Getting started with API in R

Hedia Tnani

# Getting started with API in R

## Why should I learn about APIs?

For the past 2 years, many groups have been working on creating APIs that the entire research community can leverage. There’s a a publication [A Review on Genomics APIs](https://www.sciencedirect.com/science/article/pii/S2001037015000471#f0015) which shows some of the APIs used to retrieve data such as Google Genomics API and 23andMe API.

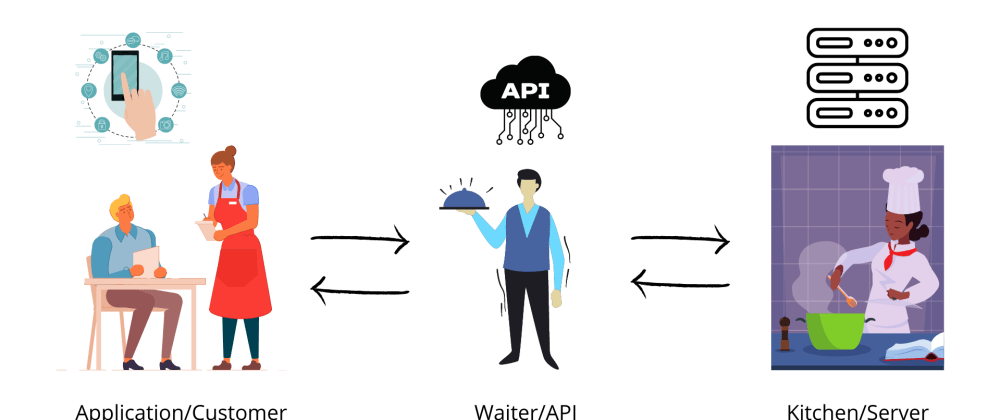
## What is an API?

**API** stands for **Application Programming Interface.**

An API acts as interface between two applications (Client and sever).

## How?

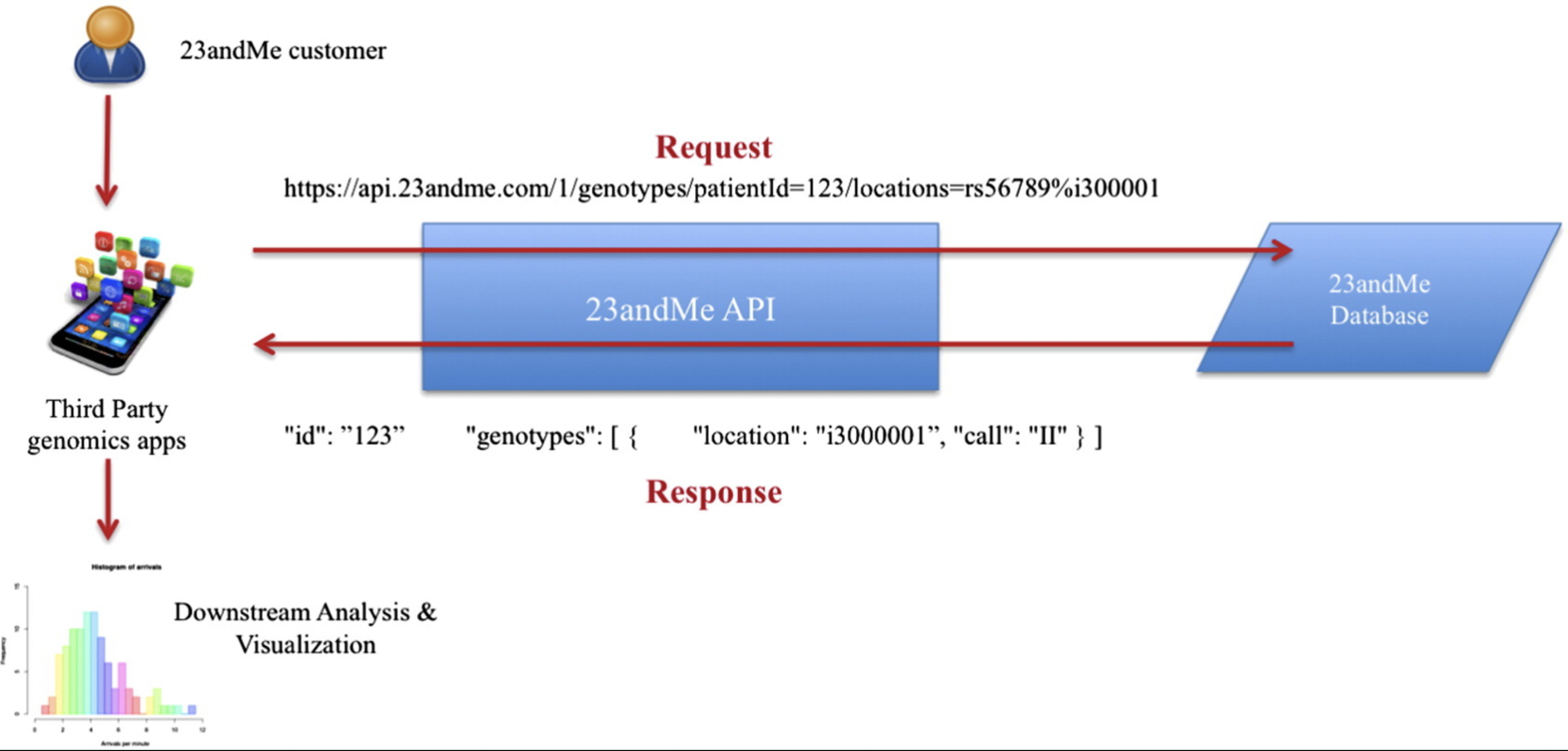
Let’s take a non technical example to make it to understand.



