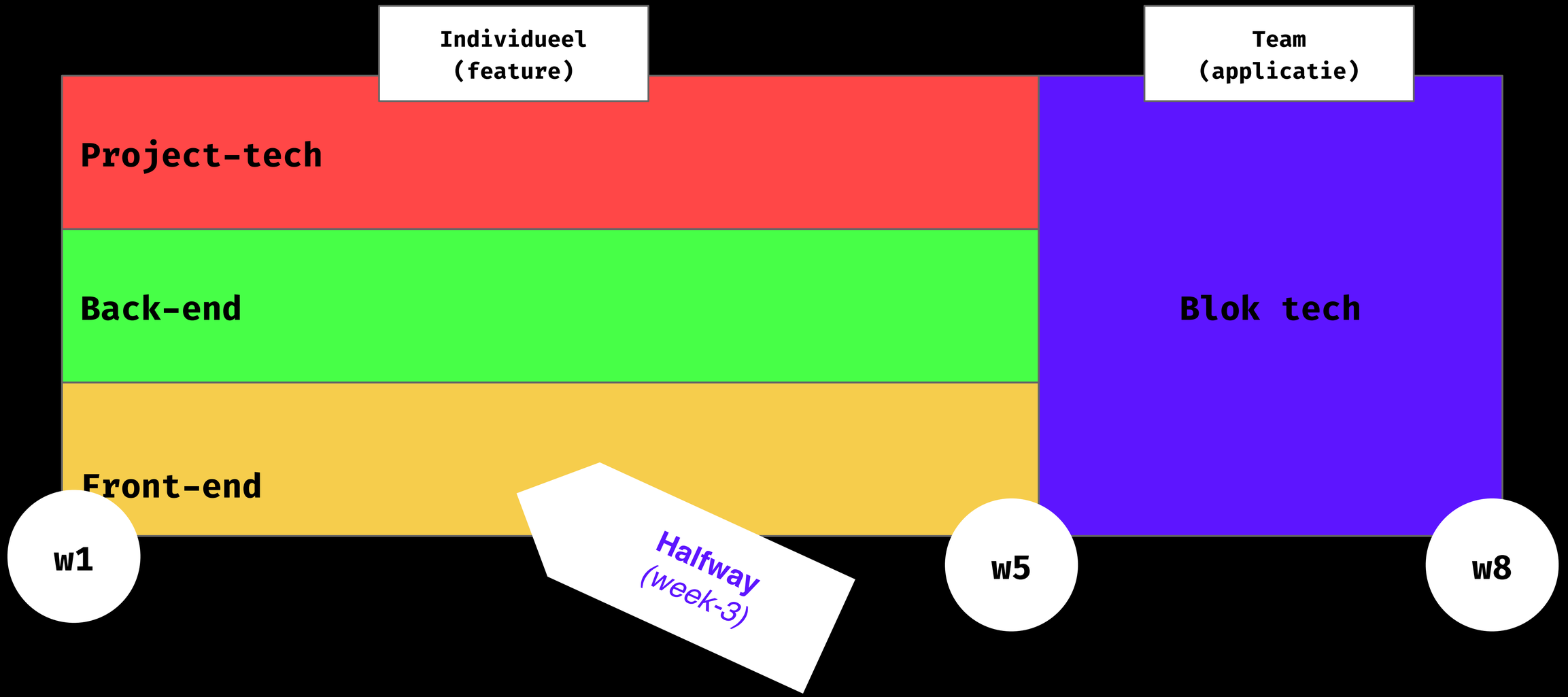
front-end

Async & Data Fetching

lab 3/8


Individueel
(feature)
Team
(applicatie)
Project-tech
Back-end
Front-end
Blok tech
w1
Halfway
(week-3)
w5
w8

Stand-up!

