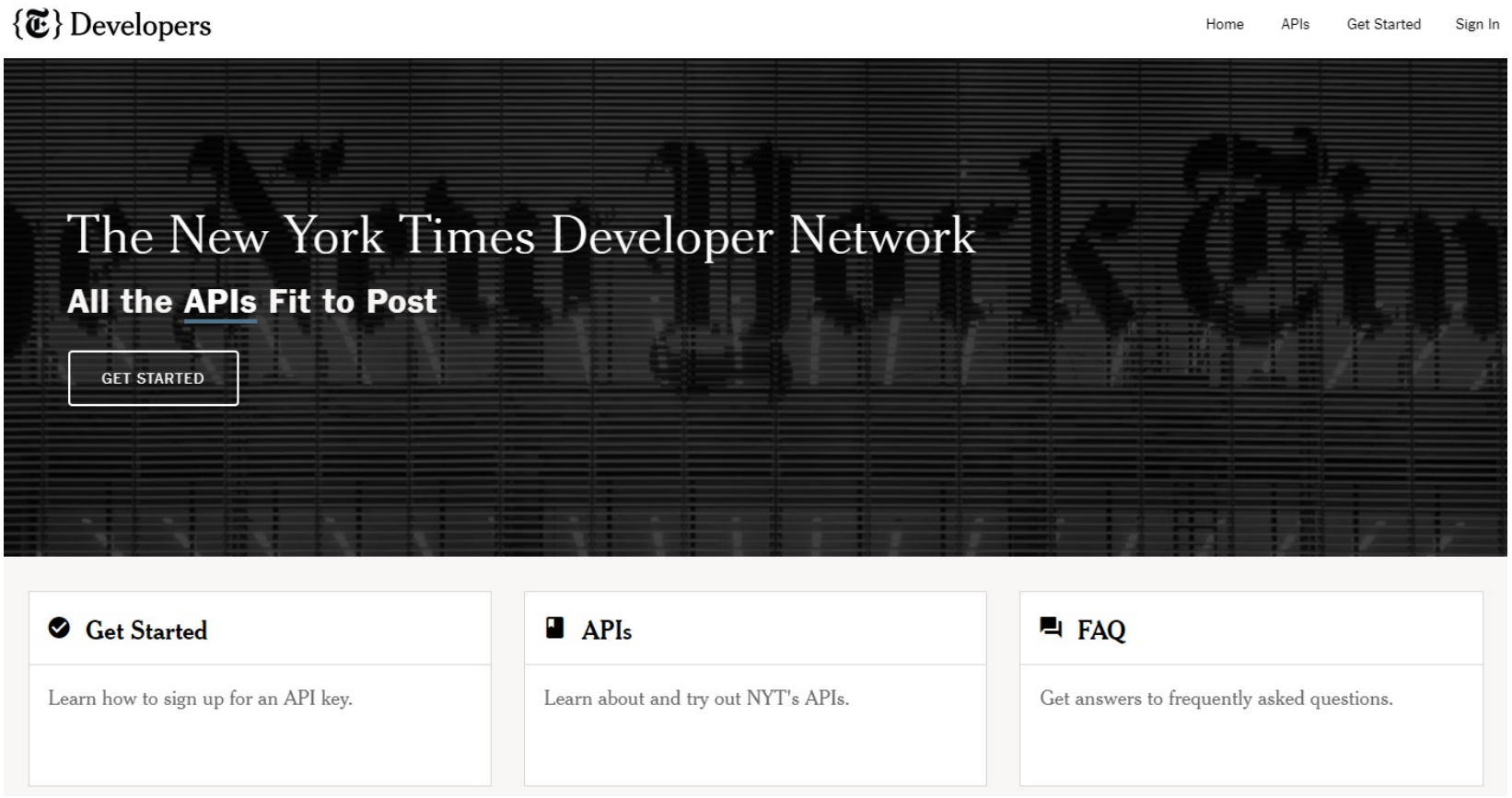
Most APIs are **rate-limited**:

- Restrictions on number of API calls by user/key/IP address and period of time
- Commercial APIs may impose a monthly fee (via your account)

Just because APIs allow access, doesn't give you *carte blanche*:

- Commercial and non-commercial APIs typically have terms of use
- Read what you can/cannot do with the API and the data!

An Access Example: The NYTimes API



Website: https://developer.nytimes.com/

An Access Example: The NYTimes API

The New York Times (NYT) offers a range of APIs

In your seminar you will use:

- The Article Search API to search for keywords in articles
- The Archive API to download the full data for a given month

We aren't given access to full articles in the public version of the Archive API, but we can obtain headlines, abstracts, snippets, and/or lead paragraphs since 1851

An Access Example: The NYTimes API

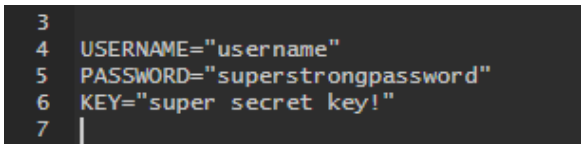
To gain access to the New York Times APIs:

- Follow these instructions: https://developer.nytimes.com/get-started
- When specifying access rights, select the boxes for Article Search and Archive API
- You will generate a (private!) key for access

Remember, your key is your **responsibility**. **Never** hard-code an API key (or a password) into code that another person/developer/user will see (e.g. a public github repo).