[Source](https://dev.to/hackthisfall/what-is-api-explained-in-easy-way-5aih)

In terms of API, we can say that we as **Customer(application)** will send a **request**, the **waiter(API)** will take our request and send it to the **chef(server).** The **chef(server)** takes action according to our request and send it back to the waiter(API). This is how actually things work in the *backend*.

An example from the 23andMe API.



[Source](https://www.sciencedirect.com/science/article/pii/S2001037015000471#s0020)

## Making API requests in R

To work with APIs in R, we need to bring in two libraries httr and jsonlite.

### 1. Preparing the URL

### **What is an URL?**

The **Uniform Resource Locator or URL** is a string (of characters) that uniquely identifies a web resource. For example, “https://rest.ensembl.org” is the URL that identifies Ensembl’s homepage. The URL is the input of any API request. It consists of two parts: - The fixed part: base URL — the base URL is the main address of the API web server - The variable part: endpoint — an endpoint is a server route for retrieving specific data from an API.

### **What is an API endpoint?**

API endpoints are the specific digital location where requests for information are sent by one program to retrieve the digital resource that exists there. **Endpoints specify where APIs can access resources and help guarantee the proper functioning of the incorporated software**. An API’s performance depends on its capacity to successfully communicate with API endpoints.

Example:

[Ensembl REST API Endpoints](https://rest.ensembl.org)

Let’s load the library(httr). The modify\_url() allows us to combine base URLs and endpoints.

library(httr)  
# Preparing the URL   
baseurl <- "https://rest.ensembl.org"  
ext <- "/sequence/id/ENSG00000157764"  
url <- modify\_url(baseurl, path = ext)  
url

[1] "https://rest.ensembl.org/sequence/id/ENSG00000157764"

### 2. Querying the database

APIs use **HTTP verbs** for data requests. One of the most common is a **GET** request. The GET request asks the API to ***retrieve*** a resource, such as a database record or the contents of a file, and send it to the client. To make a GET request we´ll be using the **GET() function** from the httr package.

response = GET(url)  
response

Response [https://rest.ensembl.org/sequence/id/ENSG00000157764]  
 Date: 2023-02-23 14:53  
 Status: 200  
 Content-Type: application/json  
 Size: 206 kB

The API response contains, among other information, the query URL and date, the response status, and the content type and size. It might also include the content itself, but this part isn’t visible here.

The Content-Type of the API response is an application/json.

JSON stands for **JavaScript Object Notation.**

### 3. Tracking Errors

When making queries, it’s entirely possible that the output isn’t the right type or that something will go wrong . Therefore, it’s important to track these errors. Web servers return status codes every time they receive an API request. A status code reports what happened with a request.

