Javascript basics V

APIs; a “full” app

INFO 6150
Fernando Augusto López Plascencia

In this lesson:

- Understanding APIs
  - Exploring APIs
  - Requests and responses
  - Using curl
- Consuming APIs through Javascript
  - Promises
  - fetch
  - Async/await
- Making a full app

You can find the full code from these slides here: https://github.com/sgenius/web-samples/tree/main/info-6150/06-apis-classes/js5-currency-conversion

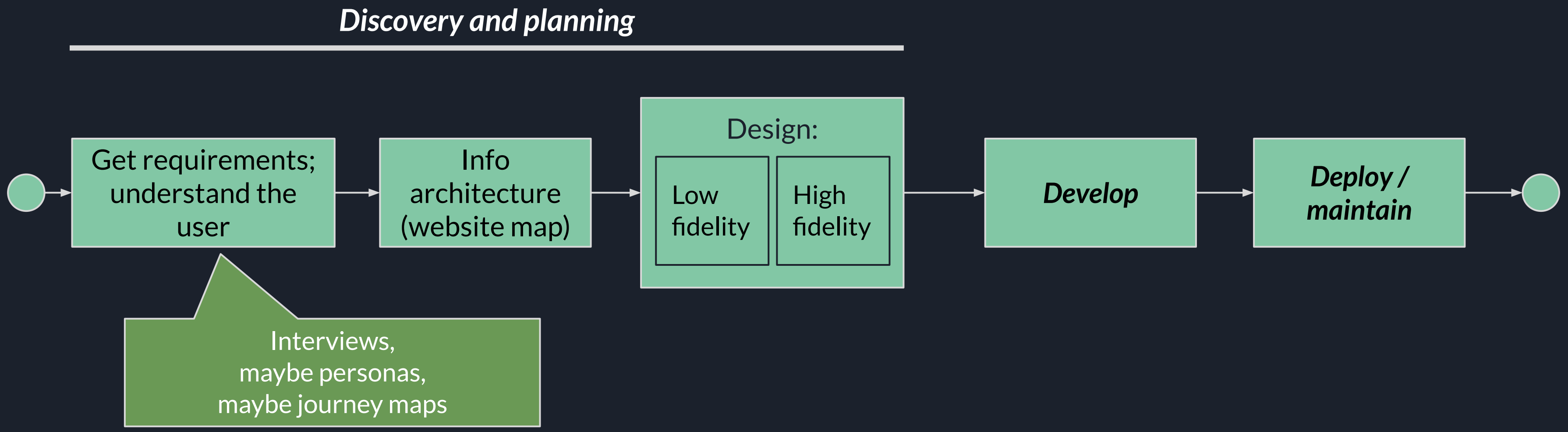
Understanding APIs

Guess what...

We’re doing a new app today!

Steps to create an app

As a reminder, this is how you create a website or app:



Discovery and planning

Get requirements; understand the user

Interviews,
maybe personas,
maybe journey maps

Info architecture (website map)

Design:

Low fidelity

High fidelity

Develop

Deploy / maintain

At any point you can go back to a previous step, but the earlier you catch bugs and do changes, the better.

Get requirements

Created by HideMaru
from Noun Project

We’re doing a new app today!

The instructor will act as Mr. Plane. Mr. Plane and his friends (the “Travel Friends Club”) want you to make something simple that they need, but that anyone could use.

Gather requirements from Mr. Plane. You have 5 minutes.

Icon by HideMaru for Noun Project

Today’s requirements

Travel Friends Club is asking you to build a currency conversion app that anyone can use.

- The user should be able to enter a number and select an origin currency and a destination currency (eg. “100 US dollars to Euros”).
- The system will respond with the converted amount, using the most up-to-date market rate for the conversion.

The app should be responsive and not use much bandwidth.

It’s not required to add the name of the club in the website.

Created by HideMaru
from Noun Project

Design and start developing

This is another one-page app so we'll skip the website map.

Take 2 minutes to draw a low-res mockup. (Make it simple!)

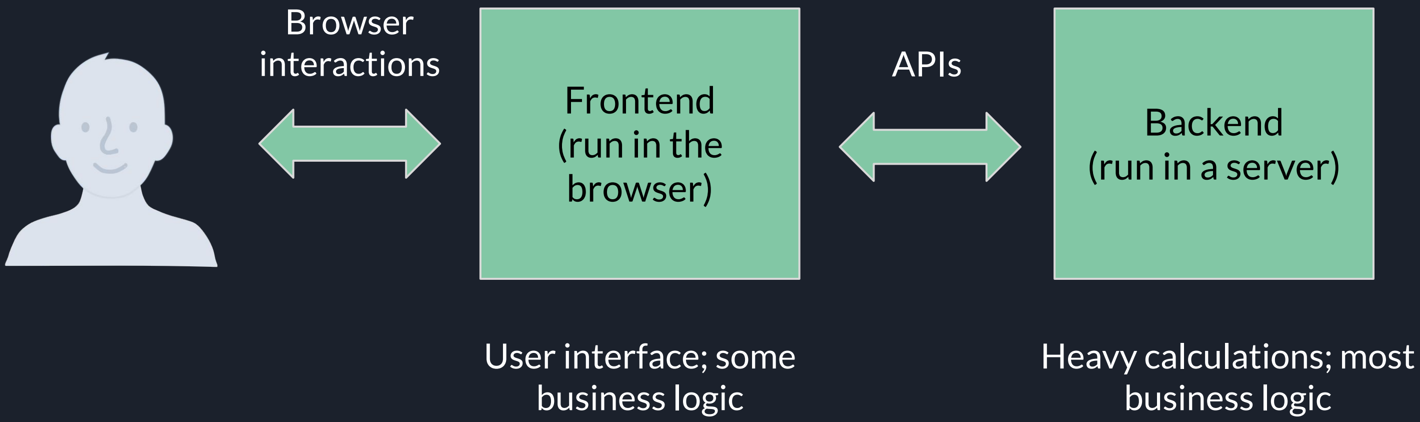
Then, take another 10 minutes to work on the HTML and CSS. Make it as similar as possible to the mockup.

Icon by HideMaru for Noun Project

Creating a "full" app

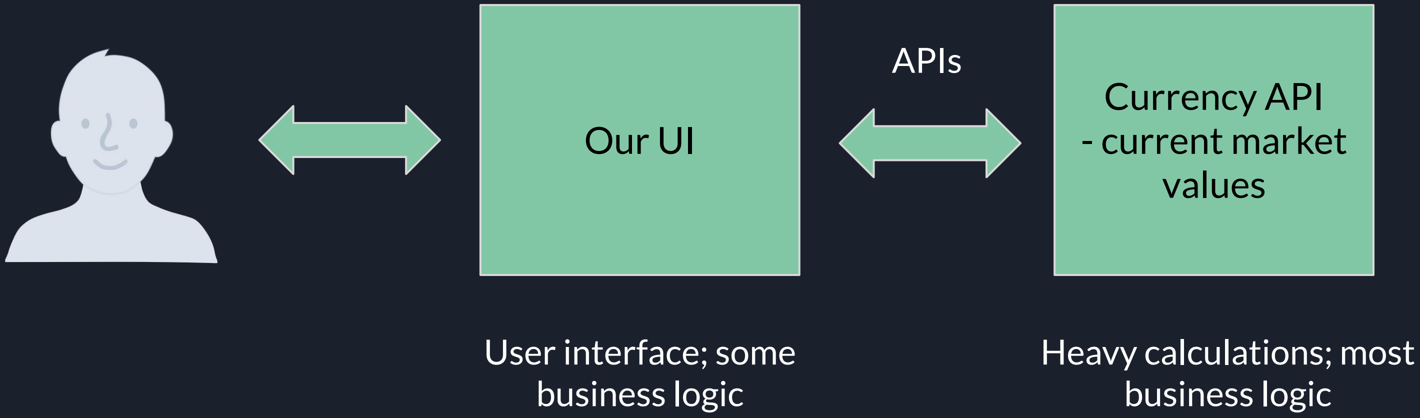
A "full" app usually has a frontend and a backend part. We will focus on the frontend part for this course.

We will use a backend with its functionality already created.


Browser interactions
Frontend (run in the browser)
APIs
Backend (run in a server)
User interface; some business logic
Heavy calculations; most business logic

Consuming data from the internet

We need to use a backend service that provides currency data.



What is an API?

API means Application Programming Interface. Broadly, it refers to any system - but the term is more commonly used for Backend Web APIs built using HTTP.

Refresher: Common Server-side APIs

- REST - uses HTML over HTTP; vanilla web. The most robust and popular. Standard
- SOAP - uses XML. Useful for very big, secure applications. Also standard
- GraphQL - a query language on top of HTTP; fast, but not standard-based
- gRPC - call functions remotely; binary data. Open source. Oh so complex

Refresher: Common Server-side APIs

- **REST - uses HTML over HTTP; vanilla web. The most robust and popular. Standard**
- SOAP - uses XML. Useful for very big, secure applications. Also standard
- GraphQL - a query language on top of HTTP; fast, but not standard-based
- gRPC - call functions remotely; binary data. Open source. Oh so complex

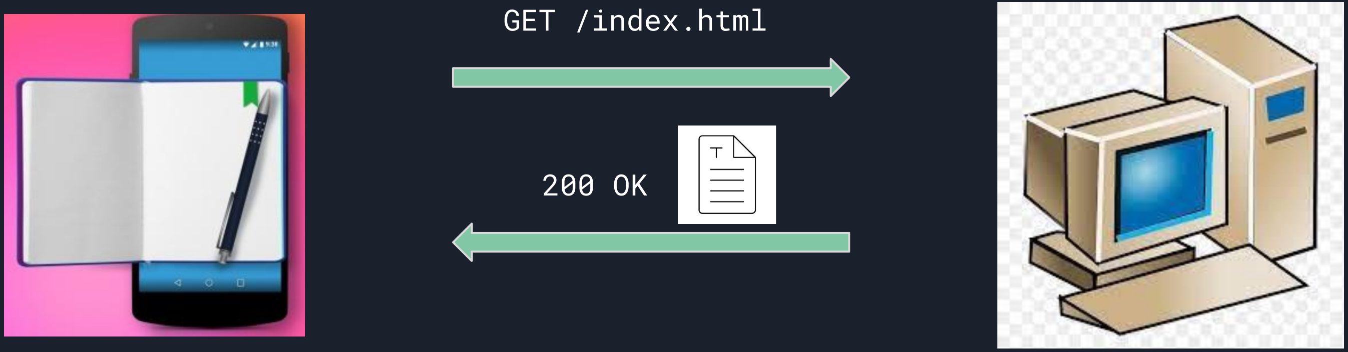
We will explore REST APIs in this course.

REST APIs are an extension of the HTTP we've already used to get web pages or send form data.

Refresher: Requests and responses

In HTTP, a client requests something from the server.

The server works on the client’s request and when ready, responds - usually with a web page in HTML or other files.



GET /index.html

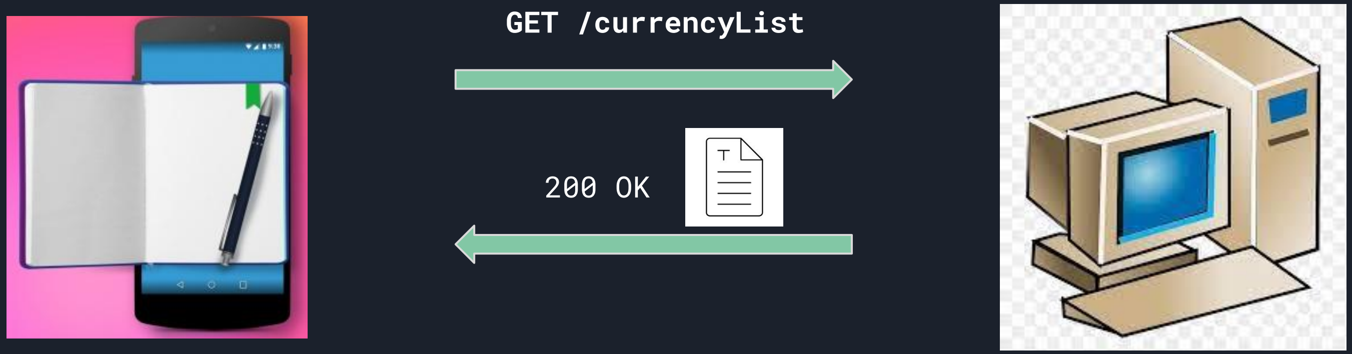
200 OK

www.wikipedia.org

REST API concepts: Endpoints

In REST APIs, a server has endpoints.

An endpoint is an HTTP address where, instead of a web page, the web server will reply with information. An endpoint may also do an operation on information, such as saving data to the server or deleting it.



GET /currencyList

200 OK

Design and start developing

Advanced Challenge! (optional)

If you already know how to use APIs:

- Finish the application. Take care of edge cases for your inputs.
  - You can use **any** API to cover the requirements.
- ★ Add an extra section in the page to display information about the countries where the destination currency is used: name, capital, flag, area, and population. It should look nice.
  - You will need **another** API for country information.
  - You'll need to figure out how to search all countries where a currency is used.

No orange bar (it's a hassle to set up, lol) but you'll see a warning before your time is up.

Normal version continues in the next slides. We will explain step by step.

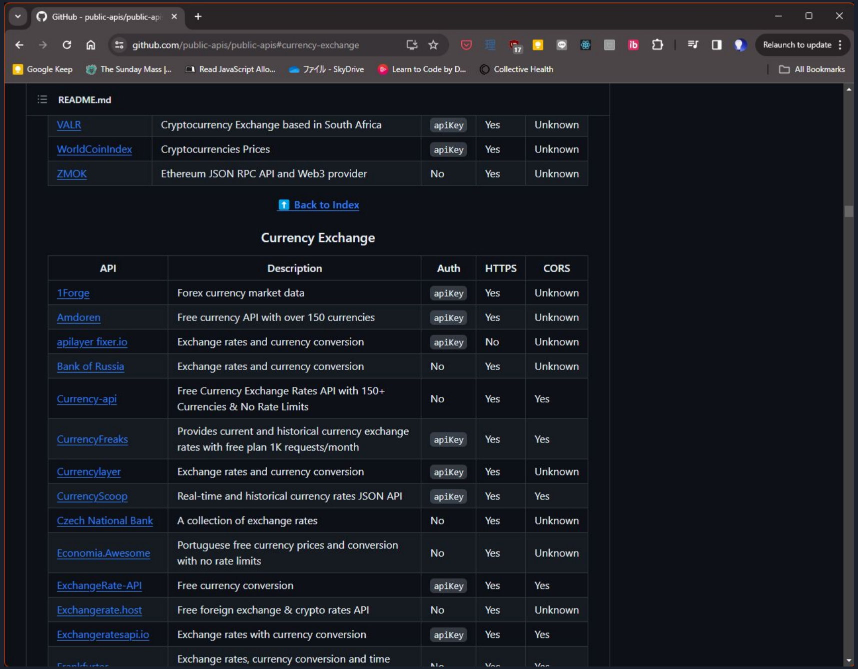
"Advanced" Icon by Read on freeicons.io

Good luck!

Exploring REST APIs

There are many open REST APIs on the internet that we can already use as backends! Some are paid, others are free.

Let’s explore one of many lists of public APIs: https://github.com/public-apis/public-apis



VALR	Cryptocurrency Exchange based in South Africa	apiKey	Yes	Unknown
WorldCoinIndex	Cryptocurrencies Prices	apiKey	Yes	Unknown
ZMOK	Ethereum JSON RPC API and Web3 provider	No	Yes	Unknown

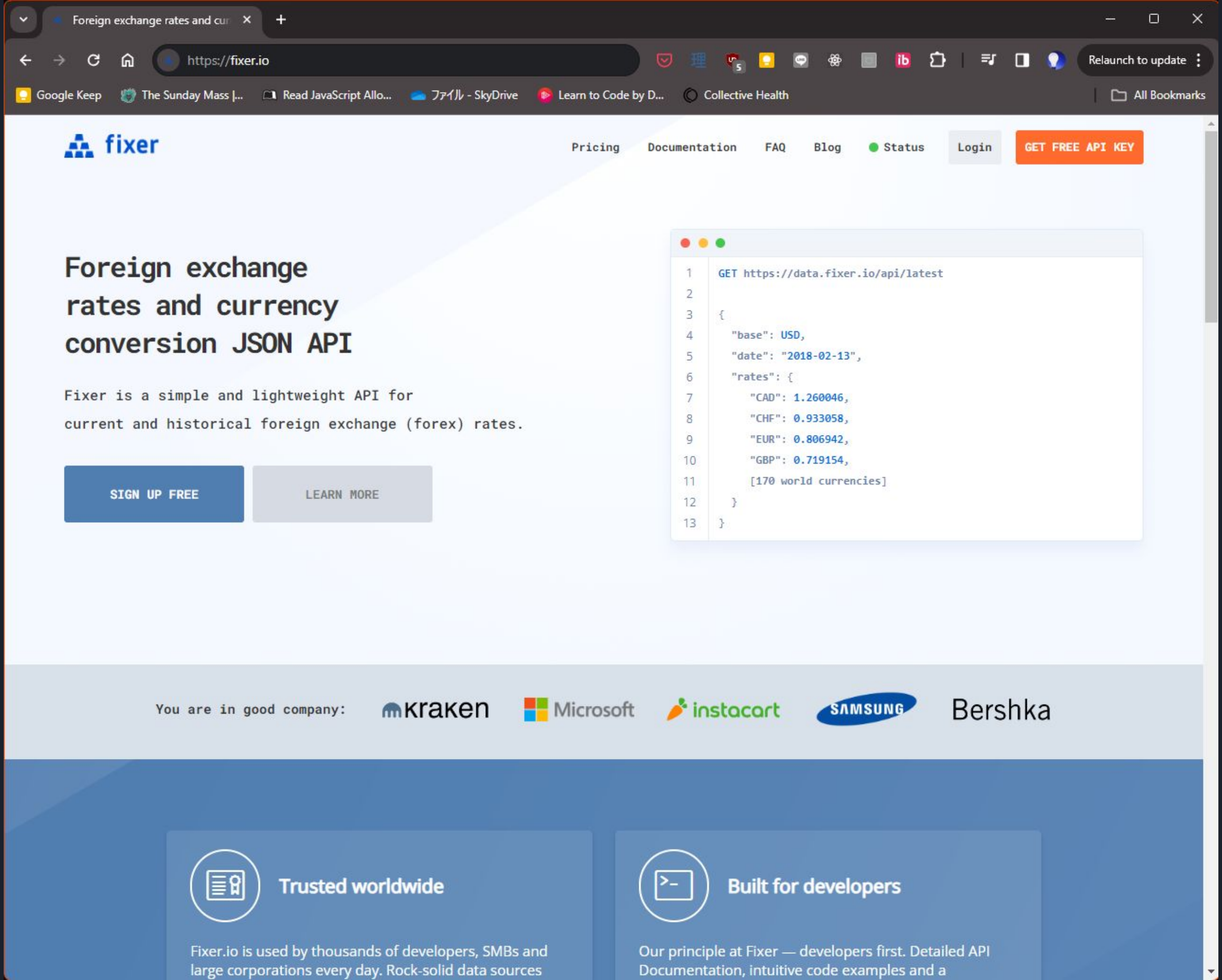
⬆ Back to Index

Currency Exchange

API	Description	Auth	HTTPS	CORS
1Forge	Forex currency market data	apiKey	Yes	Unknown
Amdoren	Free currency API with over 150 currencies	apiKey	Yes	Unknown
apilayer fixer.io	Exchange rates and currency conversion	apiKey	No	Unknown
Bank of Russia	Exchange rates and currency conversion	No	Yes	Unknown
Currency-api	Free Currency Exchange Rates API with 150+ Currencies & No Rate Limits	No	Yes	Yes
CurrencyFreaks	Provides current and historical currency exchange rates with free plan 1K requests/month	apiKey	Yes	Yes
Currencylayer	Exchange rates and currency conversion	apiKey	Yes	Unknown
CurrencyScoop	Real-time and historical currency rates JSON API	apiKey	Yes	Yes
Czech National Bank	A collection of exchange rates	No	Yes	Unknown
Economia.Awesome	Portuguese free currency prices and conversion with no rate limits	No	Yes	Unknown
ExchangeRate-API	Free currency conversion	apiKey	Yes	Yes
Exchangerate.host	Free foreign exchange & crypto rates API	No	Yes	Unknown
Exchangeratesapi.io	Exchange rates with currency conversion	apiKey	Yes	Yes
Frankfurter	Exchange rates, currency conversion and time	No	Yes	Yes

Exploring REST APIs

We will use this one: https://fixer.io/



Foreign exchange rates and currency conversion JSON API

Fixer is a simple and lightweight API for current and historical foreign exchange (forex) rates.