Progressive enhancement

today

I. ~~Stand-up~~

II. JavaScript concepts (recap)

III. Sync versus Async

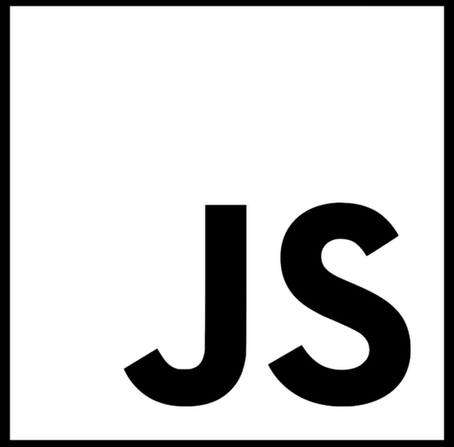
IV. Fetching API’s

JavaScript concepts

Recap

ES5

< 2015

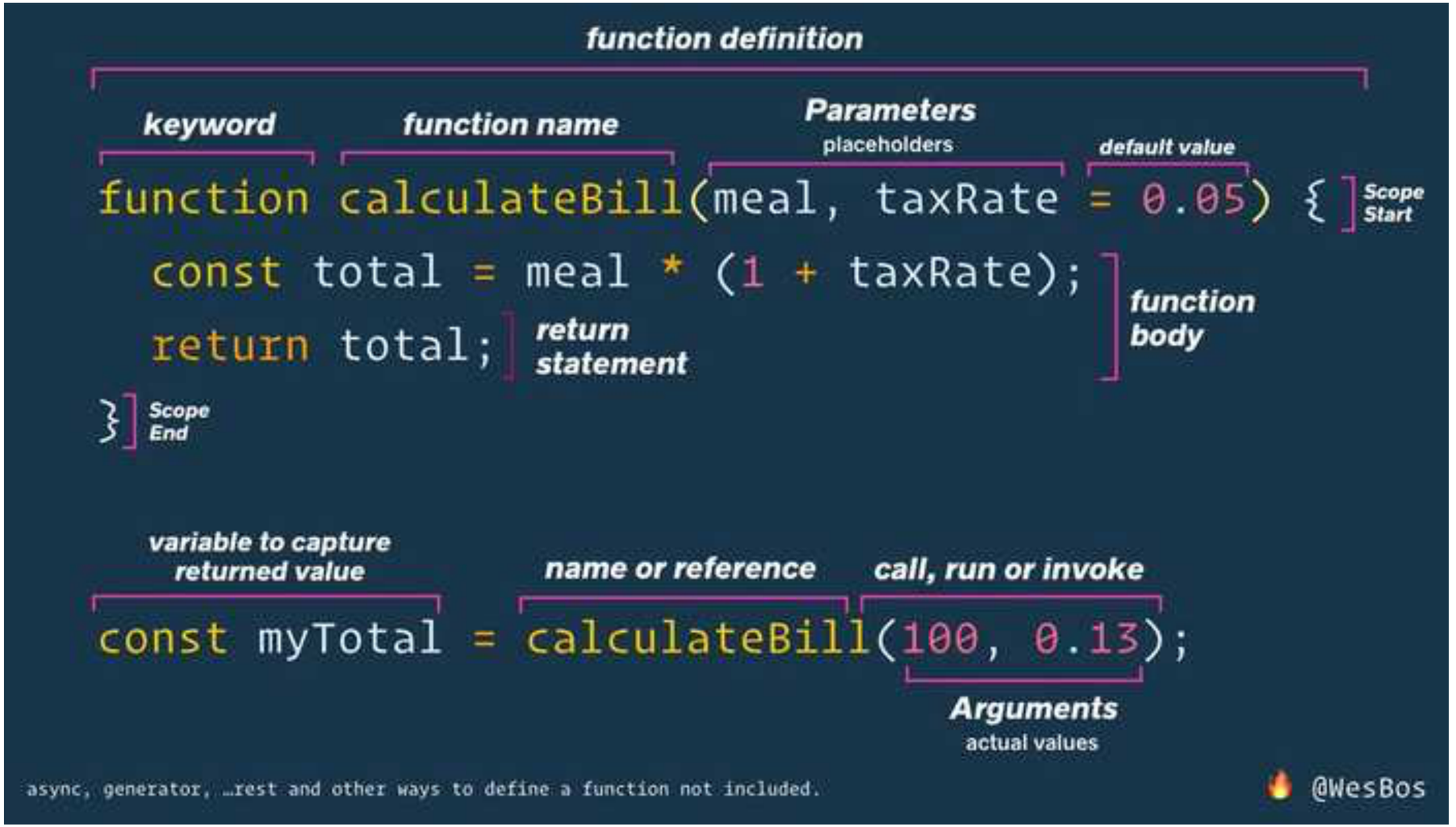


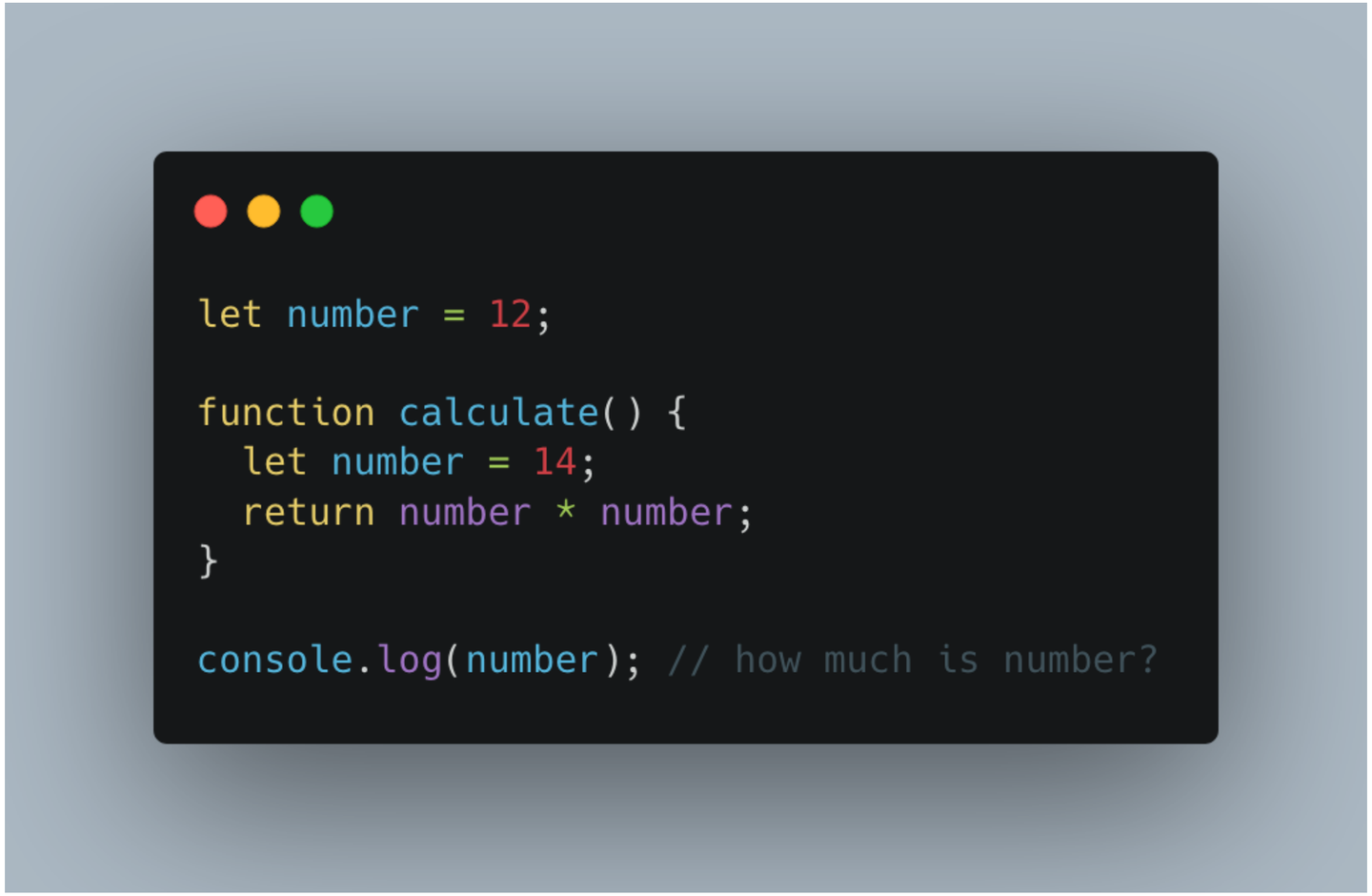
ES6

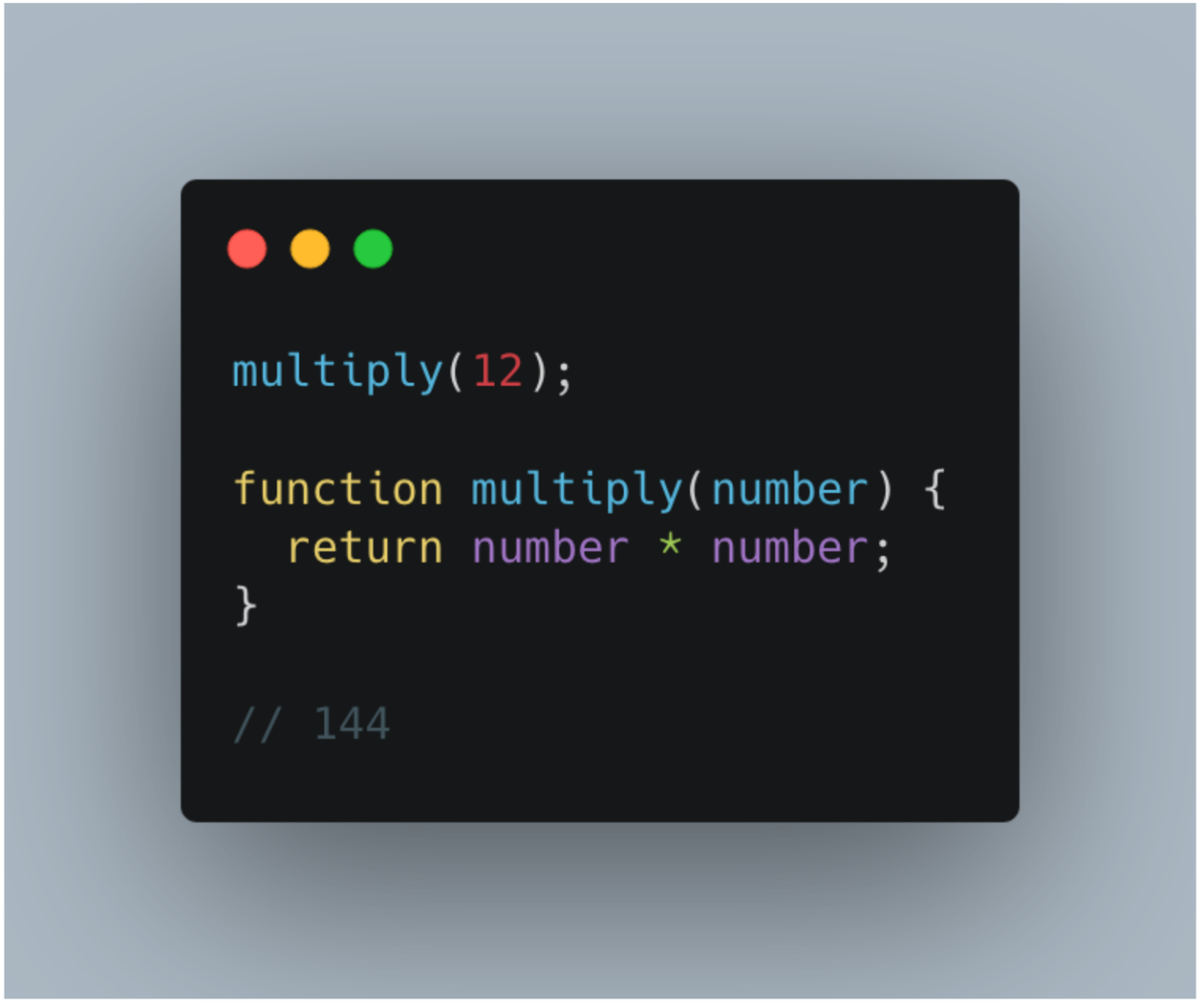
2015 >

```
const teacher = {
  name: "Robert",
  age: 29,
  location: ["52.359135", "4.909953"],
  hasCar: false,
  sayHi: function() { // We use function() instead of => to preserve context!
    console.log(this) // "this" now points to this object as a keyword!
    console.log(`Hi ${this.name}!`)
  }
}

console.log(teacher.sayHi()) // Yields "Hi Robert!"
```


function definition
keyword
function name
Parameters
placeholders
default value
function calculateBill(meal, taxRate = 0.05) {
Scope Start
const total = meal * (1 + taxRate);
return total;
return statement
function body
}
Scope End
variable to capture returned value
name or reference
call, run or invoke
const myTotal = calculateBill(100, 0.13);
Arguments
actual values
async, generator, …rest and other ways to define a function not included.
@WesBos


let number = 12;
function calculate() {
let number = 14;
return number * number;
}
console.log(number); // how much is number?


multiply(12);
function multiply(number) {
return number * number;
}
// 144

Events

Reference

Table of contents

Related Topics

- Introduction to events