Let´s check the status code of our request:

http\_status(response)

$category  
[1] "Success"  
  
$reason  
[1] "OK"  
  
$message  
[1] "Success: (200) OK"

The Status-Code element in a server response, is a 3-digit integer where the first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit:

| S.N. | Code and Description |
| --- | --- |
| 1 | **1xx: Informational**  It means the request has been received and the process is continuing. |
| 2 | **2xx: Success**  It means the action was successfully received, understood, and accepted. |
| 3 | **3xx: Redirection**  It means further action must be taken in order to complete the request. |
| 4 | **4xx: Client Error**  It means the request contains incorrect syntax or cannot be fulfilled. |
| 5 | **5xx: Server Error**  It means the server failed to fulfill an apparently valid request. |

The httr package provides us with several functions to track these errors.

* The **http\_error(response)** checks if something went wrong and returns a logical value.

http\_error(response)

[1] FALSE

* The **status\_code(response)** returns the status code.

status\_code(response)

[1] 200

* The **http\_type(response)** returns the format of the content, so we can see if it is what we expected.

http\_type(response)

[1] "application/json"

When something goes wrong and http\_error(response) returns TRUE, we stop the function execution using the stop() function.

if(http\_error(response)){  
 status\_code(404)  
 stop("Something went wrong.")}

Another useful function is stop\_for\_status(response) which converts http errors to R errors or warnings - these should always be used whenever you’re creating requests inside a function, so that the user knows why a request has failed. If request was successful, the response (invisibly). Otherwise, raised a classed http error or warning, as generated by http\_condition() . Other helpful functions are warn\_for\_status() and message\_for\_status().

### 4. Getting the API response

One very useful function to use is content which **extract content from a request. For that** we use **content()** from the httr package to extract this content.

library(httr)  
# automatically parses JSON  
APIResult = content(response)  
#APIResult$seq

So here we sent an **API request** to the webserver, and it replied with the sequence(data).

# add code to extract content here  
json\_text <- content(response, as="text")  
#json\_text

Let’s add an encoding.

json\_text <-content(response, as = "text", encoding = "UTF-8")  
#json\_text

We can provide the content\_type in the GET().

### 5. Converting the response into a dataframe

Let’s get all microarrays available for a given species. In R, we can parse this JSON string and convert it into a DataFrame using the fromJSON() from the jsonlite package. If the output of an endpoint contains a list, R will happily get all items in a list without the need for any loops.

library(httr)  
library(jsonlite)  
library(xml2)  
library(tidyverse)

── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──  
✔ ggplot2 3.4.0 ✔ purrr 0.3.5   
✔ tibble 3.1.8 ✔ dplyr 1.0.10  
✔ tidyr 1.2.1 ✔ stringr 1.4.1   
✔ readr 2.1.3 ✔ forcats 0.5.2   
── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
✖ dplyr::filter() masks stats::filter()  
✖ purrr::flatten() masks jsonlite::flatten()  
✖ dplyr::lag() masks stats::lag()

baseURL <- "https://rest.ensembl.org"  
ext <- "/regulatory/species/homo\_sapiens/microarray?"  
   
r <- GET(paste(baseURL, ext, sep = ""), content\_type("application/json"))  
   
stop\_for\_status(r)  
class(content(r)) # list

[1] "list"

#fromJSON(content(r, "text", encoding = "UTF-8"))  
#content(r, "text", encoding = "UTF-8")

The fromJSON needs as an argument a JSON string, URL or file. To make it a JSON string we convert it to text.

content(r, "text", encoding = "UTF-8")