SIGN UP FREE

LEARN MORE

```
GET https://data.fixer.io/api/latest

{
  "base": USD,
  "date": "2018-02-13",
  "rates": {
     "CAD": 1.260046,
     "CHF": 0.933058,
     "EUR": 0.806942,
     "GBP": 0.719154,
     [170 world currencies]
  }
}
```

You are in good company:

Trusted worldwide

Fixer.io is used by thousands of developers, SMBs and large corporations every day. Rock-solid data sources

Built for developers

Our principle at Fixer — developers first. Detailed API Documentation, intuitive code examples and a

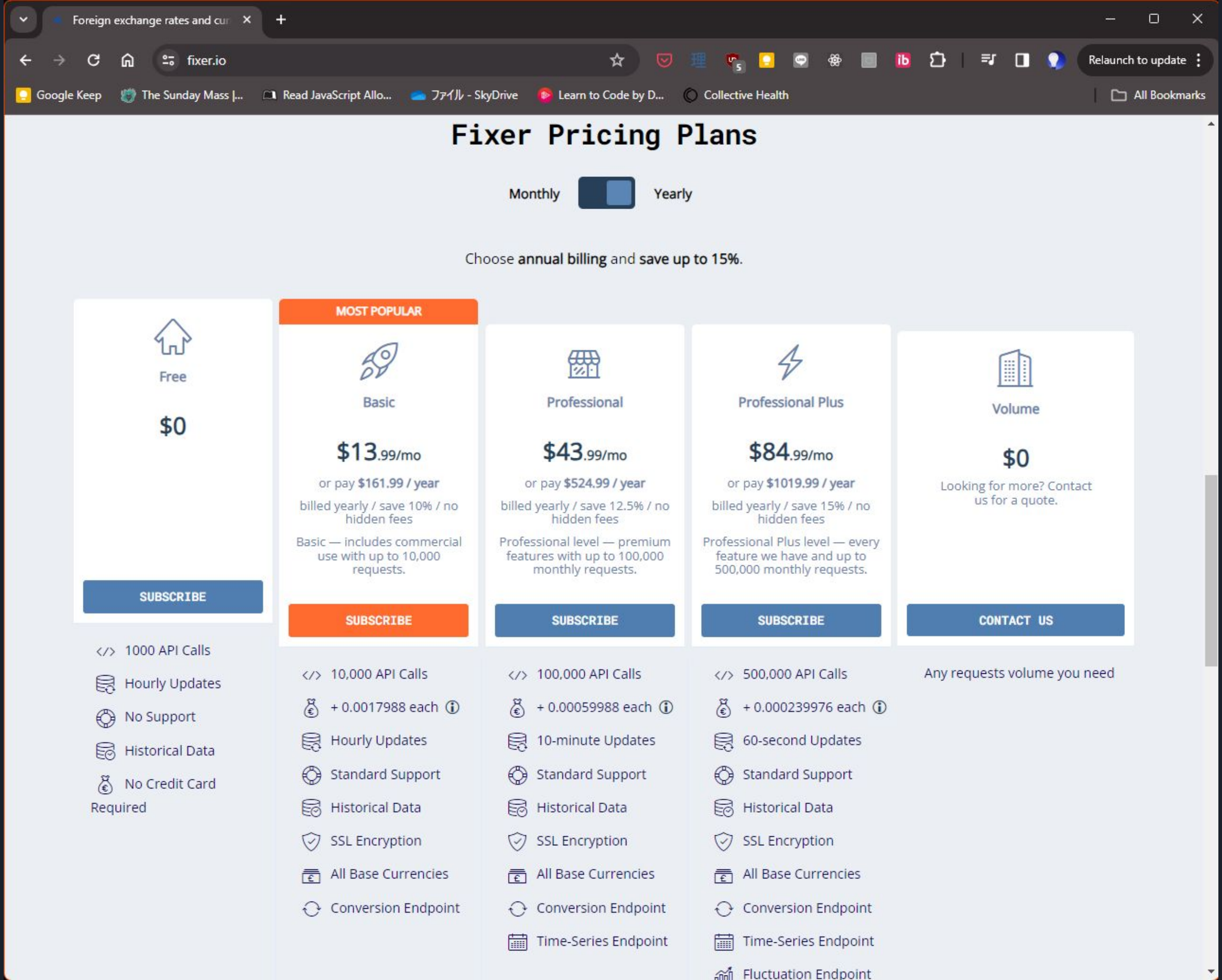
Before using an API...

Most APIs, even the free ones, come with a limited number of uses per month.

Usage may be measured in number of requests, or in the quantity of data transmitted, or both. If the service allows you to store data, that may also be limited.

Always check the terms first, so that you don’t end up with an unexpected credit card charge!

For free services, after you exceed your quota, your app may stop working until the next month.



Fixer Pricing Plans

Monthly ☑ Yearly

Choose **annual billing** and **save up to 15%**.

Free

$0

SUBSCRIBE

- 1000 API Calls
- Hourly Updates
- No Support
- Historical Data
- No Credit Card Required

MOST POPULAR

Basic

$13.99/mo

or pay **$161.99 / year**

billed yearly / save 10% / no hidden fees

Basic — includes commercial use with up to 10,000 requests.

SUBSCRIBE

- 10,000 API Calls
- + 0.0017988 each ⓘ
- Hourly Updates
- Standard Support
- Historical Data
- SSL Encryption
- All Base Currencies
- Conversion Endpoint

Professional

$43.99/mo

or pay **$524.99 / year**

billed yearly / save 12.5% / no hidden fees

Professional level — premium features with up to 100,000 monthly requests.

SUBSCRIBE

- 100,000 API Calls
- + 0.00059988 each ⓘ
- 10-minute Updates
- Standard Support
- Historical Data
- SSL Encryption
- All Base Currencies
- Conversion Endpoint
- Time-Series Endpoint

Professional Plus

$84.99/mo

or pay **$1019.99 / year**

billed yearly / save 15% / no hidden fees

Professional Plus level — every feature we have and up to 500,000 monthly requests.

SUBSCRIBE

- 500,000 API Calls
- + 0.000239976 each ⓘ
- 60-second Updates
- Standard Support
- Historical Data
- SSL Encryption
- All Base Currencies
- Conversion Endpoint
- Time-Series Endpoint
- Fluctuation Endpoint

Volume

$0

Looking for more? Contact us for a quote.

CONTACT US

Any requests volume you need

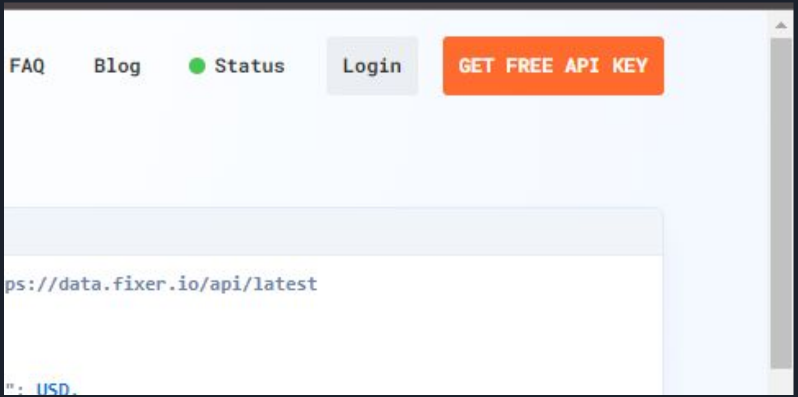
But how do they even know you’re the one making the requests?

Most APIs will ask you to create an API key.

This will be a string that identifies you as a user.

Requests to the API will need to attach the API key.

In many cases, getting the key is free.



```
ps://data.fixer.io/api/latest
": USD,
```

Exploring REST APIs

Created by HideMaru
from Noun Project

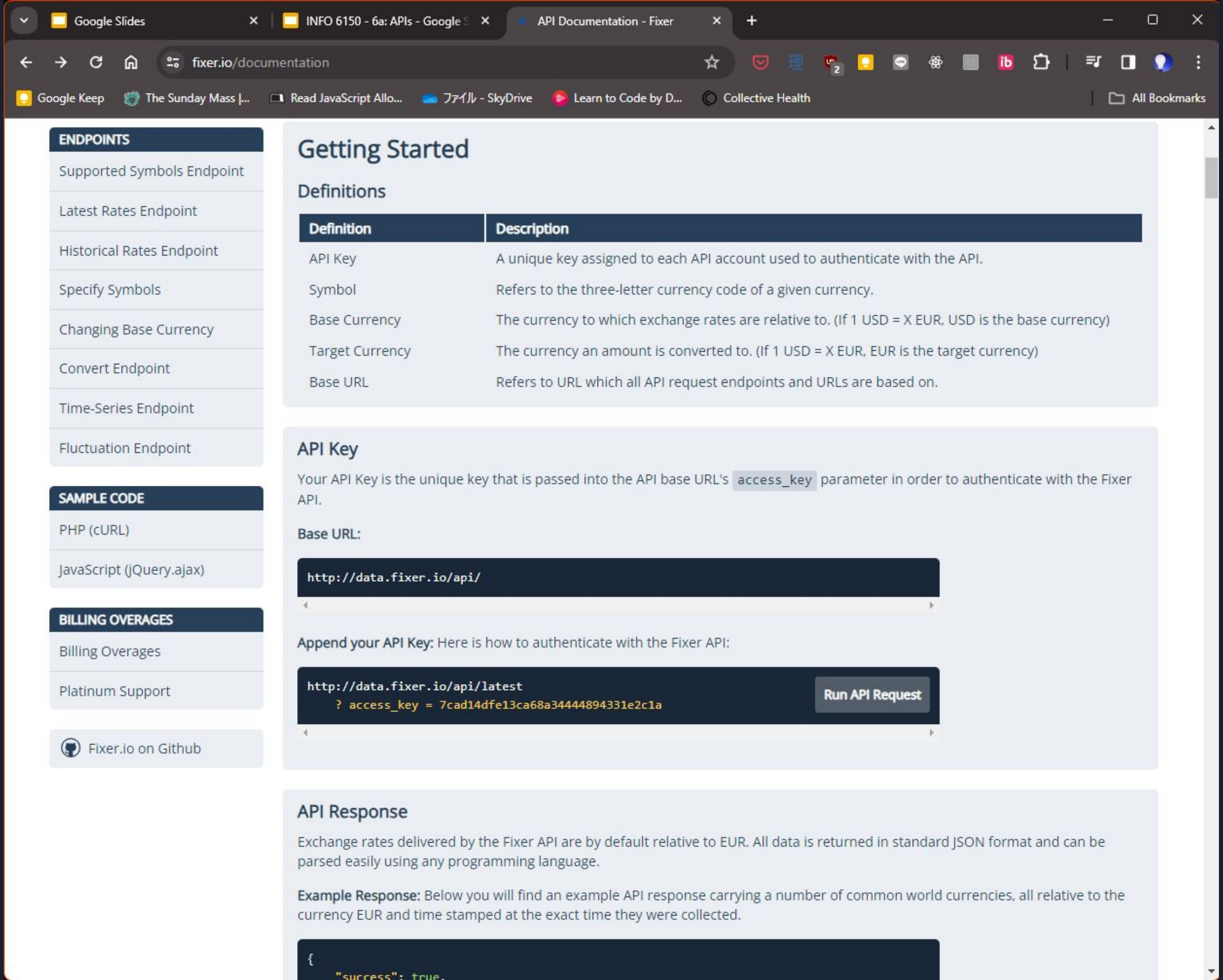
- Get a free API for fixer.io.
  - Do NOT enter credit card information.
- Take note of the API key that you get. We will use it later.

HideMaru for Noun Project

Exploring REST APIs

Created by HideMaru
from Noun Project

- In the Documentation page, scroll down to the “Endpoints” header.
- Find the endpoint that returns all available currencies.



Getting Started

Definitions

Definition	Description
API Key	A unique key assigned to each API account used to authenticate with the API.
Symbol	Refers to the three-letter currency code of a given currency.
Base Currency	The currency to which exchange rates are relative to. (If 1 USD = X EUR, USD is the base currency)
Target Currency	The currency an amount is converted to. (If 1 USD = X EUR, EUR is the target currency)
Base URL	Refers to URL which all API request endpoints and URLs are based on.

API Key

Your API Key is the unique key that is passed into the API base URL's `access_key` parameter in order to authenticate with the Fixer API.

Base URL:

```
http://data.fixer.io/api/
```

Append your API Key: Here is how to authenticate with the Fixer API:

```
http://data.fixer.io/api/latest
    ? access_key = 7cad14dfe13ca68a34444894331e2c1a
```

API Response

Exchange rates delivered by the Fixer API are by default relative to EUR. All data is returned in standard JSON format and can be parsed easily using any programming language.

Example Response: Below you will find an example API response carrying a number of common world currencies, all relative to the currency EUR and time stamped at the exact time they were collected.

```
{
    "success": true,
```

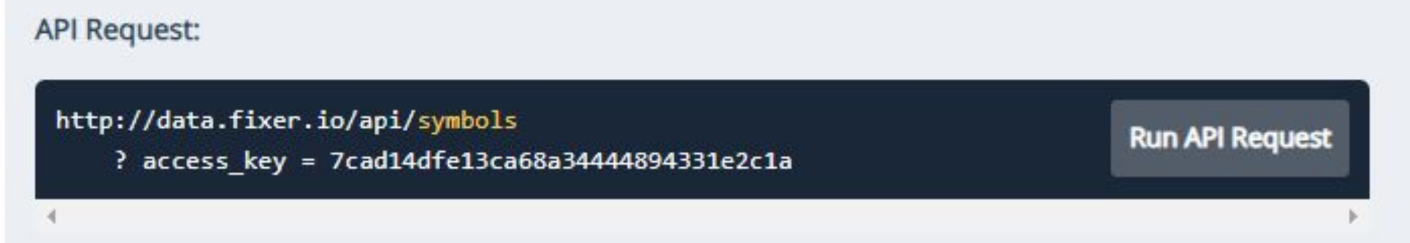
Created by HideMaru
from Noun Project

Exploring REST APIs

For each endpoint, we get a request (how we should call the endpoint) and a sample response.

Try the `/api/symbols` endpoint:

- Click on the “Run API Request” button.



API Request:

```
http://data.fixer.io/api/symbols
    ? access_key = 7cad14dfe13ca68a34444894331e2c1a
```

HideMaru for Noun Project

If you get an error:

- The documentation page has a “Potential errors” list. Check it out.

In my case, I first got this error:

```
{"success":false,"error":{"code":105,"type":"https_access_restricted","info":"Access
Restricted - Your current Subscription Plan does not support HTTPS Encryption."}}
```

To fix this, change the `https://` in the URL bar to `http://` .

