D1 - Pool

D-POO-100

JS pool - day 11

API

0.3.2

JS pool - day 11

delivery method: Github
language: Javascript

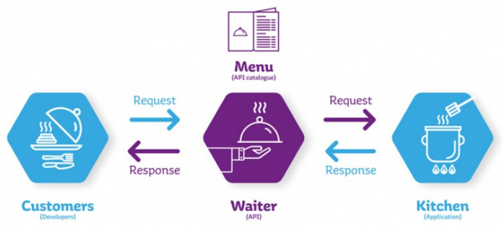
- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.

An **A**pplication **P**rograming **I**nterface defines interactions between programs.
A *consumer* program can make use of the feature of a *vendor* program.
Web APIs are legion. Mastering them is mandatory.

To discern it, the restaurant analogy helps:

- the consumer, a Front Client, wants to eat a dish ;
- the kitchen is a Back-End Server, can cook this very dish ;
- the waiter is the API, which communicates with both the consumer and the kitchen ;
- a menu defines which dish the consumer can order to the waiter, i.e. the language of the API.

The consumer should not know what's going on in the kitchen.
Sometimes, consumers order dishes that are not on the menu!

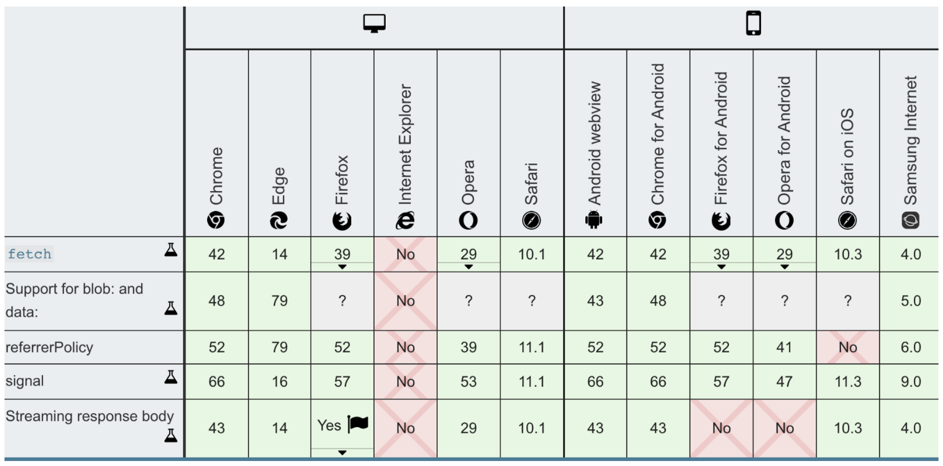


APIs are therefore an essential tool in the digital world.
They act as transmission belts between code chunks and are an huge sources of data.

JavaScript can send network requests to the server and load new information whenever it's needed.
For example, we can use a network request to:

- submit an order,
- load user information,
- receive latest updates from the server,
- ...etc.

...and all of that without reloading the page!
The `fetch()` method is modern and versatile way to send a network request and get information from the server. It is not supported by old browsers:



	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Android webview	Chrome for Android	Firefox for Android	Opera for Android	Safari on iOS	Samsung Internet
`fetch`	42	14	39	No	29	10.1	42	42	39	29	10.3	4.0
Support for blob: and data:	48	79	?	No	?	?	43	48	?	?	?	5.0
referrerPolicy	52	79	52	No	39	11.1	52	52	52	41	No	6.0
signal	66	16	57	No	53	11.1	66	66	57	47	11.3	9.0
Streaming response body	43	14	Yes	No	29	10.1	43	43	No	No	10.3	4.0

When `fetch` is called, the browser starts the request right away and returns a promise that the calling code should use to get the result. Getting a response is usually a two-stage process.

First, the `promise`, returned by `fetch`, resolves with an object of the built-in Response class as soon as the server responds with headers.
At this stage we can check HTTP status, to see whether it is successful or not, check headers, but don't have the body yet.

Second, to get the response body, we need to use an additional method call.
`Response` provides multiple promise-based methods to access the body in various formats.

EXERCISE 00

File to turn in: ex_00/ex_00.js
Function prototype: *my_fetch(url: string)*

Write a `my_fetch` function, using the `fetch` method to get the content of this URL and display the response in JSON object format into the console.

Do you remember that`fetch` returns a *promise*?
A *promise*, indeed.

EXERCISE 00BIS

File to turn in: ex_00bis/ex_00bis.js
Function prototype: *check_route(url: string)*

Create a `check_route` function that fetches an url (provided as parameter) and prints in the console:

- `all is good` if the response status code is a success ;
- `shit happens` if any error occurs (and handles it).