[1] "[{\"vendor\":\"PHALANX\",\"description\":null,\"array\":\"OneArray\",\"format\":\"EXPRESSION\",\"type\":\"OLIGO\"},{\"type\":\"OLIGO\",\"array\":\"CODELINK\",\"format\":\"EXPRESSION\",\"description\":null,\"vendor\":\"CODELINK\"},{\"description\":null,\"vendor\":\"ILLUMINA\",\"array\":\"HumanWG\_6\_V3\",\"format\":\"EXPRESSION\",\"type\":\"OLIGO\"},{\"type\":\"OLIGO\",\"array\":\"GPL6848\",\"format\":\"EXPRESSION\",\"description\":null,\"vendor\":\"AGILENT\"},{\"vendor\":\"ILLUMINA\",\"description\":null,\"type\":\"OLIGO\",\"array\":\"HumanMethylation450\",\"format\":\"METHYLATION\"},{\"vendor\":\"ILLUMINA\",\"description\":null,\"format\":\"EXPRESSION\",\"array\":\"HumanRef-8\_V3\",\"type\":\"OLIGO\"},{\"array\":\"WholeGenome\",\"format\":\"EXPRESSION\",\"type\":\"OLIGO\",\"vendor\":\"AGILENT\",\"description\":null},{\"array\":\"CGH\_44b\",\"format\":\"CGH\",\"type\":\"OLIGO\",\"description\":null,\"vendor\":\"AGILENT\"},{\"array\":\"SurePrint\_G3\_GE\_8x60k\",\"format\":\"EXPRESSION\",\"type\":\"OLIGO\",\"vendor\":\"AGILENT\",\"description\":null},{\"format\":\"EXPRESSION\",\"array\":\"HC-G110\",\"type\":\"OLIGO\",\"description\":null,\"vendor\":\"AFFY\"},{\"description\":null,\"vendor\":\"AGILENT\",\"type\":\"OLIGO\",\"array\":\"SurePrint\_G3\_GE\_8x60k\_v2\",\"format\":\"EXPRESSION\"},{\"vendor\":\"AFFY\",\"description\":null,\"type\":\"OLIGO\",\"format\":\"EXPRESSION\",\"array\":\"HG-Focus\"},{\"description\":null,\"vendor\":\"AGILENT\",\"type\":\"OLIGO\",\"format\":\"EXPRESSION\",\"array\":\"WholeGenome\_4x44k\_v1\"},{\"type\":\"OLIGO\",\"array\":\"WholeGenome\_4x44k\_v2\",\"format\":\"EXPRESSION\",\"vendor\":\"AGILENT\",\"description\":null},{\"vendor\":\"AGILENT\",\"description\":null,\"type\":\"OLIGO\",\"format\":\"EXPRESSION\",\"array\":\"GPL26966\"},{\"format\":\"EXPRESSION\",\"array\":\"HG-U133A\_2\",\"type\":\"OLIGO\",\"description\":null,\"vendor\":\"AFFY\"},{\"vendor\":\"AGILENT\",\"description\":null,\"type\":\"OLIGO\",\"format\":\"EXPRESSION\",\"array\":\"GPL19072\"},{\"format\":\"METHYLATION\",\"array\":\"HumanMethylation27\",\"type\":\"OLIGO\",\"description\":null,\"vendor\":\"ILLUMINA\"},{\"vendor\":\"AFFY\",\"description\":null,\"array\":\"HG-U133B\",\"format\":\"EXPRESSION\",\"type\":\"OLIGO\"},{\"description\":\"Human Transcriptome Array 2.0\",\"vendor\":\"AFFY\",\"format\":\"EXPRESSION\",\"array\":\"HTA-2\_0\",\"type\":\"OLIGO\"},{\"format\":\"EXPRESSION\",\"array\":\"HG-U133\_Plus\_2\",\"type\":\"OLIGO\",\"description\":null,\"vendor\":\"AFFY\"},{\"vendor\":\"AFFY\",\"description\":null,\"format\":\"EXPRESSION\",\"array\":\"HG\_U95Av2\",\"type\":\"OLIGO\"},{\"vendor\":\"AFFY\",\"description\":null,\"array\":\"HG-U95B\",\"format\":\"EXPRESSION\",\"type\":\"OLIGO\"},{\"type\":\"OLIGO\",\"array\":\"HG-U95C\",\"format\":\"EXPRESSION\",\"description\":null,\"vendor\":\"AFFY\"},{\"type\":\"OLIGO\",\"array\":\"HG-U95D\",\"format\":\"EXPRESSION\",\"vendor\":\"AFFY\",\"description\":null},{\"array\":\"HG-U95E\",\"format\":\"EXPRESSION\",\"type\":\"OLIGO\",\"description\":null,\"vendor\":\"AFFY\"},{\"type\":\"OLIGO\",\"array\":\"HG\_U95A\",\"format\":\"EXPRESSION\",\"vendor\":\"AFFY\",\"description\":null},{\"type\":\"OLIGO\",\"format\":\"EXPRESSION\",\"array\":\"HuGeneFL\",\"description\":null,\"vendor\":\"AFFY\"},{\"vendor\":\"AFFY\",\"description\":null,\"type\":\"OLIGO\",\"format\":\"EXPRESSION\",\"array\":\"U133\_X3P\"},{\"type\":\"OLIGO\",\"array\":\"PrimeView\",\"format\":\"EXPRESSION\",\"description\":null,\"vendor\":\"AFFY\"},{\"description\":null,\"vendor\":\"AFFY\",\"type\":\"OLIGO\",\"array\":\"HT\_HG-U133\_Plus\_PM\",\"format\":\"EXPRESSION\"},{\"vendor\":\"AFFY\",\"description\":null,\"type\":\"OLIGO\",\"array\":\"HuEx-1\_0-st-v2\",\"format\":\"EXPRESSION\"},{\"vendor\":\"AFFY\",\"description\":null,\"array\":\"HuGene-2\_0-st-v1\",\"format\":\"EXPRESSION\",\"type\":\"OLIGO\"},{\"vendor\":\"AFFY\",\"description\":null,\"type\":\"OLIGO\",\"format\":\"EXPRESSION\",\"array\":\"HuGene-1\_0-st-v1\"},{\"type\":\"OLIGO\",\"array\":\"HuGene-2\_1-st-v1\",\"format\":\"EXPRESSION\",\"vendor\":\"AFFY\",\"description\":null}]"