Quick aside: https vs. http

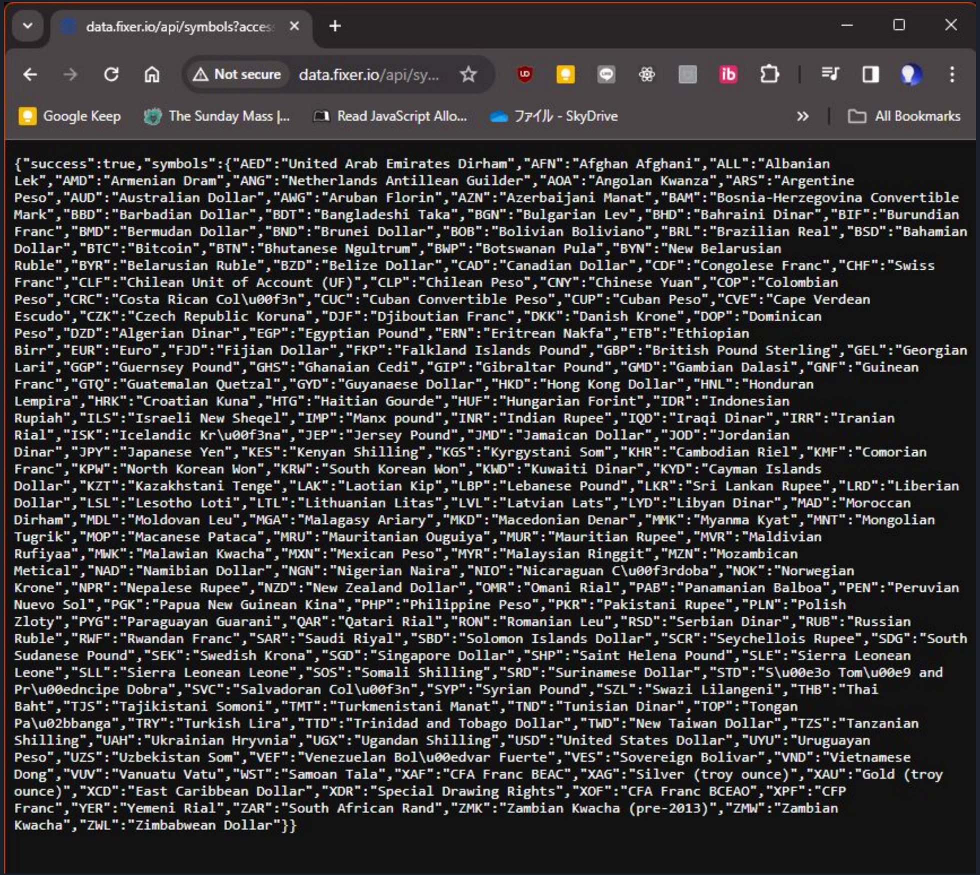
https is an encrypted version of http.

It’s safer by default, because it encrypts all communication between the browser and the servers. If a bad agent tries to to this communication, they will have a tougher time taking data out of it, steal your credentials, and do Very Bad Things with them.

In real world applications, you will always want to support https.

Exploring REST APIs

This is the response we got.

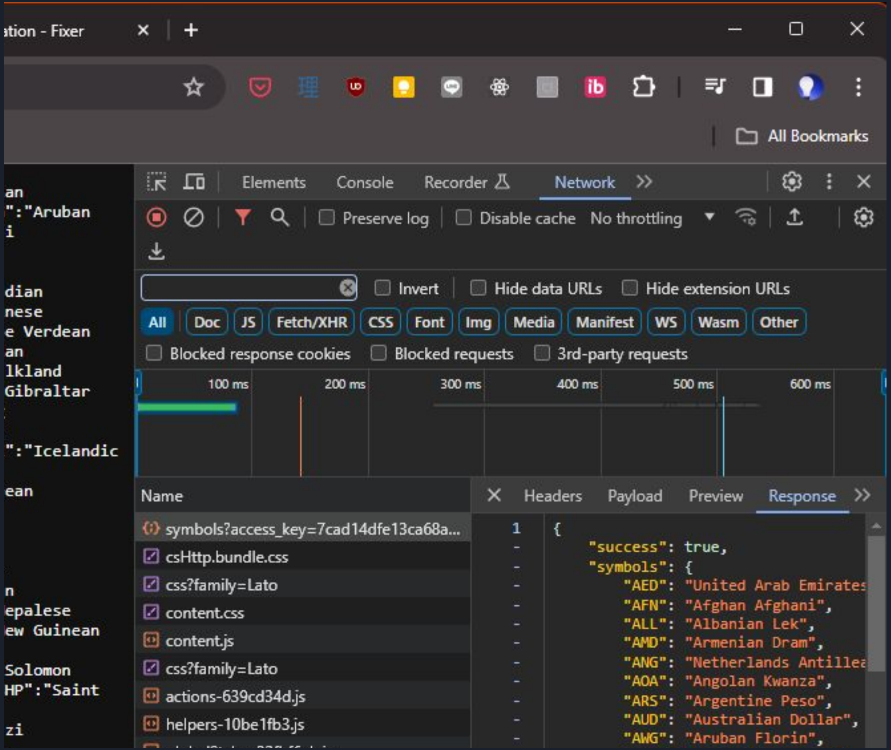


```
{"success":true,"symbols":{"AED":"United Arab Emirates Dirham","AFN":"Afghan Afghani","ALL":"Albanian
Lek","AMD":"Armenian Dram","ANG":"Netherlands Antillean Guilder","AOA":"Angolan Kwanza","ARS":"Argentine
Peso","AUD":"Australian Dollar","AWG":"Aruban Florin","AZN":"Azerbaijani Manat","BAM":"Bosnia-Herzegovina Convertible
Mark","BBD":"Barbadian Dollar","BDT":"Bangladeshi Taka","BGN":"Bulgarian Lev","BHD":"Bahraini Dinar","BIF":"Burundian
Franc","BMD":"Bermudan Dollar","BND":"Brunei Dollar","BOB":"Bolivian Boliviano","BRL":"Brazilian Real","BSD":"Bahamian
Dollar","BTC":"Bitcoin","BTN":"Bhutanese Ngultrum","BWP":"Botswanan Pula","BYN":"New Belarusian
Ruble","BYR":"Belarusian Ruble","BZD":"Belize Dollar","CAD":"Canadian Dollar","CDF":"Congolese Franc","CHF":"Swiss
Franc","CLF":"Chilean Unit of Account (UF)","CLP":"Chilean Peso","CNY":"Chinese Yuan","COP":"Colombian
Peso","CRC":"Costa Rican Col\u00f3n","CUC":"Cuban Convertible Peso","CUP":"Cuban Peso","CVE":"Cape Verdean
Escudo","CZK":"Czech Republic Koruna","DJF":"Djiboutian Franc","DKK":"Danish Krone","DOP":"Dominican
Peso","DZD":"Algerian Dinar","EGP":"Egyptian Pound","ERN":"Eritrean Nakfa","ETB":"Ethiopian
Birr","EUR":"Euro","FJD":"Fijian Dollar","FKP":"Falkland Islands Pound","GBP":"British Pound Sterling","GEL":"Georgian
Lari","GGP":"Guernsey Pound","GHS":"Ghanaian Cedi","GIP":"Gibraltar Pound","GMD":"Gambian Dalasi","GNF":"Guinean
Franc","GTQ":"Guatemalan Quetzal","GYD":"Guyanaese Dollar","HKD":"Hong Kong Dollar","HNL":"Honduran
Lempira","HRK":"Croatian Kuna","HTG":"Haitian Gourde","HUF":"Hungarian Forint","IDR":"Indonesian
Rupiah","ILS":"Israeli New Sheqel","IMP":"Manx pound","INR":"Indian Rupee","IQD":"Iraqi Dinar","IRR":"Iranian
Rial","ISK":"Icelandic Kr\u00f3na","JEP":"Jersey Pound","JMD":"Jamaican Dollar","JOD":"Jordanian
Dinar","JPY":"Japanese Yen","KES":"Kenyan Shilling","KGS":"Kyrgystani Som","KHR":"Cambodian Riel","KMF":"Comorian
Franc","KPW":"North Korean Won","KRW":"South Korean Won","KWD":"Kuwaiti Dinar","KYD":"Cayman Islands
Dollar","KZT":"Kazakhstani Tenge","LAK":"Laotian Kip","LBP":"Lebanese Pound","LKR":"Sri Lankan Rupee","LRD":"Liberian
Dollar","LSL":"Lesotho Loti","LTL":"Lithuanian Litas","LVL":"Latvian Lats","LYD":"Libyan Dinar","MAD":"Moroccan
Dirham","MDL":"Moldovan Leu","MGA":"Malagasy Ariary","MKD":"Macedonian Denar","MMK":"Myanma Kyat","MNT":"Mongolian
Tugrik","MOP":"Macanese Pataca","MRU":"Mauritanian Ouguiya","MUR":"Mauritian Rupee","MVR":"Maldivian
Rufiyaa","MWK":"Malawian Kwacha","MXN":"Mexican Peso","MYR":"Malaysian Ringgit","MZN":"Mozambican
Metical","NAD":"Namibian Dollar","NGN":"Nigerian Naira","NIO":"Nicaraguan C\u00f3rdoba","NOK":"Norwegian
Krone","NPR":"Nepalese Rupee","NZD":"New Zealand Dollar","OMR":"Omani Rial","PAB":"Panamanian Balboa","PEN":"Peruvian
Nuevo Sol","PGK":"Papua New Guinean Kina","PHP":"Philippine Peso","PKR":"Pakistani Rupee","PLN":"Polish
Zloty","PYG":"Paraguayan Guarani","QAR":"Qatari Rial","RON":"Romanian Leu","RSD":"Serbian Dinar","RUB":"Russian
Ruble","RWF":"Rwandan Franc","SAR":"Saudi Riyal","SBD":"Solomon Islands Dollar","SCR":"Seychellois Rupee","SDG":"South
Sudanese Pound","SEK":"Swedish Krona","SGD":"Singapore Dollar","SHP":"Saint Helena Pound","SLE":"Sierra Leonean
Leone","SLL":"Sierra Leonean Leone","SOS":"Somali Shilling","SRD":"Surinamese Dollar","STD":"S\u00e3o Tom\u00e9 and
Pr\u00edncipe Dobra","SVC":"Salvadoran Col\u00f3n","SYP":"Syrian Pound","SZL":"Swazi Lilangeni","THB":"Thai
Baht","TJS":"Tajikistani Somoni","TMT":"Turkmenistani Manat","TND":"Tunisian Dinar","TOP":"Tongan
Pa\u02bbanga","TRY":"Turkish Lira","TTD":"Trinidad and Tobago Dollar","TWD":"New Taiwan Dollar","TZS":"Tanzanian
Shilling","UAH":"Ukrainian Hryvnia","UGX":"Ugandan Shilling","USD":"United States Dollar","UYU":"Uruguayan
Peso","UZS":"Uzbekistan Som","VEF":"Venezuelan Bol\u00edvar Fuerte","VES":"Sovereign Bolivar","VND":"Vietnamese
Dong","VUV":"Vanuatu Vatu","WST":"Samoan Tala","XAF":"CFA Franc BEAC","XAG":"Silver (troy ounce)","XAU":"Gold (troy
ounce)","XCD":"East Caribbean Dollar","XDR":"Special Drawing Rights","XOF":"CFA Franc BCEAO","XPF":"CFP
Franc","YER":"Yemeni Rial","ZAR":"South African Rand","ZMK":"Zambian Kwacha (pre-2013)","ZMW":"Zambian
Kwacha","ZWL":"Zimbabwean Dollar"}}
```

Exploring REST APIs

At this point, open the “Network” tab of your browser’s Developer Tools, refresh the page, and click the first item in the list. You should be getting the same data, in a nicer format.

Note that this data, in the JSON format, is pretty much just a Javascript object.



```
an
":"Aruban
i
dian
nese
e Verdean
an
lkland
Gibraltar
":"Icelandic
ean
n
epalese
ew Guinean
Solomon
HP":"Saint
zi
```

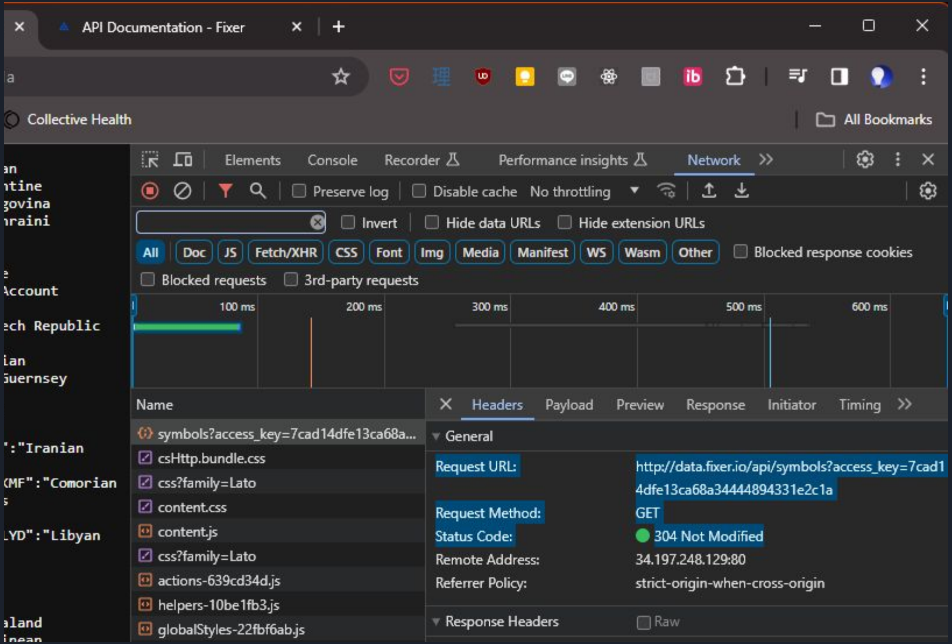
symbols?access_key=7cad14dfe13ca68a...
csHttp.bundle.css
css?family=Lato
content.css
content.js
css?family=Lato
actions-639cd34d.js
helpers-10be1fb3.js

```
{
    "success": true,
    "symbols": {
        "AED": "United Arab Emirates
        "AFN": "Afghan Afghani",
        "ALL": "Albanian Lek",
        "AMD": "Armenian Dram",
        "ANG": "Netherlands Antillea
        "AOA": "Angolan Kwanza",
        "ARS": "Argentine Peso",
        "AUD": "Australian Dollar",
        "AWG": "Aruban Florin",
```

Exploring REST APIs

Now click on “Headers”.

- What is the URL listed here?
  What are the parameters?
- What method is listed?
- What is the status code?

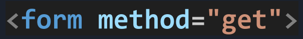


symbols?access_key=7cad14dfe13ca68a...
csHttp.bundle.css
css?family=Lato
content.css
content.js
css?family=Lato
actions-639cd34d.js
helpers-10be1fb3.js
globalStyles-22fbf6ab.js

Request URL: http://data.fixer.io/api/symbols?access_key=7cad14dfe13ca68a34444894331e2c1a
Request Method: GET
Status Code: ● 304 Not Modified
Remote Address: 34.197.248.129:80
Referrer Policy: strict-origin-when-cross-origin

Request methods

Remember when we were sending forms? We mentioned that a <form> has a method, and it can be GET or POST:

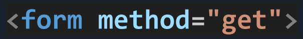


```html
<form method="get">
```

- What was the difference between GET and POST?

Request methods

Remember when we were sending forms? We mentioned that a <form> has a method, and it can be GET or POST:



```
<form method="get">
```

- When using GET, the form data gets codified into the URL.
- When using POST, the form data gets sent within the HTTP request itself.

Request methods

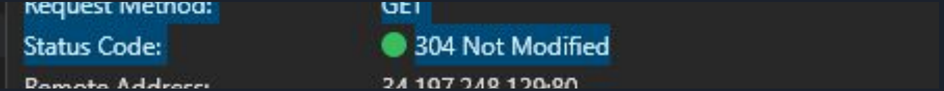
These GET and POST “methods” are just ways to do a request, defined in HTTP itself.

Each one of these methods (also called “verbs”) has a meaning: they are used to do something specific with the data in the endpoint.

Some of the main ones:

`GET`	Get some data, or a web page or other file.
`POST`	Send data, probably saving it or having other effects.
`PUT`	Update: send data that will replace the current version of what’s in the server.
`DELETE`	Delete data.

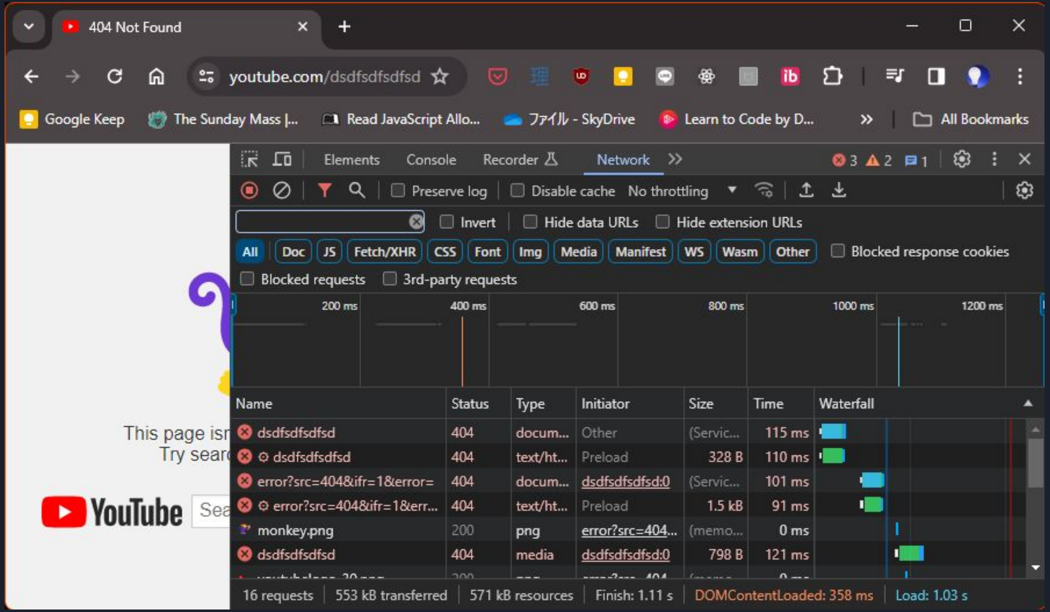
Response status codes



Request Method: GET
Status Code: ● 304 Not Modified
Remote Address: 34.197.248.129:80

In HTTP, responses may or may not contain data, but they will always at least contain a status code.

You may already be familiar with the “404 Not Found” error message that appears when a page in a website does not exist. This is actually a status code.