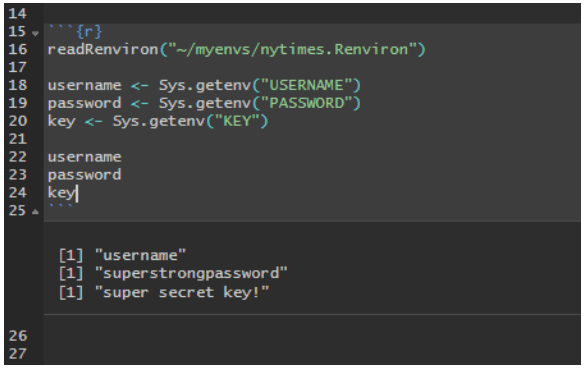
An Access Example: The NYTimes API

To avoid hard-coding passwords or keys in `R`, we use `.Renviron` files:



```
USERNAME="username"
PASSWORD="superstrongpassword"
KEY="super secret key!"
```

In this case, I have created a locally stored (not on github!) file called `nytimes.Renviron`. It stores my `USERNAME`, `PASSWORD`, and `KEY`



````
```{r}
readRenviron("~/myenvs/nytimes.Renviron")

username <- Sys.getenv("USERNAME")
password <- Sys.getenv("PASSWORD")
key <- Sys.getenv("KEY")

username
password
key
```

[1] "username"
[1] "superstrongpassword"
[1] "super secret key!"
````

I can now read this file from `R` and assign my **masked** key as an object which I can use. No-one accessing this script on github gets access to your locally stored `.Renviron` file (they have their own), so no credentials are ever unmasked.

A Workflow Example: Google Maps API

Constructing an API call

- Baseline URL endpoint: `https://maps.googleapis.com/maps/api/geocode/json`
- Parameters: `?address=london`
- Authentication token: `key=XXXXX`

From `R`, use `httr` package to make `GET` request:

```
library(httr)
r <- GET("https://maps.googleapis.com/maps/api/geocode/json",
         query=list(address="london", key="XXXXX")
     )
```

If request was successful, returned code will be `200`. `4xx` indicates client errors and `5xx` indicates server errors. If you need to attach data, use `POST` request.

A Workflow Example: Google Maps API


```
{
   "results" : [
      {
         "address_components" : [
            {
               "long_name" : "London",
               "short_name" : "London",
               "types" : [ "locality", "political" ]
            },
            {
               "long_name" : "London",
               "short_name" : "London",
               "types" : [ "postal_town" ]
            }
         ],

...
```

A Workflow Example: Google Maps API


```
...
        "formatted_address" : "London, UK",
        "geometry" : {
           "bounds" : {
              "northeast" : {
                 "lat" : 51.6723432,
                 "lng" : 0.148271
              },
              "southwest" : {
                 "lat" : 51.38494009999999,
                 "lng" : -0.3514683
              }
           },
           "location" : {
              "lat" : 51.5073509,
              "lng" : -0.1277583
           },
           "location_type" : "APPROXIMATE",
...
```

A Workflow Example: Google Maps API


```
...
            "viewport" : {
               "northeast" : {
                  "lat" : 51.6723432,
                  "lng" : 0.148271
               },
               "southwest" : {
                  "lat" : 51.38494009999999,
                  "lng" : -0.3514683
               }
            }
         },
         "place_id" : "ChIJdd4hrwug2EcRmSrV3Vo6llI",
         "types" : [ "locality", "political" ]
      }
   ],
   "status" : "OK"
}
```

APIs vs. Scraping for Data Science

Advantages

- Cleaner data collection: Avoid malformed HTML, fewer legal issues, clear data structures, more trust in data collection…
- Standardised data access procedures: Transparency, replicability
- Robustness: Many users/developers is usually a good thing, support may exist

Disadvantages

- Not always available
- Dependency on API providers (e.g. Twitter/X)
- Rate limits
- Price

APIs for Social Media

One area that APIs have been used extensively for in social science is the study of social media.

A lot of work was done with Twitter's REST and Streaming APIs, but these are now essentially defunct.

Other social media APIs do exist, but the taps are being shut:

- Facebook/Meta have limited APIs
- Reddit (8-12 weeks for researcher approval)
- Bluesky (very new)
- etc.

Coding

Markdown files

- 01-json-in-r.Rmd
- 02-aic-api.Rmd