- Creating and triggering events
- Detecting device orientation
- Event handling (overview)
- Orientation and motion data explained
- Using device orientation with 3D transforms

Event reference

Events are fired to notify code of "interesting changes" that may affect code execution. These can arise from user interactions such as using a mouse or resizing a window, changes in the state of the underlying environment (e.g. low battery or media events from the operating system), and other causes.

Each event is represented by an object that is based on the `Event` interface, and may have additional custom fields and/or functions to provide information about what happened. The documentation for every event has a table (near the top) that includes a link to the associated event interface, and other relevant information. A full list of the different event types is given in Event > Interfaces based on Event.

This topic provides an index to the main *sorts* of events you might be interested in (animation, clipboard, workers etc.) along with the main classes that implement those sorts of events. At the end is a flat list of all documented events.

> **Note:** This page lists many of the most common events you'll come across on the web. If you are searching for an event that isn't listed here, try searching for its name, topic area, or associated specification on the rest of MDN.

Event index

Event type	Description	Documentation
Animation	Events related to the Web Animation API. Used to respond to changes in animation status (e.g. when an animation starts or ends).	Animation events fired on `Document`, `Window`, `HTMLElement`.
Asynchronous data	Events related to the fetching data.	Events fired on `AbortSignal`, `XML-`

Web Api

Example

Table of contents

Related Topics

Intersection Observer API

The Intersection Observer API provides a way to asynchronously observe changes in the intersection of a target element with an ancestor element or with a top-level document's viewport.

Historically, detecting visibility of an element, or the relative visibility of two elements in relation to each other, has been a difficult task for which solutions have been unreliable and prone to causing the browser and the sites the user is accessing to become sluggish. As the web has matured, the need for this kind of information has grown. Intersection information is needed for many reasons, such as:

- Lazy-loading of images or other content as a page is scrolled.
- Implementing "infinite scrolling" web sites, where more and more content is loaded and rendered as you scroll, so that the user doesn't have to flip through pages.
- Reporting of visibility of advertisements in order to calculate ad revenues.
- Deciding whether or not to perform tasks or animation processes based on whether or not the user will see the result.