Name	Status	Type	Initiator	Size	Time	Waterfall
dsdfsdfsdfsd	404	docum...	Other	(Servic...	115 ms	
⚙ dsdfsdfsdfsd	404	text/ht...	Preload	328 B	110 ms	
error?src=404&ifr=1&error=	404	docum...	dsdfsdfsdfsd:0	(Servic...	101 ms	
⚙ error?src=404&ifr=1&err...	404	text/ht...	Preload	1.5 kB	91 ms	
monkey.png	200	png	error?src=404...	(memo...	0 ms	
dsdfsdfsdfsd	404	media	dsdfsdfsdfsd:0	798 B	121 ms	

16 requests | 553 kB transferred | 571 kB resources | Finish: 1.11 s | DOMContentLoaded: 358 ms | Load: 1.03 s

Response status codes

There are many status codes, classified in five groups:

100-199	Informational responses (eg. 102 is “processing”)
200-299	Successful responses (eg. 200 is “OK” and comes with data)
300-399	Redirections (eg. 301 is “Moved permanently” and allows to load another URL instead)
400-499	Client errors (eg. 404; also, 401 is “Unauthorized”)
500-599	Server errors (eg. 500 is “Internal Server Error”, eg. the program in the server broke)

Testing APIs: the `curl` command

While we can test the fixer.io API calls in the webpage, this is not true of most APIs. We need a way to test them.

The standard way to do this is the `curl` command. This exists in Windows, Mac and all Linux / Unix systems.

`curl` allows you to make an HTTP request through the command line and get its response, no browser needed.

Created by HideMaru
from Noun Project

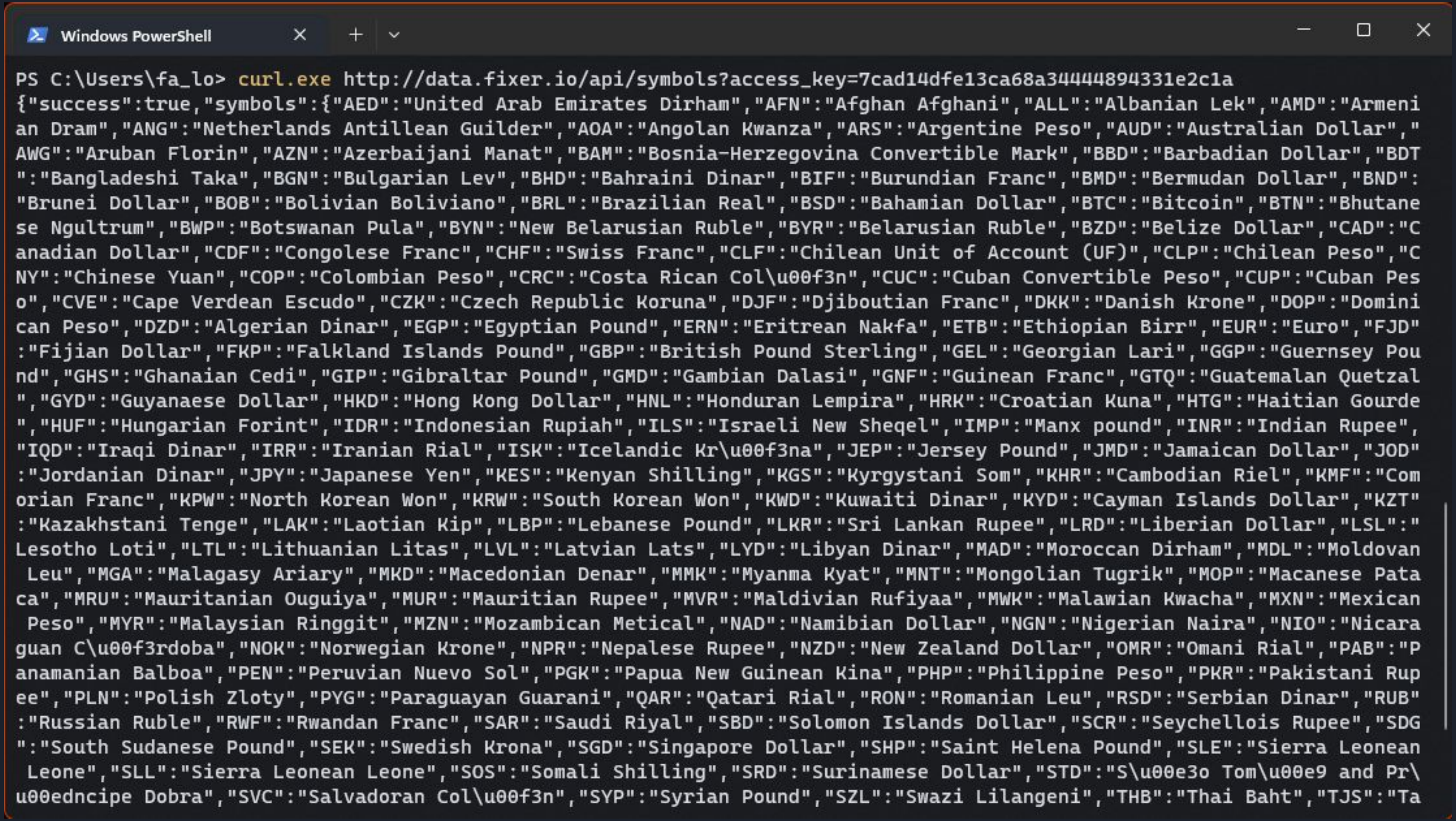
Testing APIs: the `curl` command

To do this exercise you may need administrative access in your computer.

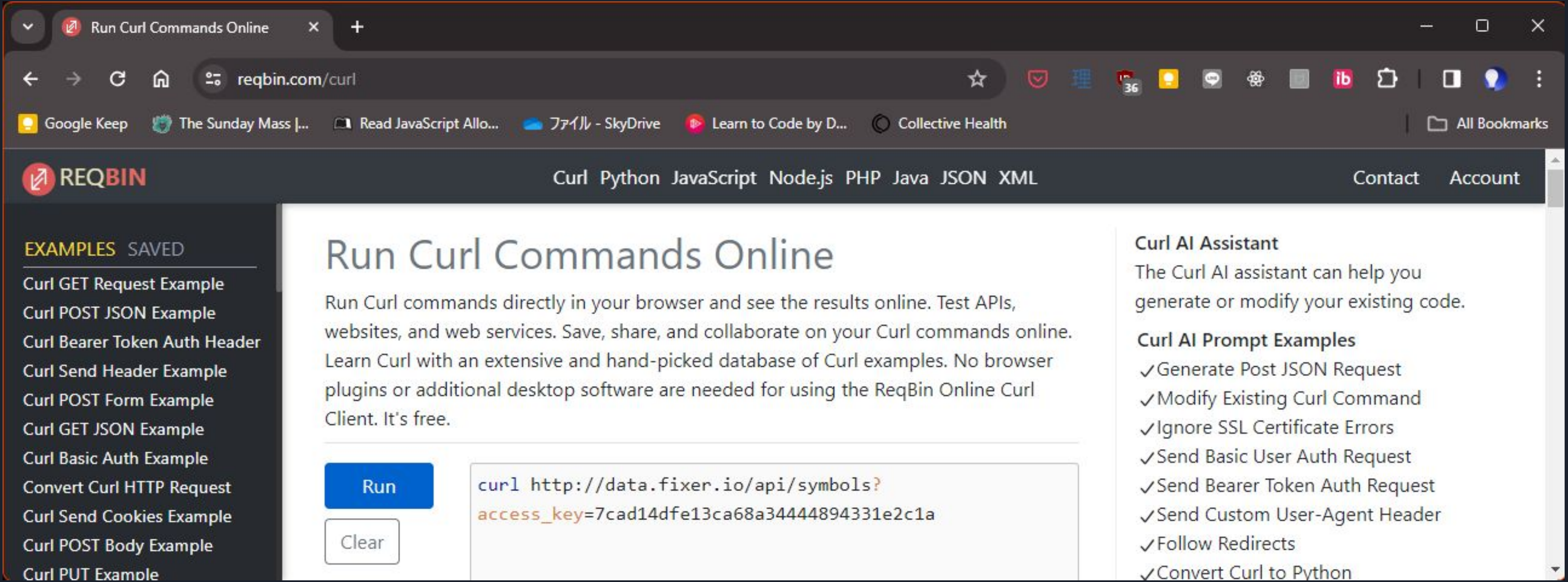
- Open your computer's command line or shell (Terminal on Mac / Linux / Unix, Terminal or PowerShell on Windows).
- Copy the URL from the page with the http://data.fixer.io/api/symbols response, including your API key.
- In the command line, type "curl", a space, and paste the URL; then press Enter.
  - In Windows, type "curl.exe" instead.
- You should get the same response that you got in the browser.

If you're not able to do this exercise in your computer's command line or shell, use this webpage instead: https://reqbin.com/curl

HideMaru for Noun Project



```
PS C:\Users\fa_lo> curl.exe http://data.fixer.io/api/symbols?access_key=7cad14dfe13ca68a34444894331e2c1a
{"success":true,"symbols":{"AED":"United Arab Emirates Dirham","AFN":"Afghan Afghani","ALL":"Albanian Lek","AMD":"Armeni
an Dram","ANG":"Netherlands Antillean Guilder","AOA":"Angolan Kwanza","ARS":"Argentine Peso","AUD":"Australian Dollar","
AWG":"Aruban Florin","AZN":"Azerbaijani Manat","BAM":"Bosnia-Herzegovina Convertible Mark","BBD":"Barbadian Dollar","BDT
":"Bangladeshi Taka","BGN":"Bulgarian Lev","BHD":"Bahraini Dinar","BIF":"Burundian Franc","BMD":"Bermudan Dollar","BND":
"Brunei Dollar","BOB":"Bolivian Boliviano","BRL":"Brazilian Real","BSD":"Bahamian Dollar","BTC":"Bitcoin","BTN":"Bhutane
se Ngultrum","BWP":"Botswanan Pula","BYN":"New Belarusian Ruble","BYR":"Belarusian Ruble","BZD":"Belize Dollar","CAD":"C
anadian Dollar","CDF":"Congolese Franc","CHF":"Swiss Franc","CLF":"Chilean Unit of Account (UF)","CLP":"Chilean Peso","C
NY":"Chinese Yuan","COP":"Colombian Peso","CRC":"Costa Rican Col\u00f3n","CUC":"Cuban Convertible Peso","CUP":"Cuban Pes
o","CVE":"Cape Verdean Escudo","CZK":"Czech Republic Koruna","DJF":"Djiboutian Franc","DKK":"Danish Krone","DOP":"Domini
can Peso","DZD":"Algerian Dinar","EGP":"Egyptian Pound","ERN":"Eritrean Nakfa","ETB":"Ethiopian Birr","EUR":"Euro","FJD"
:"Fijian Dollar","FKP":"Falkland Islands Pound","GBP":"British Pound Sterling","GEL":"Georgian Lari","GGP":"Guernsey Pou
nd","GHS":"Ghanaian Cedi","GIP":"Gibraltar Pound","GMD":"Gambian Dalasi","GNF":"Guinean Franc","GTQ":"Guatemalan Quetzal
","GYD":"Guyanaese Dollar","HKD":"Hong Kong Dollar","HNL":"Honduran Lempira","HRK":"Croatian Kuna","HTG":"Haitian Gourde
","HUF":"Hungarian Forint","IDR":"Indonesian Rupiah","ILS":"Israeli New Sheqel","IMP":"Manx pound","INR":"Indian Rupee",
"IQD":"Iraqi Dinar","IRR":"Iranian Rial","ISK":"Icelandic Kr\u00f3na","JEP":"Jersey Pound","JMD":"Jamaican Dollar","JOD"
:"Jordanian Dinar","JPY":"Japanese Yen","KES":"Kenyan Shilling","KGS":"Kyrgystani Som","KHR":"Cambodian Riel","KMF":"Com
orian Franc","KPW":"North Korean Won","KRW":"South Korean Won","KWD":"Kuwaiti Dinar","KYD":"Cayman Islands Dollar","KZT"
:"Kazakhstani Tenge","LAK":"Laotian Kip","LBP":"Lebanese Pound","LKR":"Sri Lankan Rupee","LRD":"Liberian Dollar","LSL":"
Lesotho Loti","LTL":"Lithuanian Litas","LVL":"Latvian Lats","LYD":"Libyan Dinar","MAD":"Moroccan Dirham","MDL":"Moldovan
 Leu","MGA":"Malagasy Ariary","MKD":"Macedonian Denar","MMK":"Myanma Kyat","MNT":"Mongolian Tugrik","MOP":"Macanese Pata
ca","MRU":"Mauritanian Ouguiya","MUR":"Mauritian Rupee","MVR":"Maldivian Rufiyaa","MWK":"Malawian Kwacha","MXN":"Mexican
 Peso","MYR":"Malaysian Ringgit","MZN":"Mozambican Metical","NAD":"Namibian Dollar","NGN":"Nigerian Naira","NIO":"Nicara
guan C\u00f3rdoba","NOK":"Norwegian Krone","NPR":"Nepalese Rupee","NZD":"New Zealand Dollar","OMR":"Omani Rial","PAB":"P
anamanian Balboa","PEN":"Peruvian Nuevo Sol","PGK":"Papua New Guinean Kina","PHP":"Philippine Peso","PKR":"Pakistani Rup
ee","PLN":"Polish Zloty","PYG":"Paraguayan Guarani","QAR":"Qatari Rial","RON":"Romanian Leu","RSD":"Serbian Dinar","RUB"
:"Russian Ruble","RWF":"Rwandan Franc","SAR":"Saudi Riyal","SBD":"Solomon Islands Dollar","SCR":"Seychellois Rupee","SDG
":"South Sudanese Pound","SEK":"Swedish Krona","SGD":"Singapore Dollar","SHP":"Saint Helena Pound","SLE":"Sierra Leonean
 Leone","SLL":"Sierra Leonean Leone","SOS":"Somali Shilling","SRD":"Surinamese Dollar","STD":"S\u00e3o Tom\u00e9 and Pr\
u00edncipe Dobra","SVC":"Salvadoran Col\u00f3n","SYP":"Syrian Pound","SZL":"Swazi Lilangeni","THB":"Thai Baht","TJS":"Ta
```



Run Curl Commands Online

Run Curl commands directly in your browser and see the results online. Test APIs, websites, and web services. Save, share, and collaborate on your Curl commands online. Learn Curl with an extensive and hand-picked database of Curl examples. No browser plugins or additional desktop software are needed for using the ReqBin Online Curl Client. It's free.

```
curl http://data.fixer.io/api/symbols?
access_key=7cad14dfe13ca68a34444894331e2c1a
```

Curl AI Assistant

The Curl AI assistant can help you generate or modify your existing code.

Curl AI Prompt Examples

✓Generate Post JSON Request
✓Modify Existing Curl Command
✓Ignore SSL Certificate Errors
✓Send Basic User Auth Request
✓Send Bearer Token Auth Request
✓Send Custom User-Agent Header
✓Follow Redirects
✓Convert Curl to Python

Let’s remember the requirements

Travel Friends Club is asking you to build a currency conversion app that anyone can use.

- The user should be able to enter a number and select an origin currency and a destination currency (eg. “100 US dollars to Euros”).
- The system will respond with the converted amount, using the most up-to-date market rate for the conversion.

The app should be responsive and not use much bandwidth.

It’s not required to add the name of the club in the website.

How can we use the API?

Created by HideMaru
from Noun Project

Explore the fixer.io API to see what API calls we can use to build our app.

- How can we get the conversion rate between two currencies?
- How do we know what currencies are available?

HideMaru for Noun Project

Consuming APIs through Javascript

Consuming an API through Javascript

We will now learn how to use Javascript to read from a server.

This time around, we'll start with some code, and then we'll explain it.*

*If your teacher did a bad job explaining, you can try this page from Mozilla instead!

Created by HideMaru
from Noun Project

Reading from a server

- Create a web page like in the previous exercises.
  - Make sure you have `<body onload="init()">`
- In your script file, write the following:

```
/* Constants with capitalized names is a standard. */
const ACCESS_KEY = 'your key goes here';

function init() {
    const fetchPromise = fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`);
    console.log(fetchPromise);
}
```

HideMaru for Noun Project

Design and start developing

Advanced Challenge: pause

Warning: important concepts incoming!

I need your attention for the next slides.

`fetch` and other asynchronous functions

`fetch` is the main way we can use in Javascript to make an HTTP call to a server (and the only one it makes sense to learn).

fetch is a special kind of function: an asynchronous function (async). Let’s see what this is, and why we need that.

Blocking vs. async functions

Usually, all operations in a program happen sequentially, including calls to other functions.

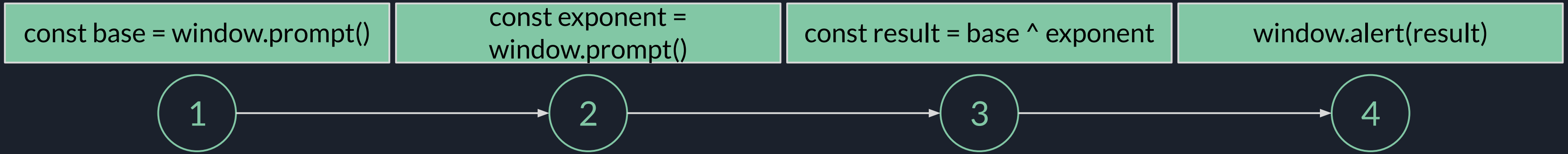
In the following program, each line is executed only after the previous one finished:

```
const base = window.prompt('Enter a number');
const exponent = window.prompt('Enter another number');
const result = base ^ exponent;
window.alert(`Your result is ${result}`);
```

`window.prompt()` shows a popup that asks for an input. When that happens, the program is blocked from continuing: it waits until the user enters the input (in this case, a number).