To convert it to a dataframe we use the fromJSON function.

fromJSON(content(r, "text", encoding = "UTF-8"))

vendor description array format  
1 PHALANX <NA> OneArray EXPRESSION  
2 CODELINK <NA> CODELINK EXPRESSION  
3 ILLUMINA <NA> HumanWG\_6\_V3 EXPRESSION  
4 AGILENT <NA> GPL6848 EXPRESSION  
5 ILLUMINA <NA> HumanMethylation450 METHYLATION  
6 ILLUMINA <NA> HumanRef-8\_V3 EXPRESSION  
7 AGILENT <NA> WholeGenome EXPRESSION  
8 AGILENT <NA> CGH\_44b CGH  
9 AGILENT <NA> SurePrint\_G3\_GE\_8x60k EXPRESSION  
10 AFFY <NA> HC-G110 EXPRESSION  
11 AGILENT <NA> SurePrint\_G3\_GE\_8x60k\_v2 EXPRESSION  
12 AFFY <NA> HG-Focus EXPRESSION  
13 AGILENT <NA> WholeGenome\_4x44k\_v1 EXPRESSION  
14 AGILENT <NA> WholeGenome\_4x44k\_v2 EXPRESSION  
15 AGILENT <NA> GPL26966 EXPRESSION  
16 AFFY <NA> HG-U133A\_2 EXPRESSION  
17 AGILENT <NA> GPL19072 EXPRESSION  
18 ILLUMINA <NA> HumanMethylation27 METHYLATION  
19 AFFY <NA> HG-U133B EXPRESSION  
20 AFFY Human Transcriptome Array 2.0 HTA-2\_0 EXPRESSION  
21 AFFY <NA> HG-U133\_Plus\_2 EXPRESSION  
22 AFFY <NA> HG\_U95Av2 EXPRESSION  
23 AFFY <NA> HG-U95B EXPRESSION  
24 AFFY <NA> HG-U95C EXPRESSION  
25 AFFY <NA> HG-U95D EXPRESSION  
26 AFFY <NA> HG-U95E EXPRESSION  
27 AFFY <NA> HG\_U95A EXPRESSION  
28 AFFY <NA> HuGeneFL EXPRESSION  
29 AFFY <NA> U133\_X3P EXPRESSION  
30 AFFY <NA> PrimeView EXPRESSION  
31 AFFY <NA> HT\_HG-U133\_Plus\_PM EXPRESSION  
32 AFFY <NA> HuEx-1\_0-st-v2 EXPRESSION  
33 AFFY <NA> HuGene-2\_0-st-v1 EXPRESSION  
34 AFFY <NA> HuGene-1\_0-st-v1 EXPRESSION  
35 AFFY <NA> HuGene-2\_1-st-v1 EXPRESSION  
 type  
1 OLIGO  
2 OLIGO  
3 OLIGO  
4 OLIGO  
5 OLIGO  
6 OLIGO  
7 OLIGO  
8 OLIGO  
9 OLIGO  
10 OLIGO  
11 OLIGO  
12 OLIGO  
13 OLIGO  
14 OLIGO  
15 OLIGO  
16 OLIGO  
17 OLIGO  
18 OLIGO  
19 OLIGO  
20 OLIGO  
21 OLIGO  
22 OLIGO  
23 OLIGO  
24 OLIGO  
25 OLIGO  
26 OLIGO  
27 OLIGO  
28 OLIGO  
29 OLIGO  
30 OLIGO  
31 OLIGO  
32 OLIGO  
33 OLIGO  
34 OLIGO  
35 OLIGO