Implementing intersection detection in the past involved event handlers and loops calling methods like `Element.getBoundingClientRect()` to build up the needed information for every element affected. Since all this code runs on the main thread, even one of these can cause performance problems. When a site is loaded with these tests, things can get downright ugly.

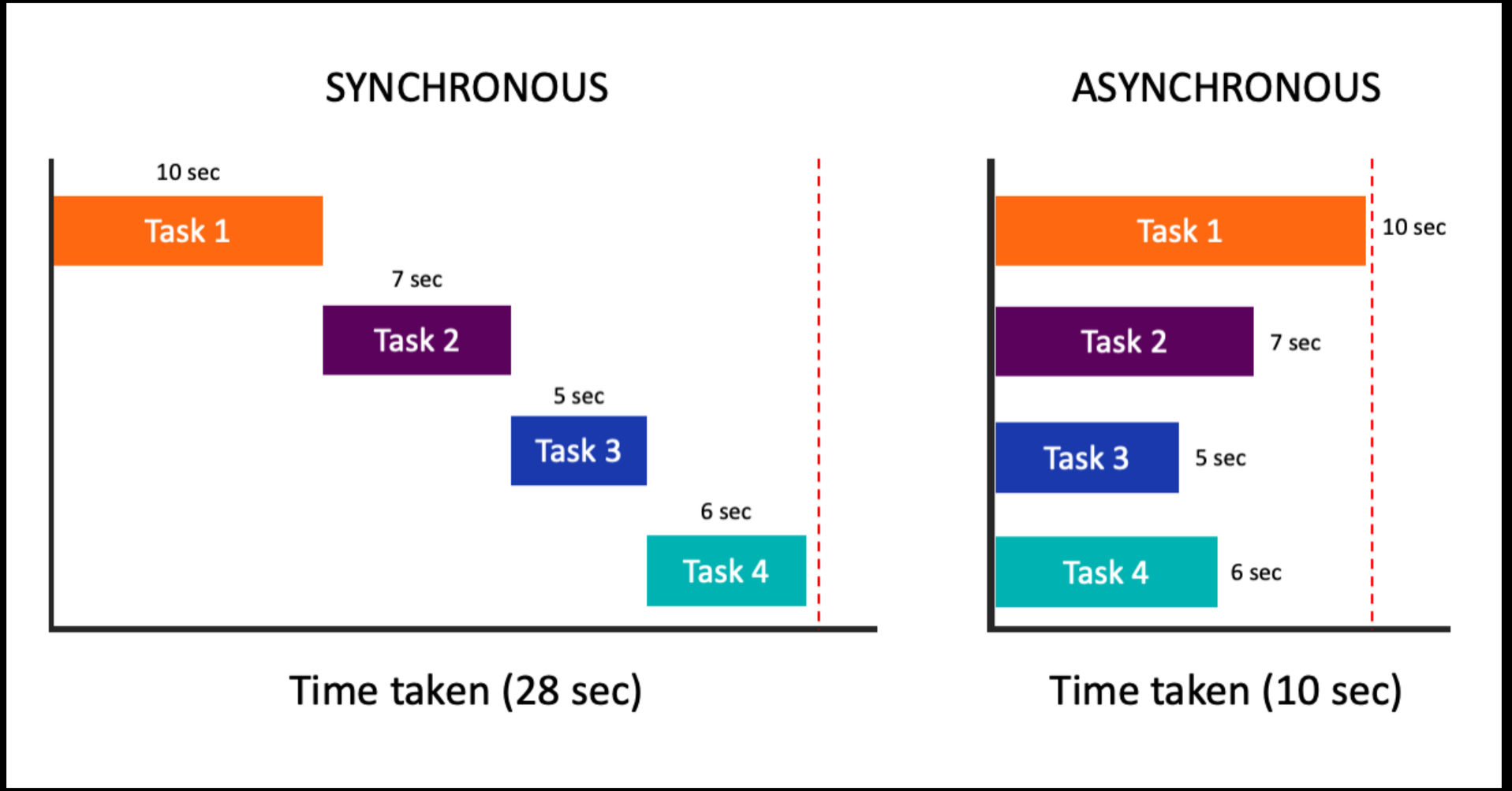
Consider a web page that uses infinite scrolling. It uses a vendor-provided library to manage the advertisements placed periodically throughout the page, has animated graphics here and there, and uses a custom library that draws notification boxes and the like. Each of these has its own intersection detection routines, all running on the main thread. The author of the web site may not even realize this is happening, since they may know very little about the inner workings of the two libraries they are using. As the user scrolls the page, these intersection detection routines are firing constantly during the scroll handling code, resulting in an experience that leaves the user frustrated with the browser, the web site, and their computer.