Blocking vs. async functions

You can imagine all of these calls as if they were in a single line or queue - each one waiting for the previous one to execute.



const base = window.prompt()

const exponent = window.prompt()

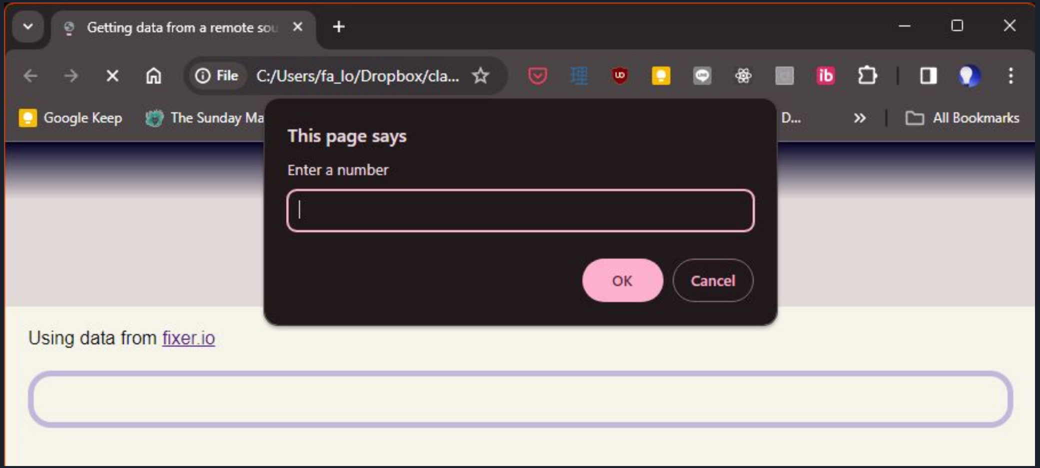
const result = base ^ exponent

window.alert(result)

Blocking vs. async functions

What would happen if the user never entered anything?

Exactly - the program would be forever stuck!



Getting data from a remote sou

File C:/Users/fa_lo/Dropbox/cla...

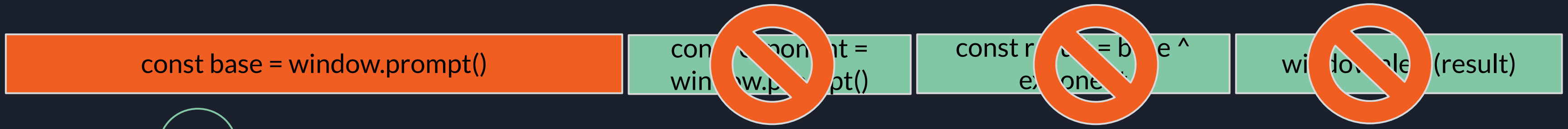
Google Keep · The Sunday Ma · D... · All Bookmarks

This page says

Enter a number

OK · Cancel

Using data from fixer.io

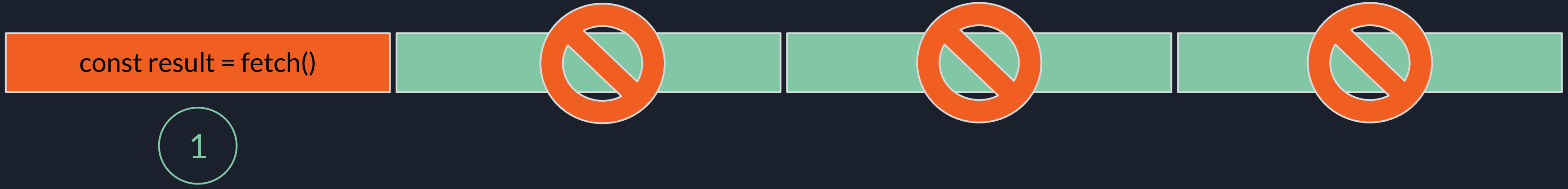


1 Waiting on the user…

A blocking `fetch()` won’t work

Now, server calls (HTTP or otherwise) are unpredictable. They may return quickly, take a very long time or probably never at all!

Because of this, if `fetch()` was a blocking function the same way as window.prompt() or almost all other Javascript functions, programs would be impossibly slow.



Waiting on the server…

What can we do, then?

What async means

The solution to this dilemma came in the form of asynchronous functions.

“Asynchronous” means that it does not execute in the same “line”. Async functions literally get put in a different “line of execution” than the main program. This means async functions can take as long as they need to, and the rest of the program can continue executing in the meantime.

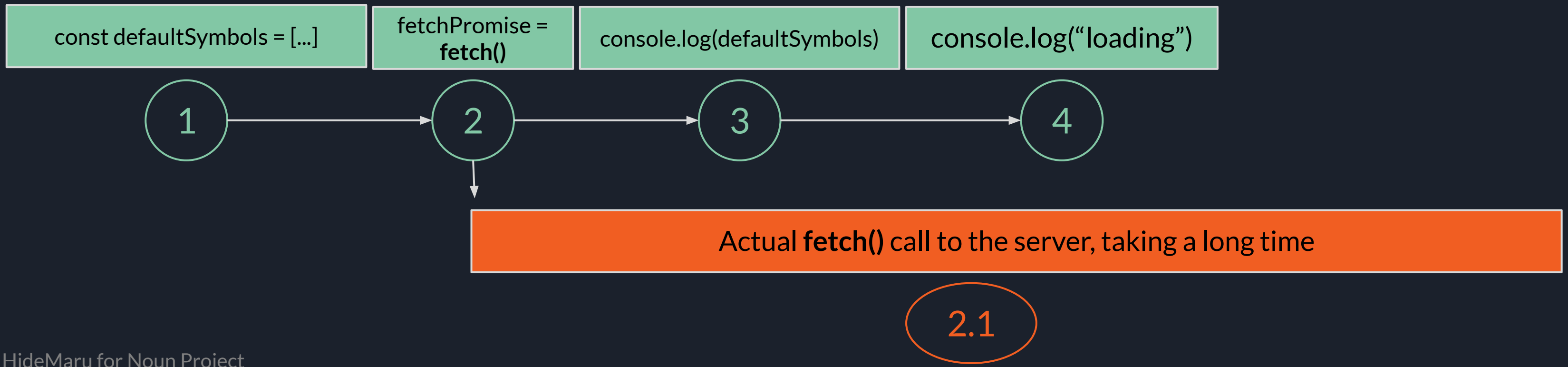
Created by HideMaru
from Noun Project

What async means

Consider this: (Add the highlighted code to the `init()` function in your script file.)

```
    const defaultSymbols = ['USD', 'EUR'];
    /* this takes a while */
    const fetchPromise = fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`);
    console.log(`symbols: ${defaultSymbols}`);
    console.log("loading");
```

We can do the console.log() calls without having to wait:



const defaultSymbols = [...]

fetchPromise = **fetch()**

console.log(defaultSymbols)

console.log("loading")

1

2

3

4

Actual **fetch()** call to the server, taking a long time

2.1

c n by HideMaru for Noun Project

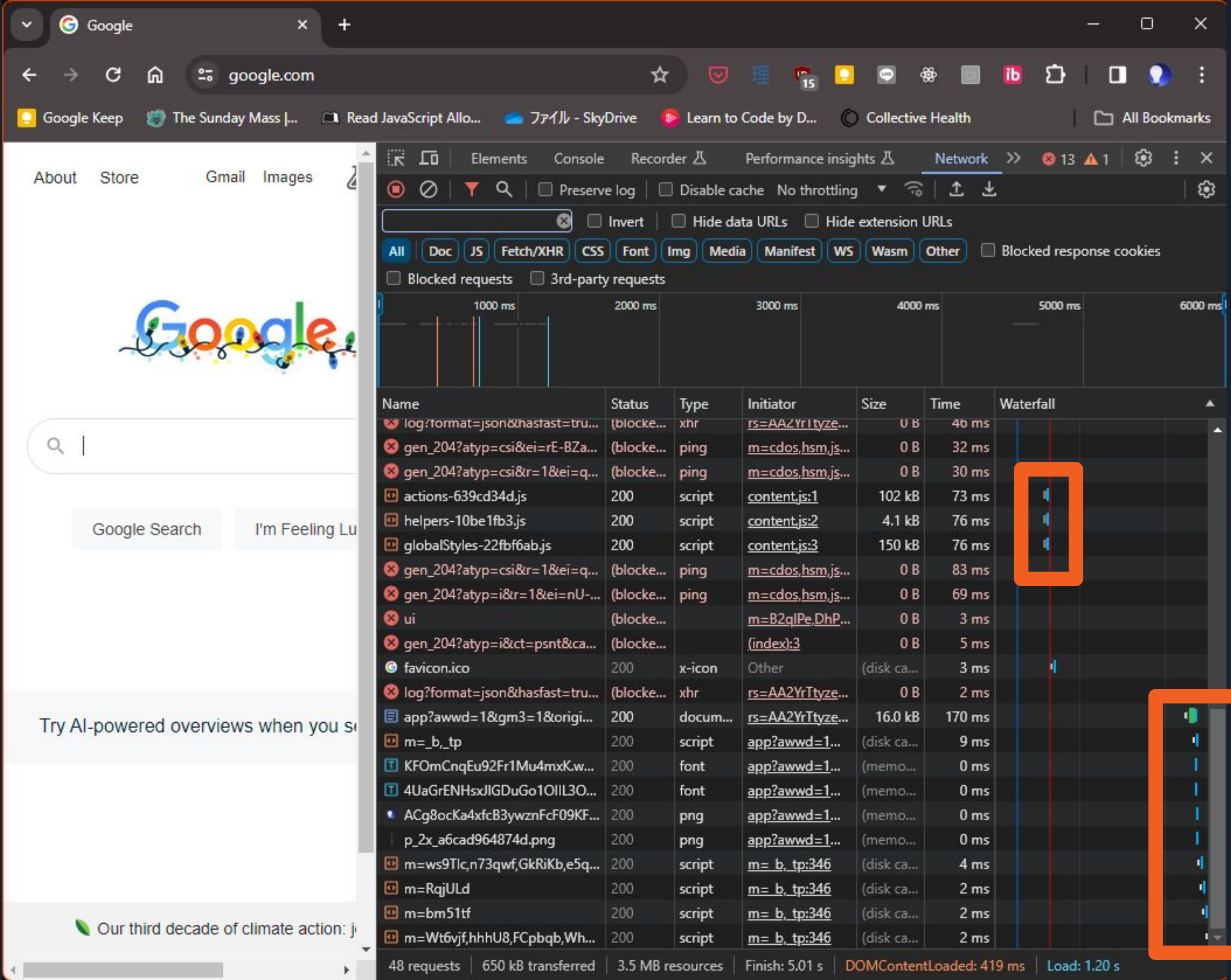
Created by HideMaru
from Noun Project

All file loads from servers are async

Actually, your browser does this all the time. You can see this:

- Open the developer console of your browser; select the “network” tab.
- Open or reload any page. See the “waterfall” column in the right side of the “network” tab: it shows at what moment each file is getting loaded.

Icon by HideMaru for Noun Project



Name	Status	Type	Initiator	Size	Time	Waterfall
log?format=json&hasfast=tru...	(blocke...	xhr	rs=AA2YrTtyze...	0 B	46 ms	
gen_204?atyp=csi&ei=rE-BZa...	(blocke...	ping	m=cdos,hsm,js...	0 B	32 ms	
gen_204?atyp=csi&r=1&ei=q...	(blocke...	ping	m=cdos,hsm,js...	0 B	30 ms	
actions-639cd34d.js	200	script	content.js:1	102 kB	73 ms	
helpers-10be1fb3.js	200	script	content.js:2	4.1 kB	76 ms	
globalStyles-22fbf6ab.js	200	script	content.js:3	150 kB	76 ms	
gen_204?atyp=csi&r=1&ei=q...	(blocke...	ping	m=cdos,hsm,js...	0 B	83 ms	
gen_204?atyp=i&r=1&ei=nU-...	(blocke...	ping	m=cdos,hsm,js...	0 B	69 ms	
ui	(blocke...		m=B2qlPe,DhP...	0 B	3 ms	
gen_204?atyp=i&ct=psnt&ca...	(blocke...		(index):3	0 B	5 ms	
favicon.ico	200	x-icon	Other	(disk ca...	3 ms	
log?format=json&hasfast=tru...	(blocke...	xhr	rs=AA2YrTtyze...	0 B	2 ms	
app?awwd=1&gm3=1&origi...	200	docum...	rs=AA2YrTtyze...	16.0 kB	170 ms	
m=_b,_tp	200	script	app?awwd=1...	(disk ca...	9 ms	
KFOmCnqEu92Fr1Mu4mxK.w...	200	font	app?awwd=1...	(memo...	0 ms	
4UaGrENHsxJlGDuGo1OIlL3O...	200	font	app?awwd=1...	(memo...	0 ms	
ACg8ocKa4xfcB3ywznFcF09KF...	200	png	app?awwd=1...	(memo...	0 ms	
p_2x_a6cad964874d.png	200	png	app?awwd=1...	(memo...	0 ms	
m=ws9Tlc,n73qwf,GkRiKb,e5q...	200	script	m=_b,_tp:346	(disk ca...	4 ms	
m=RqjULd	200	script	m=_b,_tp:346	(disk ca...	2 ms	
m=bm51tf	200	script	m=_b,_tp:346	(disk ca...	2 ms	
m=Wt6vjf,hhhU8,FCpbqb,Wh...	200	script	m=_b,_tp:346	(disk ca...	2 ms	

48 requests | 650 kB transferred | 3.5 MB resources | Finish: 5.01 s | DOMContentLoaded: 419 ms | Load: 1.20 s

All file loads from servers are async

Files are all being loaded at roughly the same times.

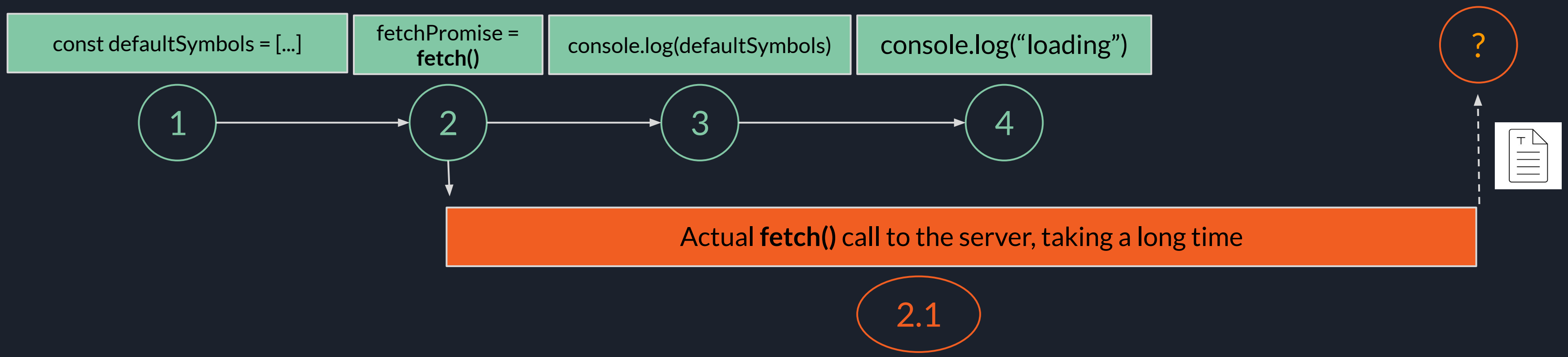
This can happen because they are loaded asynchronously: instead of waiting for file A, then for file B, then for file C… the browser can just start loading many files at once.

- It’s not perfect: the browser cannot load everything at the same time. There is a limit to it

Async functions return promises

So, if our fetch() function can execute at its own pace, and it will finish later… how can we use its results?

- We can't get them immediately, so how can we use them?
- Maybe the server returns after the program finishes!?



const defaultSymbols = [...]

fetchPromise = **fetch()**

console.log(defaultSymbols)

console.log("loading")

1

2

3

4

?

Actual **fetch()** call to the server, taking a long time

2.1

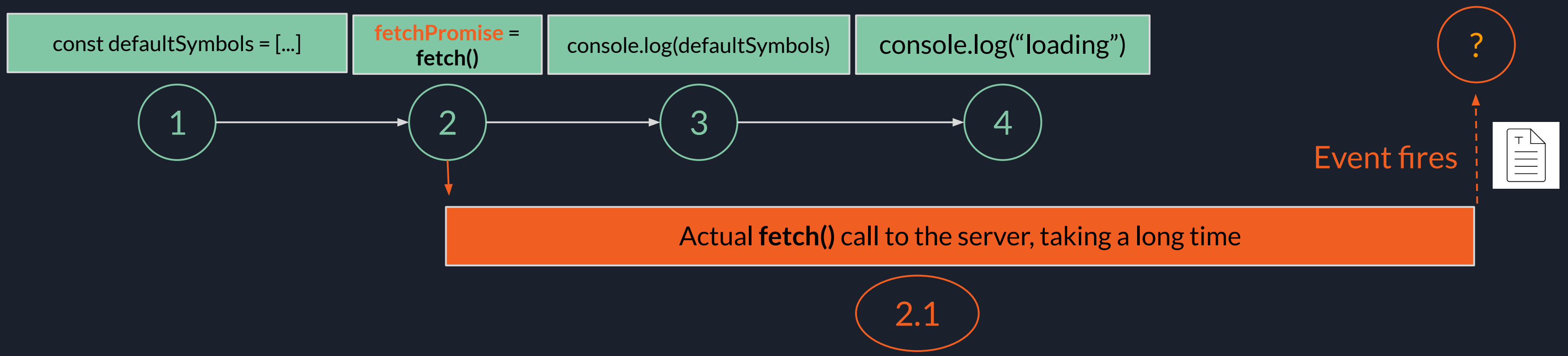
Async functions return promises

The solution is via a thing called a promise.

A promise is a special object that represents the fact that an async function is working somewhere else.

When the async function finishes its work, an event will fire.