Let´s convert it to a tibble

as\_tibble(fromJSON(content(r, "text", encoding = "UTF-8")))

# A tibble: 35 × 5  
 vendor description array format type   
 <chr> <chr> <chr> <chr> <chr>  
 1 PHALANX <NA> OneArray EXPRESSION OLIGO  
 2 CODELINK <NA> CODELINK EXPRESSION OLIGO  
 3 ILLUMINA <NA> HumanWG\_6\_V3 EXPRESSION OLIGO  
 4 AGILENT <NA> GPL6848 EXPRESSION OLIGO  
 5 ILLUMINA <NA> HumanMethylation450 METHYLATION OLIGO  
 6 ILLUMINA <NA> HumanRef-8\_V3 EXPRESSION OLIGO  
 7 AGILENT <NA> WholeGenome EXPRESSION OLIGO  
 8 AGILENT <NA> CGH\_44b CGH OLIGO  
 9 AGILENT <NA> SurePrint\_G3\_GE\_8x60k EXPRESSION OLIGO  
10 AFFY <NA> HC-G110 EXPRESSION OLIGO  
# … with 25 more rows

### 6. Making a function

Let´s make a function.

library(httr)  
library(jsonlite)  
  
get\_microarrays <- function(baseURL, ext, content\_type){  
  
 r <- GET(paste(baseURL, ext, sep = ""), accept(content\_type))  
  
 stop\_for\_status(r)  
  
 if (content\_type == 'application/json'){  
 return (fromJSON(content(r, "text", encoding = "UTF-8")))  
 } else {  
 return (content(r, "text", encoding = "UTF-8"))  
 }  
}  
  
baseURL <- "https://rest.ensembl.org"  
ext <- "/regulatory/species/homo\_sapiens/microarray?"  
con <- "application/json"  
  
  
get\_microarrays(baseURL, ext, con)

format array description vendor  
1 EXPRESSION OneArray <NA> PHALANX  
2 EXPRESSION CODELINK <NA> CODELINK  
3 EXPRESSION HumanWG\_6\_V3 <NA> ILLUMINA  
4 EXPRESSION GPL6848 <NA> AGILENT  
5 METHYLATION HumanMethylation450 <NA> ILLUMINA  
6 EXPRESSION HumanRef-8\_V3 <NA> ILLUMINA  
7 EXPRESSION WholeGenome <NA> AGILENT  
8 CGH CGH\_44b <NA> AGILENT  
9 EXPRESSION SurePrint\_G3\_GE\_8x60k <NA> AGILENT  
10 EXPRESSION HC-G110 <NA> AFFY  
11 EXPRESSION SurePrint\_G3\_GE\_8x60k\_v2 <NA> AGILENT  
12 EXPRESSION HG-Focus <NA> AFFY  
13 EXPRESSION WholeGenome\_4x44k\_v1 <NA> AGILENT  
14 EXPRESSION WholeGenome\_4x44k\_v2 <NA> AGILENT  
15 EXPRESSION GPL26966 <NA> AGILENT  
16 EXPRESSION HG-U133A\_2 <NA> AFFY  
17 EXPRESSION GPL19072 <NA> AGILENT  
18 METHYLATION HumanMethylation27 <NA> ILLUMINA  
19 EXPRESSION HG-U133B <NA> AFFY  
20 EXPRESSION HTA-2\_0 Human Transcriptome Array 2.0 AFFY  
21 EXPRESSION HG-U133\_Plus\_2 <NA> AFFY  
22 EXPRESSION HG\_U95Av2 <NA> AFFY  
23 EXPRESSION HG-U95B <NA> AFFY  
24 EXPRESSION HG-U95C <NA> AFFY  
25 EXPRESSION HG-U95D <NA> AFFY  
26 EXPRESSION HG-U95E <NA> AFFY  
27 EXPRESSION HG\_U95A <NA> AFFY  
28 EXPRESSION HuGeneFL <NA> AFFY  
29 EXPRESSION U133\_X3P <NA> AFFY  
30 EXPRESSION PrimeView <NA> AFFY  
31 EXPRESSION HT\_HG-U133\_Plus\_PM <NA> AFFY  
32 EXPRESSION HuEx-1\_0-st-v2 <NA> AFFY  
33 EXPRESSION HuGene-2\_0-st-v1 <NA> AFFY  
34 EXPRESSION HuGene-1\_0-st-v1 <NA> AFFY  
35 EXPRESSION HuGene-2\_1-st-v1 <NA> AFFY  
 type  
1 OLIGO  
2 OLIGO  
3 OLIGO  
4 OLIGO  
5 OLIGO  
6 OLIGO  
7 OLIGO  
8 OLIGO  
9 OLIGO  
10 OLIGO  
11 OLIGO  
12 OLIGO  
13 OLIGO  
14 OLIGO  
15 OLIGO  
16 OLIGO  
17 OLIGO  
18 OLIGO  
19 OLIGO  
20 OLIGO  
21 OLIGO  
22 OLIGO  
23 OLIGO  
24 OLIGO  
25 OLIGO  
26 OLIGO  
27 OLIGO  
28 OLIGO  
29 OLIGO  
30 OLIGO  
31 OLIGO  
32 OLIGO  
33 OLIGO  
34 OLIGO  
35 OLIGO