Onderwerp wat je niet snapt.

Onderwerp wat je begrijpt.

Short Break!

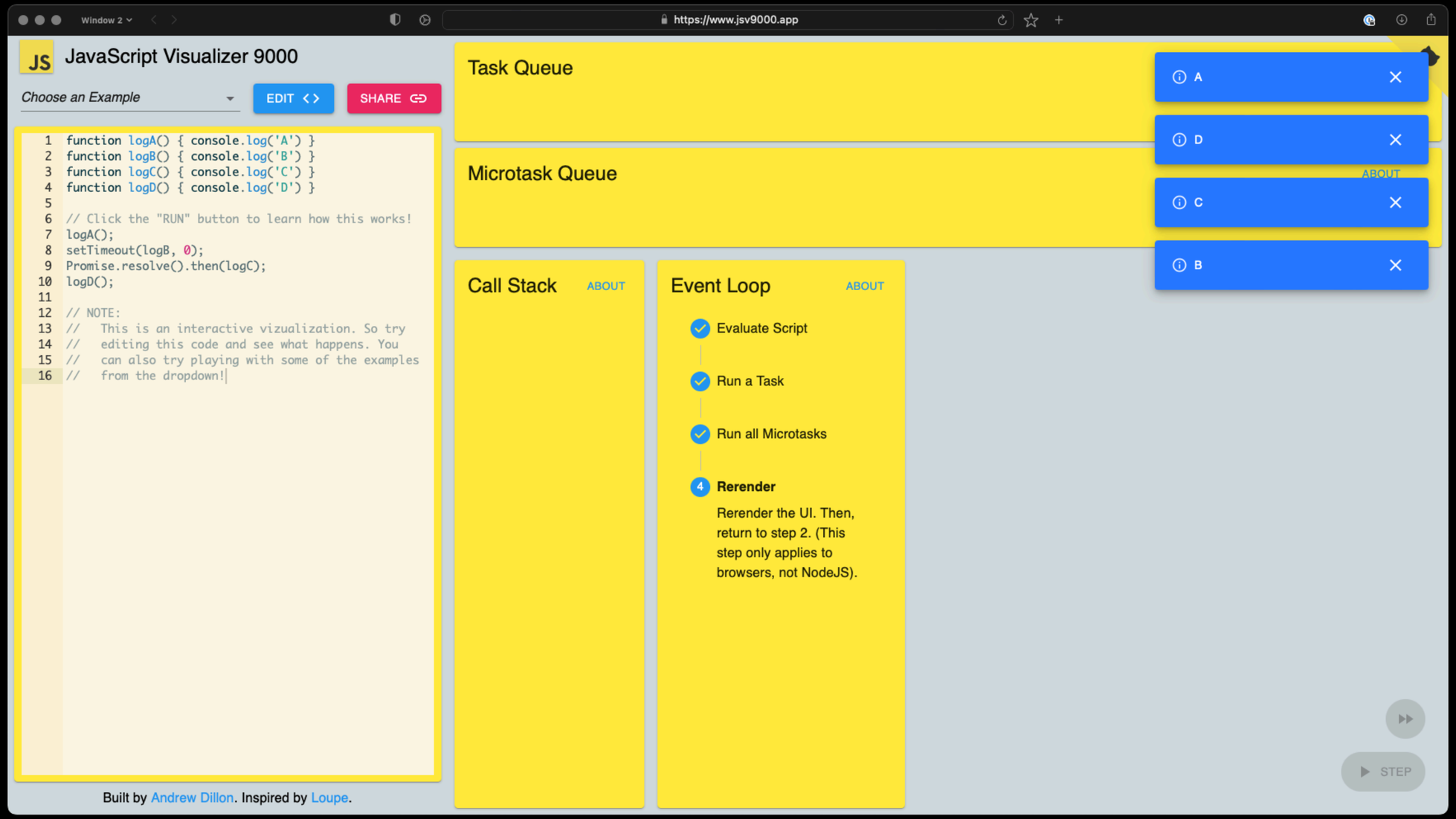
Sync vs Async

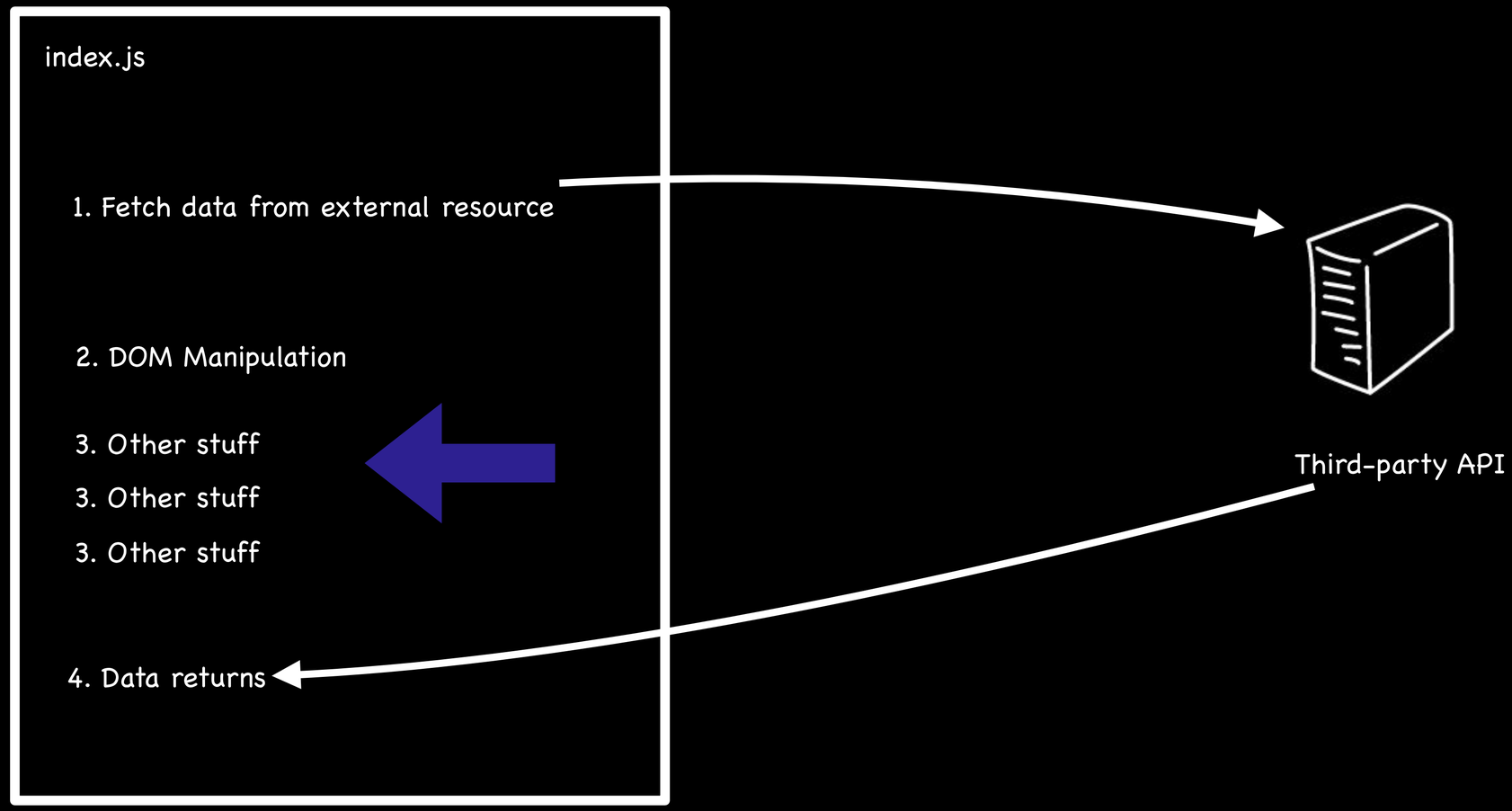

SYNCHRONOUS
10 sec
Task 1
7 sec
Task 2
5 sec
Task 3
6 sec
Task 4
Time taken (28 sec)
ASYNCHRONOUS
Task 1
10 sec
Task 2
7 sec
Task 3
5 sec
Task 4
6 sec
Time taken (10 sec)