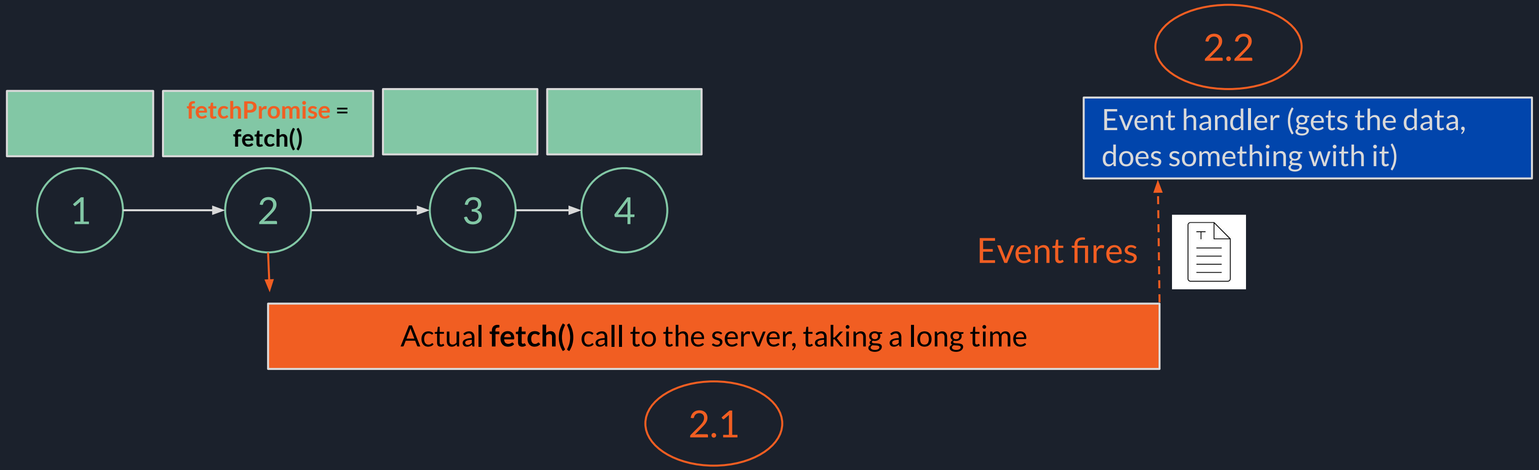
- This happens even if the original code had already ended!


const defaultSymbols = [...]
fetchPromise = fetch()
console.log(defaultSymbols)
console.log("loading")
?
1
2
3
4
Event fires
Actual fetch() call to the server, taking a long time
2.1

Async functions return promises

But, what will happen when that event fires?

For that, we provide an event handler, of course! (Same as with `onload()`, `onclick()`...)



fetchPromise = fetch()
1
2
3
4
2.2
Event handler (gets the data, does something with it)
Event fires
Actual fetch() call to the server, taking a long time
2.1

Working with promises

Any promise has two main events:

- `then()`: for when the promise resolves successfully
- `catch()`: for when the promise fails

Advanced Challenge: continue

Continue if you already know the following JavaScript concepts:

- fetch() and its response transformers
- Handling failures with .catch()
- Chaining

Normal version continues in the next slides. We will explain step by step.

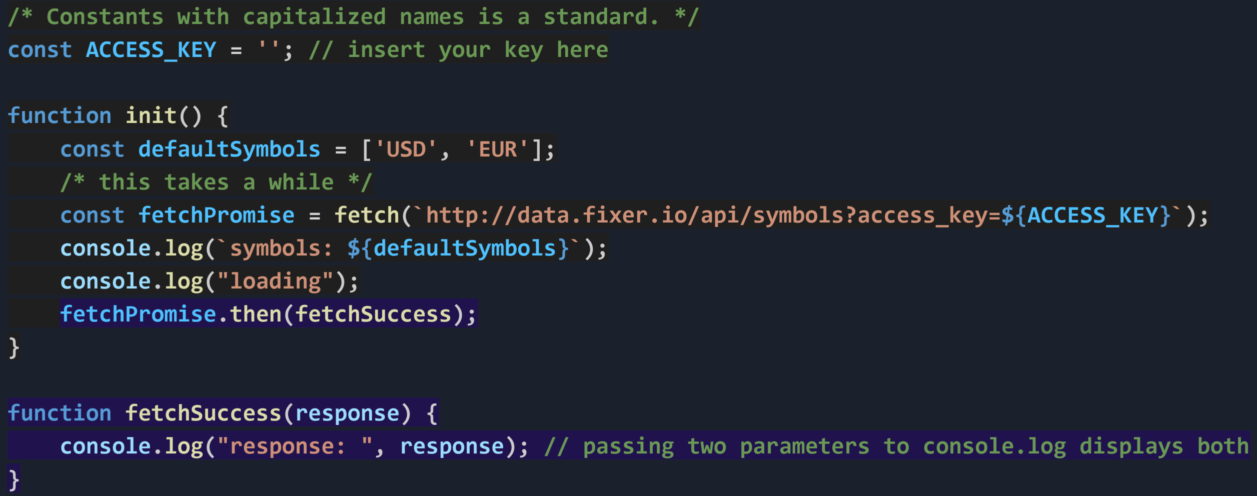
“Advanced” Icon by Read on freeicons.io

Good luck!

Created by HideMaru
from Noun Project

Let’s write a handler

In your script file, add the highlighted code:



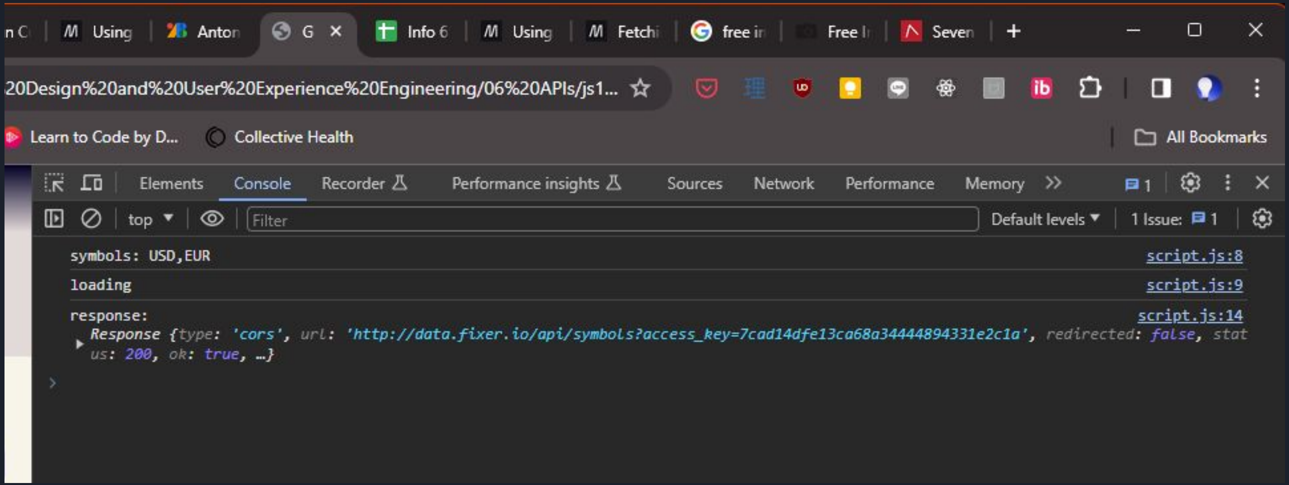
```
/* Constants with capitalized names is a standard. */
const ACCESS_KEY = ''; // insert your key here

function init() {
    const defaultSymbols = ['USD', 'EUR'];
    /* this takes a while */
    const fetchPromise = fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`);
    console.log(`symbols: ${defaultSymbols}`);
    console.log("loading");
    fetchPromise.then(fetchSuccess);
}

function fetchSuccess(response) {
    console.log("response: ", response); // passing two parameters to console.log displays both
}
```

HideMaru for Noun Project

Created by HideMaru
from Noun Project

Let’s write a handler

- Run the code. Take care of noticing the order in which the messages in your console appear.
  - Remember: open the developer console and select the “Console” tab to see this output.



```
symbols: USD,EUR                                                                script.js:8
loading                                                                         script.js:9
response:                                                                       script.js:14
▸ Response {type: 'cors', url: 'http://data.fixer.io/api/symbols?access_key=7cad14dfe13ca68a34444894331e2c1a', redirected: false, stat
  us: 200, ok: true, …}
```

HideMaru for Noun Project

Proof that this is async:

- If you’re curious, you can add console messages to the start and end of each function.
- Notice when each message appears.
  - The handler code may start before or after the main code ends!

Continuing to read...

Okay, we have our handler. Let's do something more interesting with the data we get from the server. For this, we'll use the response parameter of our handler.

- This parameter is an object that represents the response from the server.

Actually, because `fetch()` can get all kinds of data from any server, at this point it does not know how to read the data!

We know, though, that this is JSON data. The response object has a method that can read the data as JSON and give it to us: `response.json()`.

...except that this method is also async. *We will need another handler.*

Created by HideMaru
from Noun Project

Continuing to read...

Update your code with the following:

```
function fetchSuccess(response) {
    const readPromise = response.json();
    readPromise.then(readSuccess);
}

/* Callback for <response>.json.
   It receives the data from the response, already converted into JSON.
   A JSON object is practically a Javascript object, so we can use it as such. */
function readSuccess(jsonData) {
    console.log(jsonData);
}
```

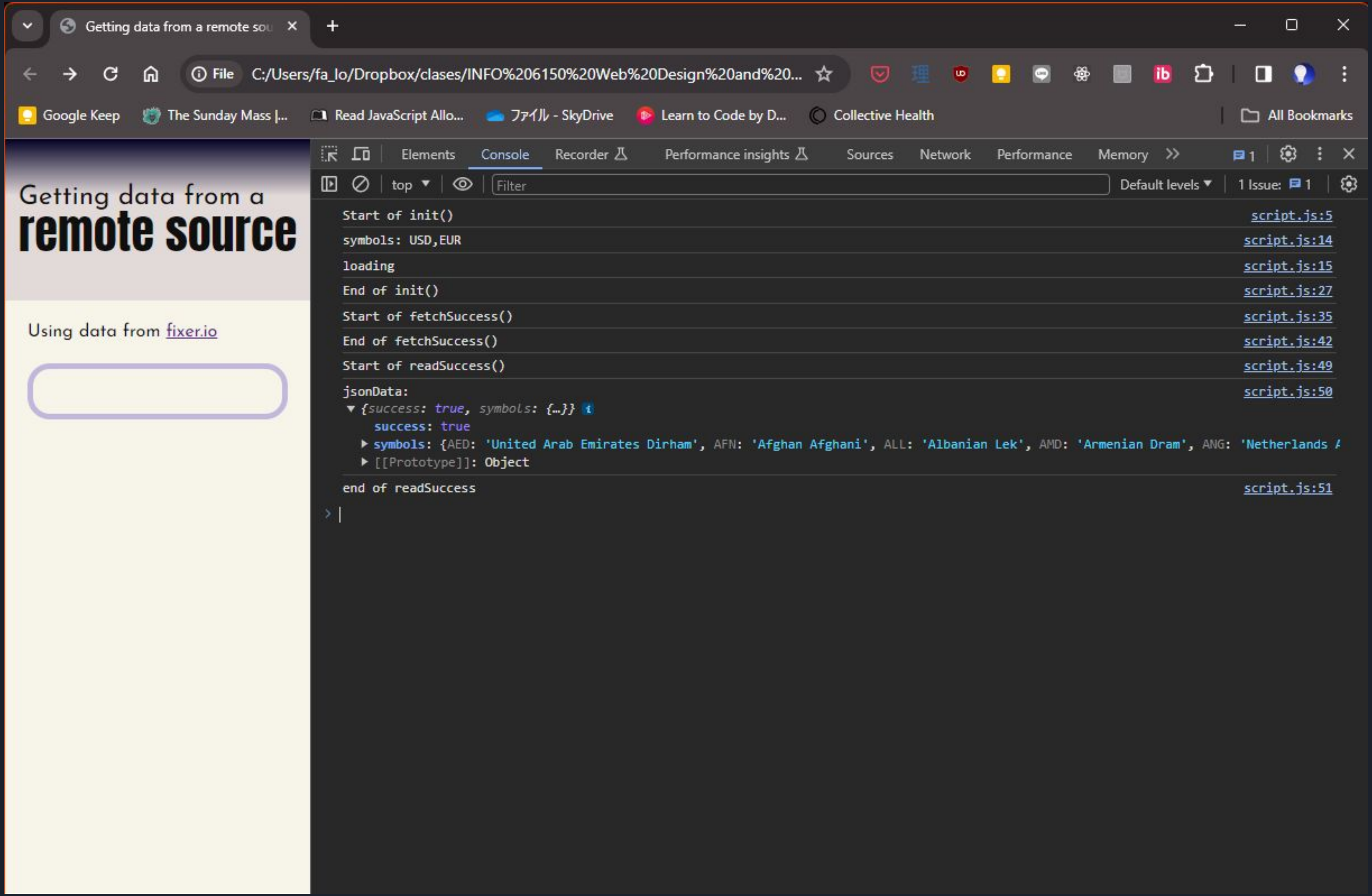
HideMaru for Noun Project

Continuing to read...

Created by HideMaru
from Noun Project

- Run your code. See what you get in the console.

HideMaru for Noun Project



Getting data from a
remote source

Using data from fixer.io

```
Start of init()                                          script.js:5
symbols: USD,EUR                                         script.js:14
loading                                                  script.js:15
End of init()                                            script.js:27
Start of fetchSuccess()                                  script.js:35
End of fetchSuccess()                                    script.js:42
Start of readSuccess()                                   script.js:49
jsonData:                                                script.js:50
▼ {success: true, symbols: {…}} i
    success: true
  ▶ symbols: {AED: 'United Arab Emirates Dirham', AFN: 'Afghan Afghani', ALL: 'Albanian Lek', AMD: 'Armenian Dram', ANG: 'Netherlands A
  ▶ [[Prototype]]: Object
end of readSuccess                                       script.js:51
>
```

The chain notation and callbacks

At this point, you’ll notice that we’re getting full of promises and handlers. This is… *a lot of code*. And we’re not done - we haven’t even used the data for anything useful!

We need a way to make working with promises much simpler, or we’ll never end. Fortunately, there is such a way: enter the chain notation.

The chain notation

This:

```
const someObject = functionOne();
const result = someObject.someMethod();
```

can also be written like this:

```
const result = functionOne().someMethod();
```

The chain notation

So we can replace this:

```
const fetchPromise = fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`);
fetchPromise.then(fetchSuccess);
```

with this:

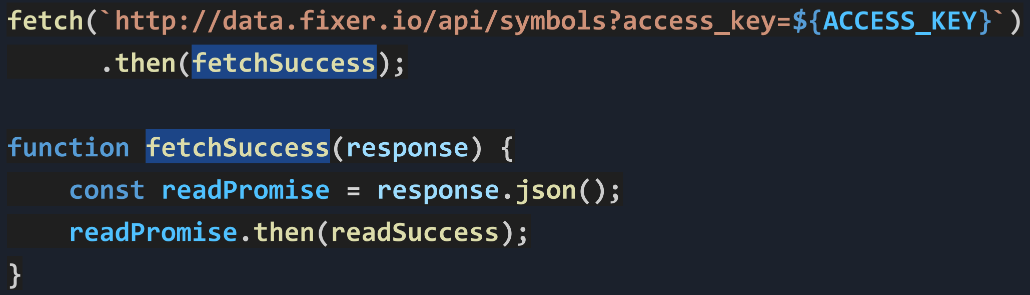
```
    fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`)
      .then(fetchSuccess);
```

`fetch` is still returning a promise - but instead of saving it into a constant, we’re calling a method of that promise right away.

Callbacks

As we said before, a callback is a function that is the parameter of another function.

In this code



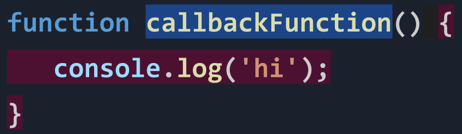
```
fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`)
      .then(fetchSuccess);

function fetchSuccess(response) {
    const readPromise = response.json();
    readPromise.then(readSuccess);
}
```

`fetchSuccess` is a callback, and `readSuccess` is another callback.

Callbacks

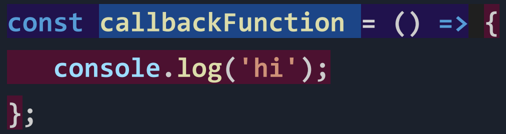
In modern Javascript, it’s much more common to see arrow functions instead of traditional ones.

Instead of doing this:



```
function callbackFunction() {
   console.log('hi');
}
```

it’s more common to see this:

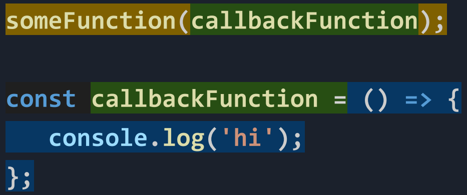


```
const callbackFunction = () => {
   console.log('hi');
};
```

Callbacks

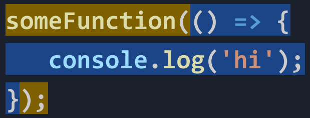
Now, you'll notice that we are not really using `fetchSuccess` for anything else than as a callback.

To make the code even simpler and shorter, we can then just define it in the parameter itself. So we go from this:



```
someFunction(callbackFunction);

const callbackFunction = () => {
    console.log('hi');
};
```

to this:



```
someFunction(() => {
    console.log('hi');
});
```

Callbacks

Now, you'll notice that we are not really using `fetchSuccess` for anything else than as a callback.

To make the code even simpler and shorter, we can then just define it in the parameter itself. So we go from this:

```
someFunction(callbackFunction);

const callbackFunction = () => {
   console.log('hi');
}
```

to this:

We're doing the same, with less words!

```
someFunction(() => {
   console.log('hi');
});
```

All together now!

Let’s get rid of `fetchSuccess` and chain it instead:

```
function init() {
    const defaultSymbols = ['USD', 'EUR'];
    fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`)
      .then((response) => {
         const readPromise = response.json();
         readPromise.then(readSuccess);
      });
    console.log(`symbols: ${defaultSymbols}`);
    console.log("loading");
}
```

All together now!

Let’s get rid of `fetchSuccess` and chain it instead:

```
function init() {
    const defaultSymbols = ['USD', 'EUR'];
    fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`)
      .then((response) => {
         const readPromise = response.json();
         readPromise.then(readSuccess);
      });
    console.log(`symbols: ${defaultSymbols}`);
    console.log("loading");
}

function readSuccess(jsonData) {
    console.log('jsonData: ', jsonData);
}
```

readSuccess is also a callback - let’s chain it too!