Let’s get the mouse homologue of the human BRCA2. If you specify another content type (not json), the helper function will get you this as text.

gene <- "BRCA2"  
  
# define the URL parameters  
server <- "http://rest.ensembl.org/"  
con <- "application/json"  
request <- paste("homology/symbol/human/", gene, "?target\_species=mouse", sep = "")  
r <- GET(paste(server, request, sep = ""), accept(con))  
stop\_for\_status(r)  
result = fromJSON(content(r, "text", encoding = "UTF-8"))  
result$data$id

[1] "ENSG00000139618"

Let’s retrieve the Kegg database using the Kegg API.

#### Kegg API using R

library(httr)  
library(jsonlite)  
library(xml2)  
   
server <- "https://rest.kegg.jp"  
ext <- "/info/hsa"  
   
r <- GET(paste(server, ext, sep = ""), content\_type("application/json"))  
   
content(r)

[1] "T01001 Homo sapiens (human) KEGG Genes Database\nhsa Release 105.0+/02-23, Feb 23\n Kanehisa Laboratories\n 24,729 entries\n\nlinked db pathway\n brite\n module\n ko\n genome\n enzyme\n network\n disease\n drug\n ncbi-geneid\n ncbi-proteinid\n uniprot\n"

Let’s have a look at the list of human pathways in Kegg.

library(httr)  
library(jsonlite)  
library(xml2)  
library(tidyverse)  
  
server <- "https://rest.kegg.jp"  
ext <- "/list/pathway/hsa"  
   
r <- GET(paste(server, ext, sep = ""), content\_type("application/json"))  
   
txt <- content(r, "text")  
df <- tibble(kegg\_results = read\_lines(txt))

df %>% separate(., col=kegg\_results, into=c('Pathway\_number', 'Pathway\_name'), sep='\t')

# A tibble: 352 × 2  
 Pathway\_number Pathway\_name   
 <chr> <chr>   
 1 path:hsa00010 Glycolysis / Gluconeogenesis - Homo sapiens (human)   
 2 path:hsa00020 Citrate cycle (TCA cycle) - Homo sapiens (human)   
 3 path:hsa00030 Pentose phosphate pathway - Homo sapiens (human)   
 4 path:hsa00040 Pentose and glucuronate interconversions - Homo sapiens (huma…  
 5 path:hsa00051 Fructose and mannose metabolism - Homo sapiens (human)   
 6 path:hsa00052 Galactose metabolism - Homo sapiens (human)   
 7 path:hsa00053 Ascorbate and aldarate metabolism - Homo sapiens (human)   
 8 path:hsa00061 Fatty acid biosynthesis - Homo sapiens (human)   
 9 path:hsa00062 Fatty acid elongation - Homo sapiens (human)   
10 path:hsa00071 Fatty acid degradation - Homo sapiens (human)   
# … with 342 more rows

Let’s retrieve the amino acid sequence of a human gene in a fasta format.

server <- "https://rest.kegg.jp"  
ext <- "/get/hsa:10458/aaseq"  
   
r <- GET(paste(server, ext, sep = ""), content\_type("text/x-fasta"))  
  
print(content(r))