Here is a list of APIs, which routes you can use to test your function:
coinbase API, dogs images API, popular memes API, jokes API.

From now on, you must systematically handle errors.

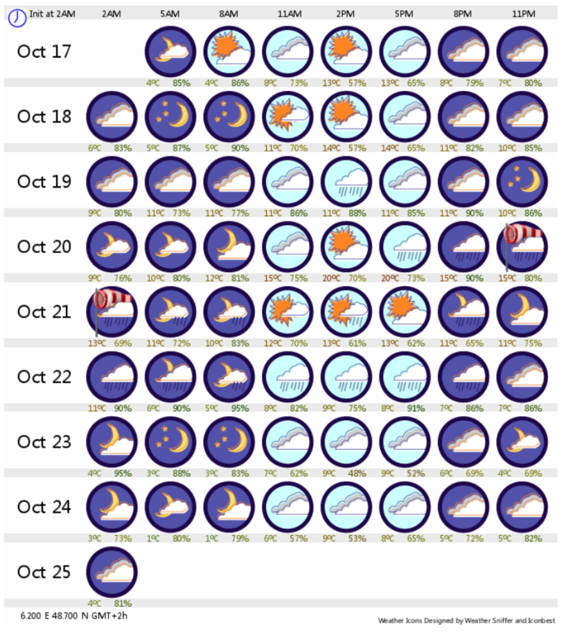
Exercise 01

File to turn in: ex_01/ex_01.js
Function prototype: *display_weather_chart(longitude: float, latitude: float)*

Here is a nice weather forecast API.
Create a `display_weather_chart` function that takes two arguments, respectively a longitude and a latitude.
It must display a chart representing the weather of the next 7 days in the given place.
You must use this route and read the **wiki** available on their website.

Look online to see what query parameters are

Additionally, you could give the following query parameter to the API:
unit=metric, output=internal, tzshift=0



Exercise 02

File to turn in: ex_02/ex_02.js
ex_02/index.html
ex_02/any css file you fancy

Download the global temperature since 1880 dataset from the Nasa website.
On a button click, fetch and display the content of this file in a paragraph element.

Exercise 02bis

File to turn in: ex_02bis/ex_02bis.js
ex_02bis/index.html
ex_02bis/any css file you fancy

Modify your previous code, so that it only outputs **one row by line**, parsing it to keep only the year column and the January temperatures.

Exercise 02ter (Optional)

File to turn in: ex_02ter/ex_02ter.js
ex_02ter/index.html
ex_02ter/any css file you fancy

From your previous code, draw a nice chart representing the temperature evolution since 1880, for January.

Take a look at the ChartJs library to help you out.

Exercise 03

File to turn in: ex_03/ex_03.js
ex_03/ex_03.js
ex_03/any css file you fancy
Function prototype: display_planet_info(planet: object)

Create a function `display_planet_info` that accept as parameter a planet object and for each call append in a `<ul id="planet-list">` element a new `<li class="planet">` element.

In that new element, for the given planet, display the way you fancy :

- the name
- the diameter
- the climate
- the terrain
- the population.

The description of the planet JSON object can be found in this swapi API.
You can test your function by using fetch but this is not mandatory.

We strongly encourage you to use librairies like jquery or vue.js to help you out.

Exercise 03bis

File to turn in: ex_03bis/ex_03bis.js
ex_03bis/ex_03bis.js
ex_03bis/any css file you fancy

Using the same API and the function you've created, populate the page by the ten first planets. Implement a pagination system by adding a next and a previous button. On click, the 10 planets displayed must change accordingly.

Also, create an **input element**. By default, the input value must be empty.
Only planets which contain the input value in their name should be displayed.

You must grey out the buttons if no previous or next is to be found

Exercise 03ter

File to turn in: ex_03ter/ex_03ter.js
ex_03ter/ex_03ter.js
ex_03ter/any css file you fancy

Make the planets clickable. On click, the list of planets must be replaced by a full report of this planet. This report must also include:

- A list of the film names that this planet has appeared in
- A list of the residents names that live on this planet

Exercise 04

File to turn in: ex_04/ex_04.js
ex_04/ex_04.js
ex_04/any css file you fancy

Find 5 free APIs of your choice, call at least one of their routes and display the result the way you like on a web page.
Be creative, use css and animations to make the best (or the worst) webpage that history has never known.

Each fetched element on the page must appear independently of one another.