All together now!

Our code, much shorter and cleaner:

```
function init() {
    const defaultSymbols = ['USD', 'EUR'];
    fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`)
      .then((response) => {
         response.json()
            .then((jsonData) => {
               console.log('jsonData: ', jsonData);
            });
      });
    console.log(`symbols: ${defaultSymbols}`);
    console.log("loading");
}
```

Handling failures

Right now, if the call to the server fails, our program will horribly break. The user will know nothing of what happened. That is terrible UX!

fetch allows us to chain another method: `.catch()`.

`fetch.catch()` gets a parameter, also a callback. This callback gets as parameter an `Error` object. (`Error` is a Javascript-defined class representing errors.)

Handling failures

```
fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`)
  .then((response) => {
     /* code omitted */
  }).catch((err) => {
     console.error(err);
     window.alert(`Error loading symbols: ${err}`)
  });
```

Handling failures

```
fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`)
  .then((response) => {
     /* code omitted */
  }).catch((err) => {
     console.error(err);
     window.alert(`Error loading symbols: ${err}`)
  });
```

Our program so far

(Some code and comments have been omitted for clarity.)

```
const ACCESS_KEY = ''; // insert your key here

function init() {
    fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`)
      .then((response) => {
         response.json()
            .then((jsonData) => {
               console.log('jsonData: ', jsonData);
            });
      }).catch((err) => {
         console.error(err);
         window.alert(`Error loading symbols: ${err}`)
      });
}
```

From JSON data to HTML nodes

Let’s now do something useful with the data we got from the server.

```
console.log('jsonData: ', jsonData);
```

From the documentation, and also from our tests, we know this data has the following shape:

```
{
   "success": true,
   "symbols": {
     "AED": "United Arab Emirates Dirham",
     "AFN": "Afghan Afghani",
     "ALL": "Albanian Lek",
```

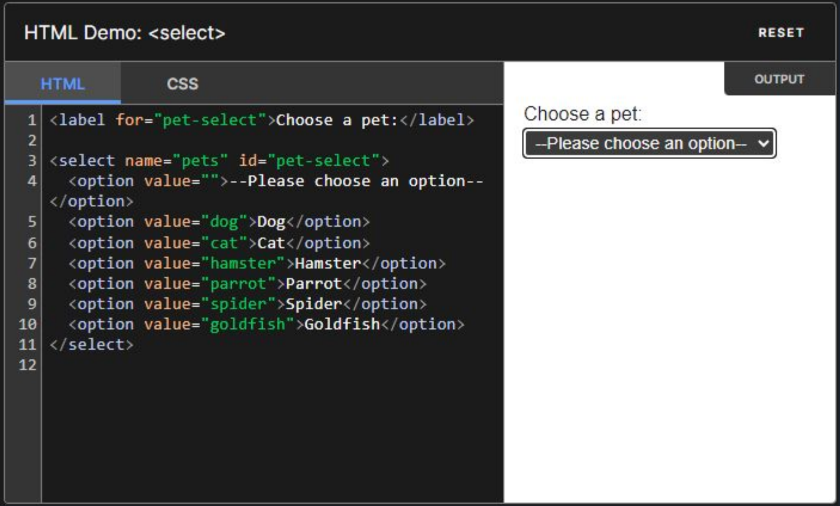
For now let’s output all of those “symbols” in the screen.

From JSON data to HTML nodes

Created by HideMaru
from Noun Project

Take a few minutes to research how to output this data into a <select> box.

- `<select>` allows to choose one or many options from a list. It's better than checkboxes or radio buttons if there are too many options.
- We want to have a single `<select>`, then create one option per each currency that the server recognizes.
- It's okay to write the `<select>` in the HTML directly, and just fill it in using Javascript.



```
<label for="pet-select">Choose a pet:</label>

<select name="pets" id="pet-select">
  <option value="">--Please choose an option--</option>
  <option value="dog">Dog</option>
  <option value="cat">Cat</option>
  <option value="hamster">Hamster</option>
  <option value="parrot">Parrot</option>
  <option value="spider">Spider</option>
  <option value="goldfish">Goldfish</option>
</select>

```

HideMaru for Noun Project

Our HTML

(A fraction of it.)

```
                <section id="app">
                    <select id="currencies" name="currencies">

                    </select>
                </section>
```

JS: a possible answer

First, let’s get a reference to our <select> that already lives in the HTML:

```
const selectNode = document.querySelector('#currencies');
```

Then, we get the symbols array that is inside the jsonData object:

```
const symbols = jsonData.symbols;
```

And now, we cycle through the symbols object, which looks like this:

```
{

     "AED": "United Arab Emirates Dirham",
     "AFN": "Afghan Afghani",
     "ALL": "Albanian Lek",
```

We already know how to cycle through an array. How to cycle through an object? Research it before continuing.

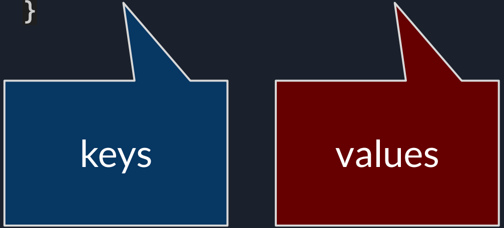
JS: A possible answer. Cycling through objects

There are a few techniques to cycle through objects. I would recommend to use `Object.keys()`.

`Object.keys()` receives an object as parameter, and returns an array of all of the object’s keys.

Remember:

```
{

        "AED": "United Arab Emirates Dirham",
        "AFN": "Afghan Afghani",
        "ALL": "Albanian Lek",
}
```



JS: A possible answer. Cycling through objects

This is how it works:

```
const symbols = jsonData.symbols;
const symbolKeys = Object.keys(symbols); /* ["AED", "AFN", "ALL"...] */
```

And now, we can cycle through the `symbolKeys` array using any method, such as:

```
for (key of symbolKeys) {

   const value = symbols[key]; /* if key is "AED", value is "United Arab Emirates Dirham" */

}
```

Apart from `Object.keys()`, you can also use `Object.values()` (less convenient) and `Object.entries()` (more convenient, but a bit more difficult to understand at first).

There is also the for...in cycle, but has quirks and therefore it's not recommended.

Warning: five slides left!

JS: A possible answer. References to variable keys

Before we continue, you may have tripped over this:

```
const value = symbols[key]; /* if key is "AED", value is "United Arab Emirates Dirham" */
```

You knew that you could do this; this is how objects work:

```
const value = symbols.AED; /* value is "United Arab Emirates Dirham" */
```

But with this notation you have to know the name of a key to get its value.

For cases where you don’t know that name, and it’s in a constant or variable instead, Javascript gives you the `object[variable key]` notation that we saw above.

By the way, even if you know the name, you could do this if you wanted:

```
const value = symbols["AED"]; /* value is "United Arab Emirates Dirham" */
```

Warning: four slides left!

JS: a possible answer

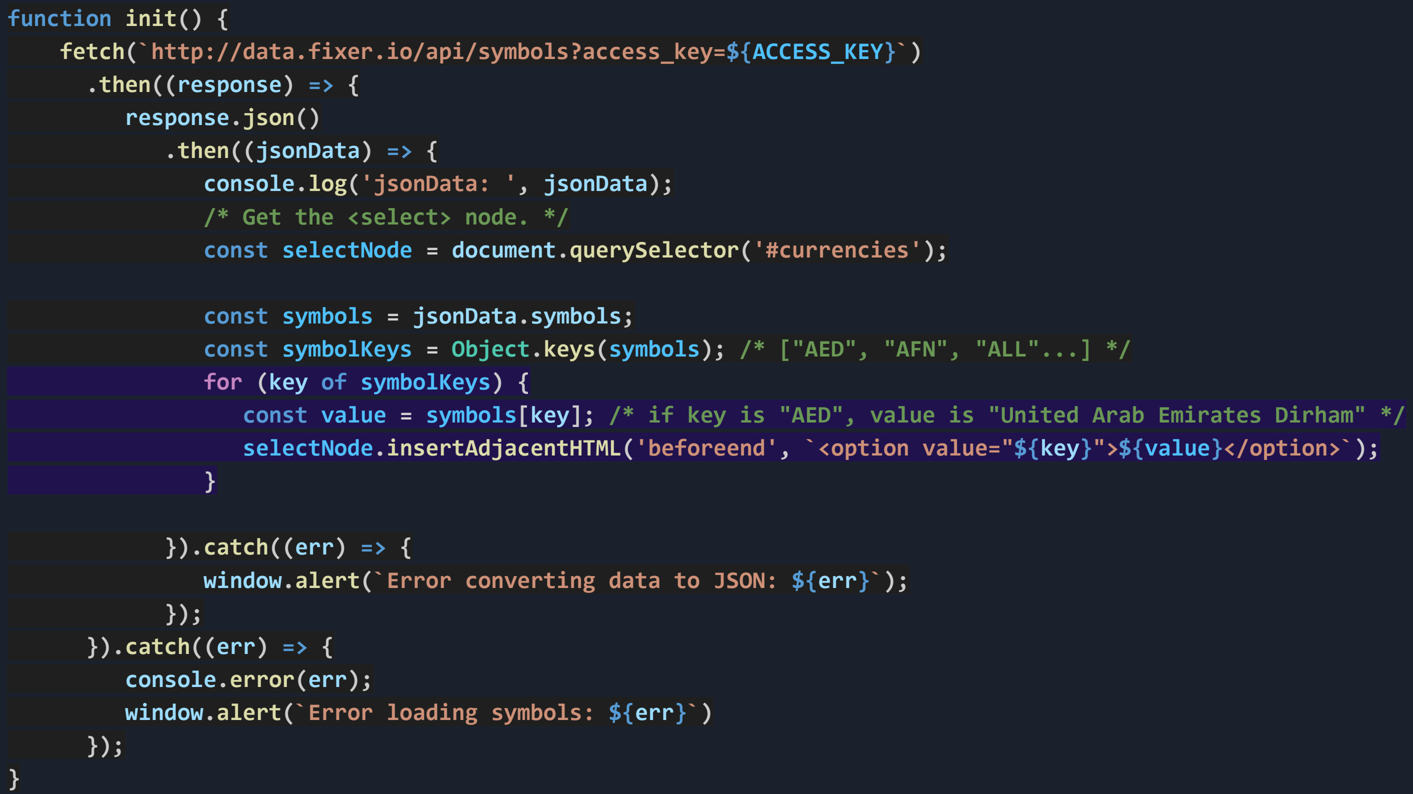
Created by HideMaru
from Noun Project

Now let’s fill the callback function inside of that forEach cycle. What do we want to do with each `symbol`?

Create an <option> tag per symbol inside of the <select>.

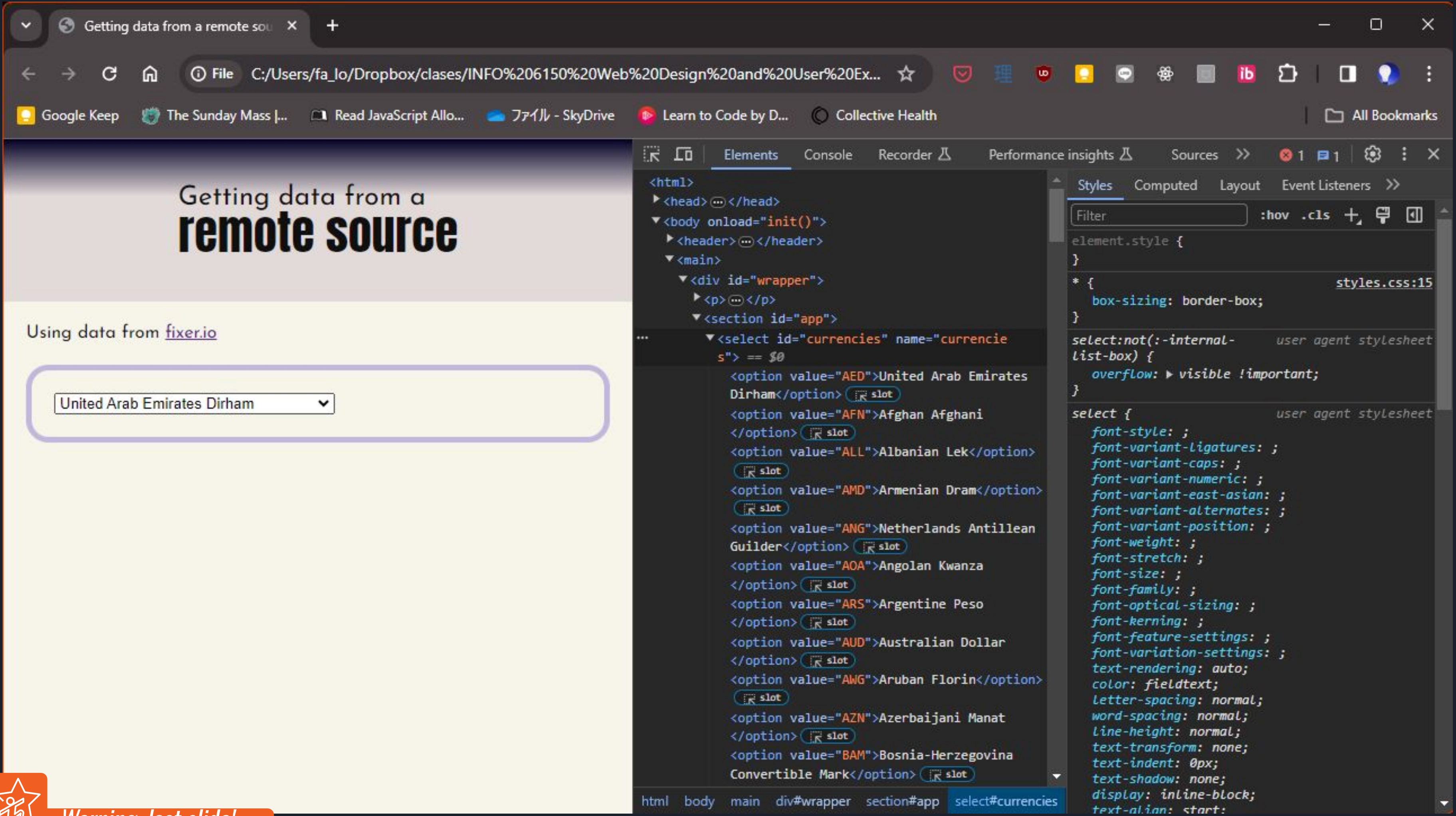
Warning: three slides left!

Our code so far



```
function init() {
    fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`)
      .then((response) => {
         response.json()
            .then((jsonData) => {
               console.log('jsonData: ', jsonData);
               /* Get the <select> node. */
               const selectNode = document.querySelector('#currencies');

               const symbols = jsonData.symbols;
               const symbolKeys = Object.keys(symbols); /* ["AED", "AFN", "ALL"...] */
               for (key of symbolKeys) {
                  const value = symbols[key]; /* if key is "AED", value is "United Arab Emirates Dirham" */
                  selectNode.insertAdjacentHTML('beforeend', `<option value="${key}">${value}</option>`);
               }

            }).catch((err) => {
               window.alert(`Error converting data to JSON: ${err}`);
            });
      }).catch((err) => {
         console.error(err);
         window.alert(`Error loading symbols: ${err}`)
      });
}
```

Warning: two slides left!



Getting data from a

remote source

Using data from fixer.io

United Arab Emirates Dirham

```
<html>
 ▶<head>⋯</head>
 ▼<body onload="init()">
   ▶<header>⋯</header>
   ▼<main>
     ▼<div id="wrapper">
       ▶<p>⋯</p>
       ▼<section id="app">
         ▼<select id="currencies" name="currencie
           s"> == $0
             <option value="AED">United Arab Emirates
             Dirham</option> slot
             <option value="AFN">Afghan Afghani
             </option> slot
             <option value="ALL">Albanian Lek</option>
              slot
             <option value="AMD">Armenian Dram</option>
              slot
             <option value="ANG">Netherlands Antillean
             Guilder</option> slot
             <option value="AOA">Angolan Kwanza
             </option> slot
             <option value="ARS">Argentine Peso
             </option> slot
             <option value="AUD">Australian Dollar
             </option> slot
             <option value="AWG">Aruban Florin</option>
              slot
             <option value="AZN">Azerbaijani Manat
             </option> slot
             <option value="BAM">Bosnia-Herzegovina
             Convertible Mark</option> slot
```

```
element.style {
}
* {                                          styles.css:15
  box-sizing: border-box;
}
select:not(:-internal-          user agent stylesheet
list-box) {
  overflow: ▶ visible !important;
}
select {                        user agent stylesheet
  font-style: ;
  font-variant-ligatures: ;
  font-variant-caps: ;
  font-variant-numeric: ;
  font-variant-east-asian: ;
  font-variant-alternates: ;
  font-variant-position: ;
  font-weight: ;
  font-stretch: ;
  font-size: ;
  font-family: ;
  font-optical-sizing: ;
  font-kerning: ;
  font-feature-settings: ;
  font-variation-settings: ;
  text-rendering: auto;
  color: fieldtext;
  letter-spacing: normal;
  word-spacing: normal;
  line-height: normal;
  text-transform: none;
  text-indent: 0px;
  text-shadow: none;
  display: inline-block;
  text-align: start;
```

Warning: last slide!

The async and await keywords

Now our code is shorter, but our `init()` function became larger and somehow harder to read.

There is another way of writing asynchronous functions, so that they're easier to read.

Whenever a function will (always) return a promise, we can call it with the `await` keyword.

We can also apply the `await` keyword to promises directly.

The async and await keywords

This:

```
const fetchPromise = fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`);
fetchPromise.then((response) => {
   console.log(response);
})
```

can be written as this:

```
const response = await fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`);
console.log(response);
```

It can also be written as this:

```
const fetchPromise = fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`);
const response = await fetchPromise;
console.log(response);
```

The `await` keyword will wait until the promise is resolved and assign the value to `response` when that happens.

The async and await keywords

We use the async function to signal that a function will always return a promise.