[1] ">hsa:10458 K05627 BAI1-associated protein 2 | (RefSeq) BAIAP2, BAP2, FLAF3, IRSP53, WAML; BAR/IMD domain containing adaptor protein 2 (A)\nMSLSRSEEMHRLTENVYKTIMEQFNPSLRNFIAMGKNYEKALAGVTYAAKGYFDALVKMG\nELASESQGSKELGDVLFQMAEVHRQIQNQLEEMLKSFHNELLTQLEQKVELDSRYLSAAL\nKKYQTEQRSKGDALDKCQAELKKLRKKSQGSKNPQKYSDKELQYIDAISNKQGELENYVS\nDGYKTALTEERRRFCFLVEKQCAVAKNSAAYHSKGKELLAQKLPLWQQACADPSKIPERA\nVQLMQQVASNGATLPSALSASKSNLVISDPIPGAKPLPVPPELAPFVGRMSAQESTPIMN\nGVTGPDGEDYSPWADRKAAQPKSLSPPQSQSKLSDSYSNTLPVRKSVTPKNSYATTENKT\nLPRSSSMAAGLERNGRMRVKAIFSHAAGDNSTLLSFKEGDLITLLVPEARDGWHYGESEK\nTKMRGWFPFSYTRVLDSDGSDRLHMSLQQGKSSSTGNLLDKDDLAIPPPDYGAASRAFPA\nQTASGFKQRPYSVAVPAFSQGLDDYGARSMSRNPFAHVQLKPTVTNDRCDLSAQGPEGRE\nHGDGSARTLAGR\n"

Let’s try the xml content\_type.

server <- "https://rest.kegg.jp"  
ext <- "/get/hsa00600/kgml"  
   
r <- GET(paste(server, ext, sep = ""), content\_type("text/xml"))  
  
content(r,encoding = "UTF-8")

{xml\_document}  
<pathway name="path:hsa00600" org="hsa" number="00600" title="Sphingolipid metabolism" image="https://www.kegg.jp/kegg/pathway/hsa/hsa00600.png" link="https://www.kegg.jp/kegg-bin/show\_pathway?hsa00600">  
 [1] <entry id="128" name="hsa:5660 hsa:768239" type="gene" link="https://www ...  
 [2] <entry id="35" name="hsa:55331" type="gene" reaction="rn:R06518" link="h ...  
 [3] <entry id="36" name="ko:K05848" type="ortholog" reaction="rn:R06528" lin ...  
 [4] <entry id="38" name="hsa:130367 hsa:81537" type="gene" reaction="rn:R065 ...  
 [5] <entry id="39" name="hsa:130367 hsa:81537" type="gene" reaction="rn:R065 ...  
 [6] <entry id="40" name="hsa:130367 hsa:81537" type="gene" reaction="rn:R065 ...  
 [7] <entry id="41" name="hsa:427 hsa:56624" type="gene" reaction="rn:R06518" ...  
 [8] <entry id="42" name="hsa:427 hsa:56624" type="gene" reaction="rn:R06528" ...  
 [9] <entry id="43" name="hsa:10715 hsa:204219 hsa:253782 hsa:29956 hsa:79603 ...  
[10] <entry id="44" name="hsa:123099 hsa:8560" type="gene" reaction="rn:R0651 ...  
[11] <entry id="45" name="ko:K04709" type="ortholog" reaction="rn:R06527" lin ...  
[12] <entry id="46" name="hsa:123099 hsa:8560" type="gene" reaction="rn:R0652 ...  
[13] <entry id="47" name="hsa:123099 hsa:8560" type="gene" reaction="rn:R0652 ...  
[14] <entry id="48" name="hsa:339221 hsa:55512 hsa:55627 hsa:6609 hsa:6610" t ...  
[15] <entry id="49" name="hsa:10715 hsa:204219 hsa:253782 hsa:29956 hsa:79603 ...  
[16] <entry id="51" name="ko:K26283" type="ortholog" reaction="rn:R01503" lin ...  
[17] <entry id="52" name="ko:K01123" type="ortholog" reaction="rn:R02542" lin ...  
[18] <entry id="53" name="hsa:64781" type="gene" reaction="rn:R01495" link="h ...  
[19] <entry id="54" name="hsa:8611 hsa:8612 hsa:8613" type="gene" reaction="r ...  
[20] <entry id="55" name="hsa:8611 hsa:8612 hsa:8613" type="gene" reaction="r ...  
...

### It´s YOUR TURN!!!!

## References

https://www.w3schools.com/js/js\_json\_intro.asp

https://www.tutorialspoint.com/http/http\_status\_codes.htm

https://medium.com/geekculture/a-beginners-guide-to-apis-9aa7b1b2e172