If we're writing a function and will use await inside of it, our function has to be marked with async.

So we can do

```
async function init() {
   const response = await fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`);
   const jsonData = await response.json();
```

Error handling

Note: this applies to async/await, but also to most other Javascript code.

With async/await, it became much simpler to consume data from our APIs.

However, we no longer have the convenience of a `.catch(error) {}` function to handle errors.

What we do in this case is the standard Javascript error handling using `try…catch`.

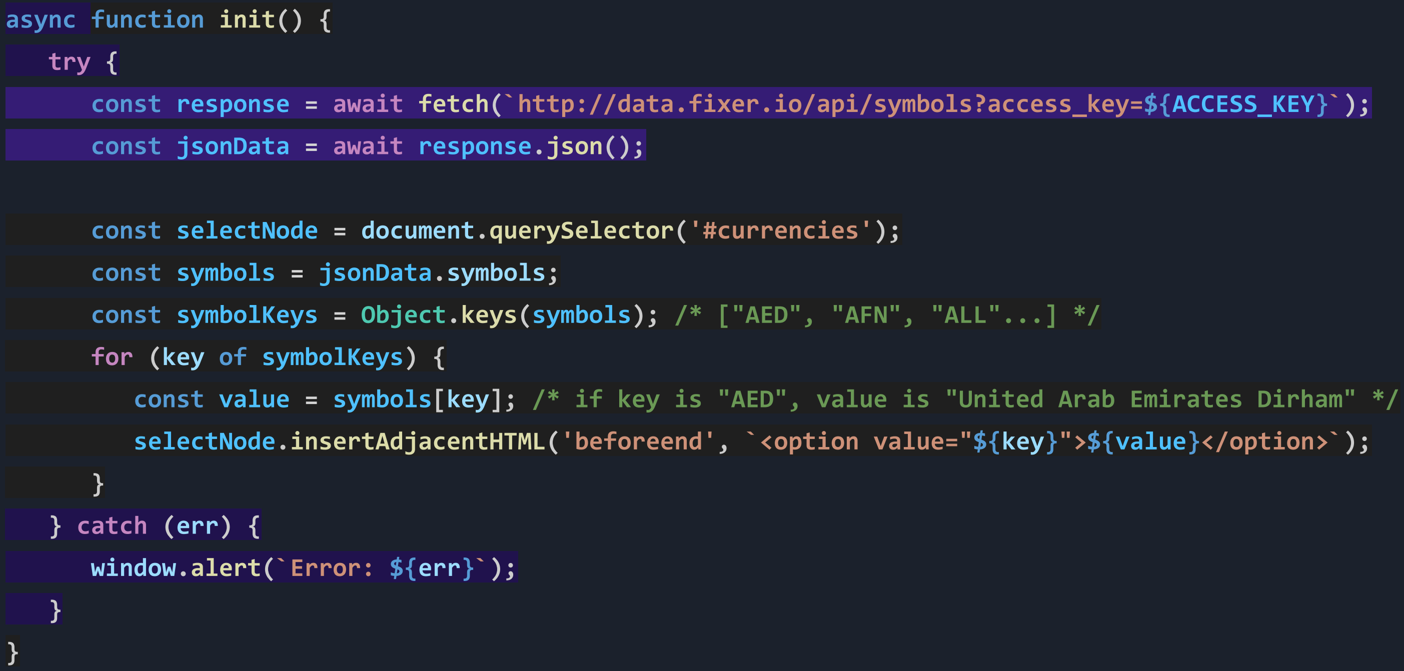
(Note: try…catch exists in Java as well!)

Error handling

Just surround any code that might fail with a `try` block; right after that, write your error handling code inside a `catch` block:

```
try {
   somethingThatFails();
} catch (err) {
   console.log('error: ', err);
}
```

Our code, with async/await and try/catch



```
async function init() {
   try {
      const response = await fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`);
      const jsonData = await response.json();

      const selectNode = document.querySelector('#currencies');
      const symbols = jsonData.symbols;
      const symbolKeys = Object.keys(symbols); /* ["AED", "AFN", "ALL"...] */
      for (key of symbolKeys) {
         const value = symbols[key]; /* if key is "AED", value is "United Arab Emirates Dirham" */
         selectNode.insertAdjacentHTML('beforeend', `<option value="${key}">${value}</option>`);
      }
   } catch (err) {
      window.alert(`Error: ${err}`);
   }
}
```

Async functions return promises

Whenever you mark a function with async, it will always return a promise.

If the function returned anything (such as `async sum(a, b) => { return a + b}`), the value will come wrapped in a promise.

This means you can handle its result with `await` or with `.then()` and `.catch()` - whatever you prefer.

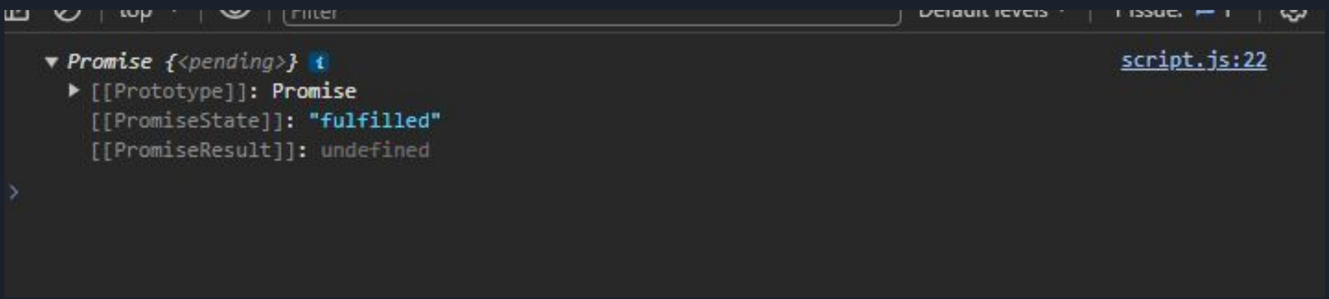
Async functions return promises

An experiment: what does init() return?

```
async function init() {
    try {
        const response = await fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`);
        const jsonData = await response.json();

        const selectNode = document.querySelector('#currencies');
        const symbols = jsonData.symbols;
        const symbolKeys = Object.keys(symbols); /* ["AED", "AFN", "ALL"...] */
        for (key of symbolKeys) {
            const value = symbols[key]; /* if key is "AED", value is "United Arab Emirates Dirham" */
            selectNode.insertAdjacentHTML('beforeend', `<option value="${key}">${value}</option>`);
        }
    } catch (err) {
        window.alert(`Error: ${err}`);
    }
}

const gettingData = init();
console.log(gettingData);
```

Async functions return promises

This is what we got: a promise that, when resolved, returned `undefined` (nothing).



```
▼ Promise {<pending>}                                   script.js:22
  ▶ [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: undefined
>
```

How are we doing?

Requirements: Build a currency conversion app.

- The user should be able to enter a number and select an origin currency and a destination currency (eg. “100 US dollars to Euros”).
- The system will respond with the converted amount, using the most up-to-date market rate for the conversion.

What we have:

- Display available currencies in a select box.

What else to do?

- Display another `<select>` box, for the 2nd currency.
- Show an input for the amount.
- Add a submit button to trigger the transaction.
  - Research which endpoint we can use for that.

Created by HideMaru
from Noun Project

Try it out!

At this point, you have the knowledge needed to finish this assignment. Try it out yourself!

- A complete version of the app will be available in the course materials.
- Feel free to also browse through the CSS used in the app and learn from it.

Tips:

- Make sure to put all inputs and selects inside a <form>
- The free version of this API only supports `http://data.fixer.io/api/latest` using EUR as "base" currency. How can we use this?
  - When clicking the "conversion" button, use the values inside each input and select to create your api call.
- Don't call the `/symbols` API twice. Instead, create both `<select>` tags while you read its results.

Icon by HideMaru for Noun Project

The full app

Our new HTML (only `<main>` is shown). Note the names and ids for selects and inputs.

```
            <div id="wrapper">
                <p>Using data from <a href="https://www.fixer.io">fixer.io</a></p>
                <section id="app">
                    <form id="currencyForm">
                        <div class="row">
                            I want to convert
                            <label for="quantity">(quantity)</label>
                            <input type="number" id="quantity" name="quantity" required />
                        </div>
                        <div class="row">
                            <label for="currencyFrom">(of which currency)</label>
                            <select id="currencyFrom" name="currencyFrom" required></select>
                        </div>

                        <div class="row">
                            <label for="currencyTo">(to which currency)</label>
                            <select id="currencyTo" name="currencyTo" required></select>
                            .
                        </div>
                        <div class="row">
                            <input type="submit" onclick="clickHandler(event)" value="Convert" />
                        </div>
                        <div id="result" class="row result">

                        </div>
                    </form>
                </section>
            </div>
        </main>
```

The full app

Assembling the two select boxes at the same time: We will want to get the nodes first, then insert new HTML into both at the same time.

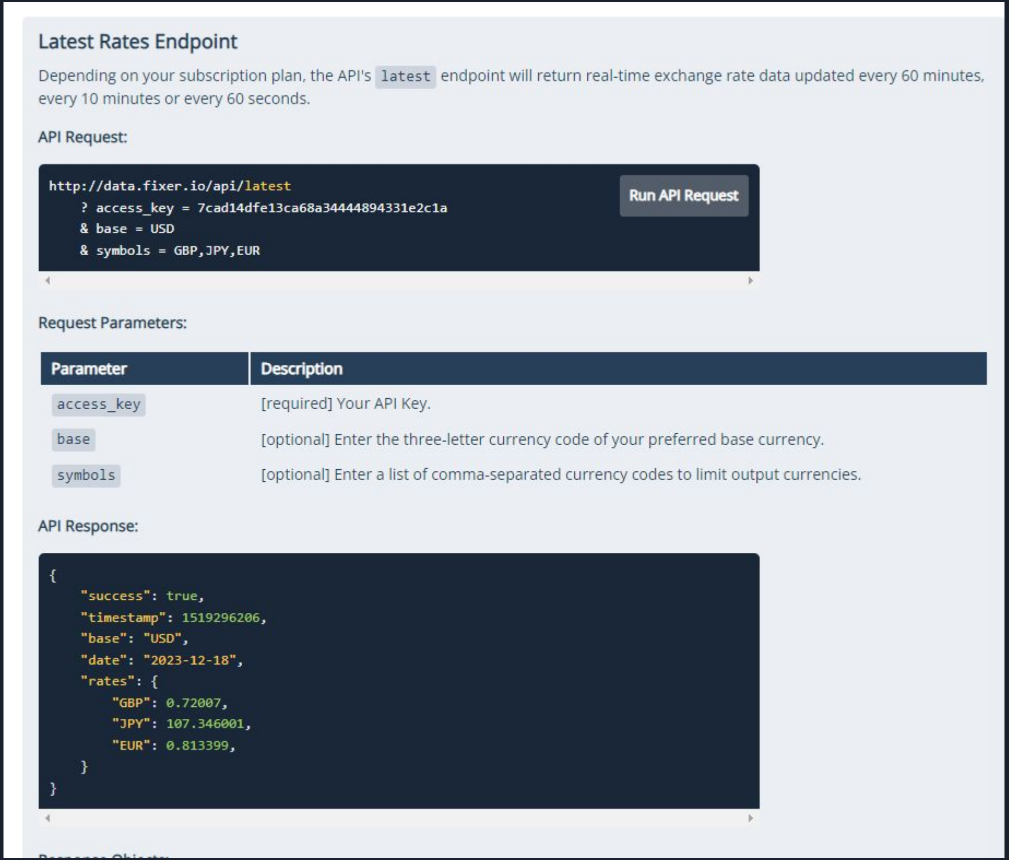
```
async function init() {
    // First, get the references to the nodes we'll use
    const currencyFromNode = document.querySelector('#currencyFrom');
    const currencyToNode = document.querySelector('#currencyTo');

    try {
        const response = await fetch(`http://data.fixer.io/api/symbols?access_key=${ACCESS_KEY}`);
        const jsonData = await response.json();

        const symbols = jsonData.symbols;
        const symbolKeys = Object.keys(symbols);

        for (key of symbolKeys) {
            const value = symbols[key];
            const newOptionHTML = `<option value="${key}">${value}</option>`;
            currencyFromNode.insertAdjacentHTML('beforeend', newOptionHTML);
            currencyToNode.insertAdjacentHTML('beforeend', newOptionHTML);
        }
    } catch (err) {
        window.alert(`Error getting data to build the form: ${err}`);
    }
}
```

The full app

The only API we can use in the free version for our purposes is this one, using `base = EUR`.

We can use the Euro as intermediate between the two currencies:

- First, convert from `currencyFrom` to euros;
- Then, convert from euros to `currencyTo`.



Latest Rates Endpoint

Depending on your subscription plan, the API's `latest` endpoint will return real-time exchange rate data updated every 60 minutes, every 10 minutes or every 60 seconds.

API Request:

```
http://data.fixer.io/api/latest
    ? access_key = 7cad14dfe13ca68a34444894331e2c1a
    & base = USD
    & symbols = GBP,JPY,EUR
```

Request Parameters:

Parameter	Description
`access_key`	[required] Your API Key.
`base`	[optional] Enter the three-letter currency code of your preferred base currency.
`symbols`	[optional] Enter a list of comma-separated currency codes to limit output currencies.

API Response:

```
{
    "success": true,
    "timestamp": 1519296206,
    "base": "USD",
    "date": "2023-12-18",
    "rates": {
        "GBP": 0.72007,
        "JPY": 107.346001,
        "EUR": 0.813399,
    }
}
```

The full app

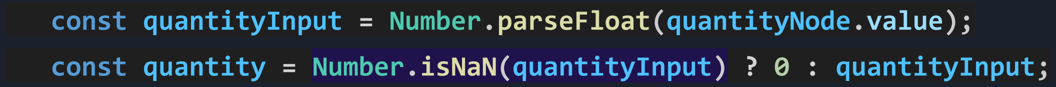
Click handler: Getting values

```javascript
async function clickHandler(event) {
    // prevent the form from reloading the page
    event.preventDefault();

    // Get node references
    const currencyFromNode = document.querySelector('#currencyFrom');
    const currencyToNode = document.querySelector('#currencyTo');
    const quantityNode = document.querySelector('#quantity');
    const resultNode = document.querySelector('#result');

    // Get values from the nodes
    const currencyFrom = currencyFromNode.value;
    const currencyTo = currencyToNode.value;

    // get value from the input; if it's Not a Number (NaN), force it to be 0.
    const quantityInput = Number.parseFloat(quantityNode.value);
    const quantity = Number.isNaN(quantityInput) ? 0 : quantityInput;

    // Compose the URL we will send
    // We can only use this endpoint in the free version, using euros
    const url =
`http://data.fixer.io/api/latest?access_key=${ACCESS_KEY}&base=EUR&symbols=${currencyFrom},${currencyTo}`
```

Aside: Not a Number (NaN)

When reading numbers, always check if the input is a valid number. If not you'll get NaN instead.

Check if a value is NaN using `Number.isNaN()`.



```
    const quantityInput = Number.parseFloat(quantityNode.value);
    const quantity = Number.isNaN(quantityInput) ? 0 : quantityInput;
```

Aside: the ternary operator (? :)

```
const quantity = Number.isNaN(quantityInput) ? 0 : quantityInput;
```

This is the same as saying:

```
   let quantity;
   if (Number.isNaN(quantityInput)) {
      quantity = 0;
   } else {
      quantity = quantityInput;
   }
```

but in a single line, and allows us to declare a `const` instead.

The full app

Loading rates to use in calculation:

```
   // try sending it and getting the result
   try {
      const response = await fetch(url);
      const json = await response.json();

      /* response format is as follows:
         {
            "success": true,
            "timestamp": 1519296206,
            "base": "USD",
            "date": "2023-12-18",
            "rates": {
               "GBP": 0.72007,
               "JPY": 107.346001,
               "EUR": 0.813399,
            }
         }
      */
      const { rates } = json; // this is the same as "const rates = json.rates"

      // get the rates for the currencies we want
      const fromRate = rates[currencyFrom]; // eg. if currencyFrom = GBP, this is rates.GBP
      const toRate = rates[currencyTo];
```

Aside: Destructuring

Use this to more easily get data from within an object.

```
const { rates } = json; // this is the same as "const rates = json.rates"
```

You can destructure many elements at the same time; for example:

```
const { success, timestamp, base, rates } = json;
```

would create one constant for each one of those elements in the object.

If the element does not exist, you will get a value of `undefined`.

The full app

Using the rates from the API:

```
      // get the rates for the currencies we want
      const fromRate = rates[currencyFrom]; // eg. if currencyFrom = GBP, this is rates.GBP
      const toRate = rates[currencyTo];

      // convert to euros first
      const valueInEuros = quantity / fromRate;

      // then convert to the target currency
      const result = valueInEuros * toRate;

      console.log({ fromRate, toRate })
      console.log({ quantity, valueInEuros, result })

      // output the value to the HTML
      resultNode.innerHTML = result;

   } catch (err) {
      window.alert(`Error converting: ${err}`);
   }
```

Aside: innerHTML

In certain cases you can substitute what’s inside of an element just by doing this:

```
    // output the value to the HTML
    resultNode.innerHTML = result;
```

`innerHTML` represents the full content of an HTML node.

Do not do this if the node contains other nodes with event handlers attached: this would lead to a memory leak (eg. there’s memory assigned to the program but that the program can no longer use).

Some important concepts about REST

Idempotence: Making the same call multiple times should have the same effect on the server.

For example, GET /index.html should always return the same page.

Some important concepts about REST

Cacheable: One of the principles of REST.

REST responses can either be cacheable or non-cacheable. “Cacheable” means that the response can be reused for a limited time. To do this, the client can save the response into a “cache” (temporary memory).

Some important concepts about REST

Stateless: Another principle of REST.

When we send a request to the server, the request contains *all* of the necessary information to understand it and send the response.

This means that, for example, if the application wants to have session information (eg. information about the current user and what they’re doing), it must be completely handled by the app. In principle, a session cannot be saved to the server.

This is why, for example, session information is sent *with each call* through, for example, cookies.