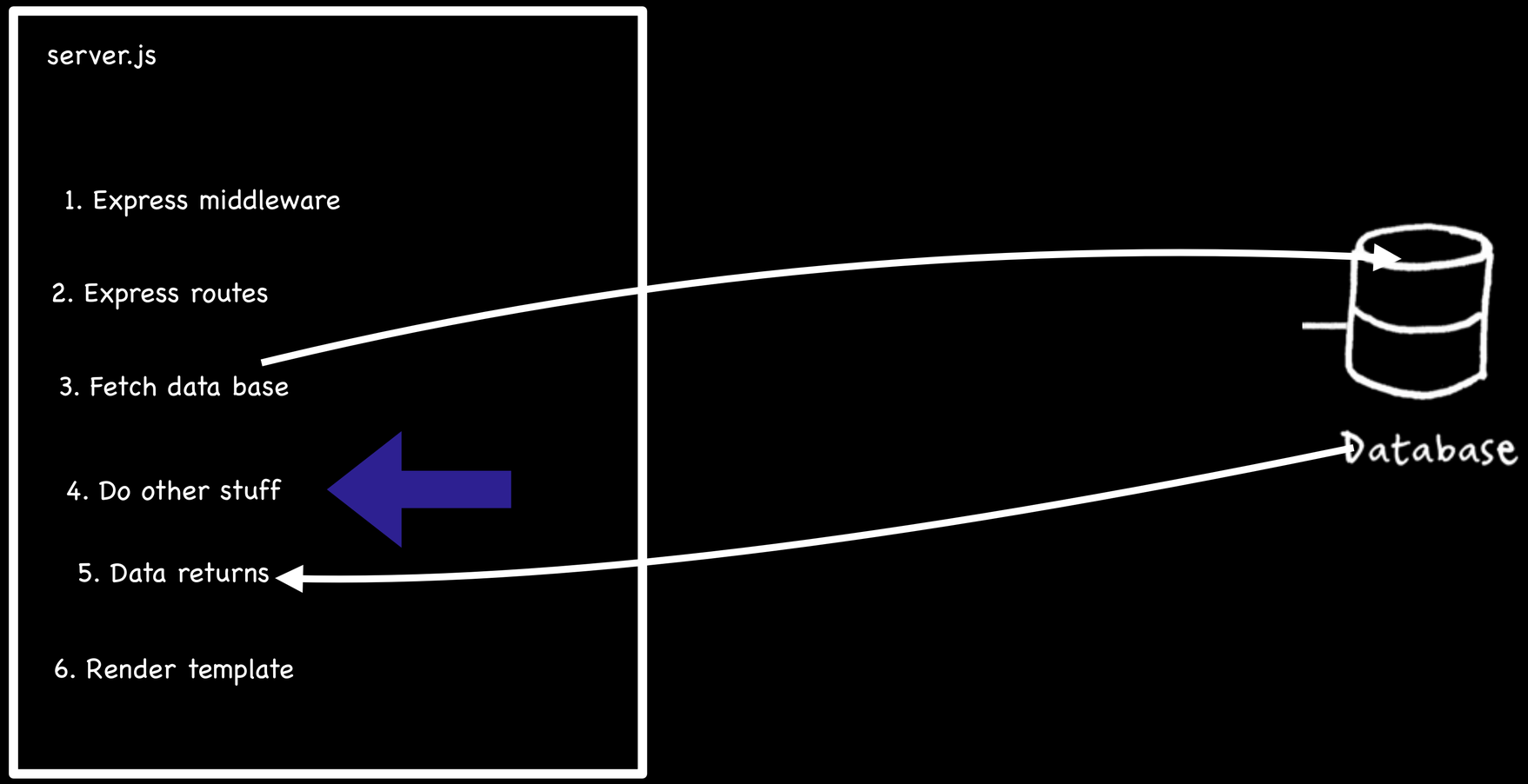
synchronous, single thread of control

synchronous, two threads of control

asynchronous


Window 2
https://www.jsv9000.app
JS
JavaScript Visualizer 9000
Choose an Example
EDIT
SHARE
1 function logA() { console.log('A') }
2 function logB() { console.log('B') }
3 function logC() { console.log('C') }
4 function logD() { console.log('D') }
5
6 // Click the "RUN" button to learn how this works!
7 logA();
8 setTimeout(logB, 0);
9 Promise.resolve().then(logC);
10 logD();
11
12 // NOTE:
13 //   This is an interactive vizualization. So try
14 //   editing this code and see what happens. You
15 //   can also try playing with some of the examples
16 //   from the dropdown!
Built by Andrew Dillon. Inspired by Loupe.
Task Queue
Microtask Queue
ABOUT
Call Stack
ABOUT
Event Loop
ABOUT
Evaluate Script
Run a Task
Run all Microtasks
4
Rerender
Rerender the UI. Then, return to step 2. (This step only applies to browsers, not NodeJS).
A
D
C
B
STEP


index.js
1. Fetch data from external resource
2. DOM Manipulation
3. Other stuff
3. Other stuff
3. Other stuff
4. Data returns
Third-party API


server.js
1. Express middleware
2. Express routes
3. Fetch data base
4. Do other stuff
5. Data returns
6. Render template
Database

Sync vs async

Definition

Sync: each operation must wait for the previous one to complete before executing.

Async: an operation can occur while another one is still being processed.

Sync vs Async

Methods

1. **Callbacks:** Functions as arguments.
2. **Promises:** Returns an object with status.
3. **Async / Await:** Syntactic sugar on top of promises.

Callbacks

What ?

```
function greeting(name) {
  alert('Hello ' + name);
}

function processUserInput(callback) {
  var name = prompt('Please enter your name.');
  callback(name);
}

processUserInput(greeting);
```

Callbacks

What ?

```
function processUserInput(callback) {
  var name = prompt('Please enter your name.');
  callback(name);
}

processUserInput(greeting, name => {
  alert('Hello ' + name);
});
```

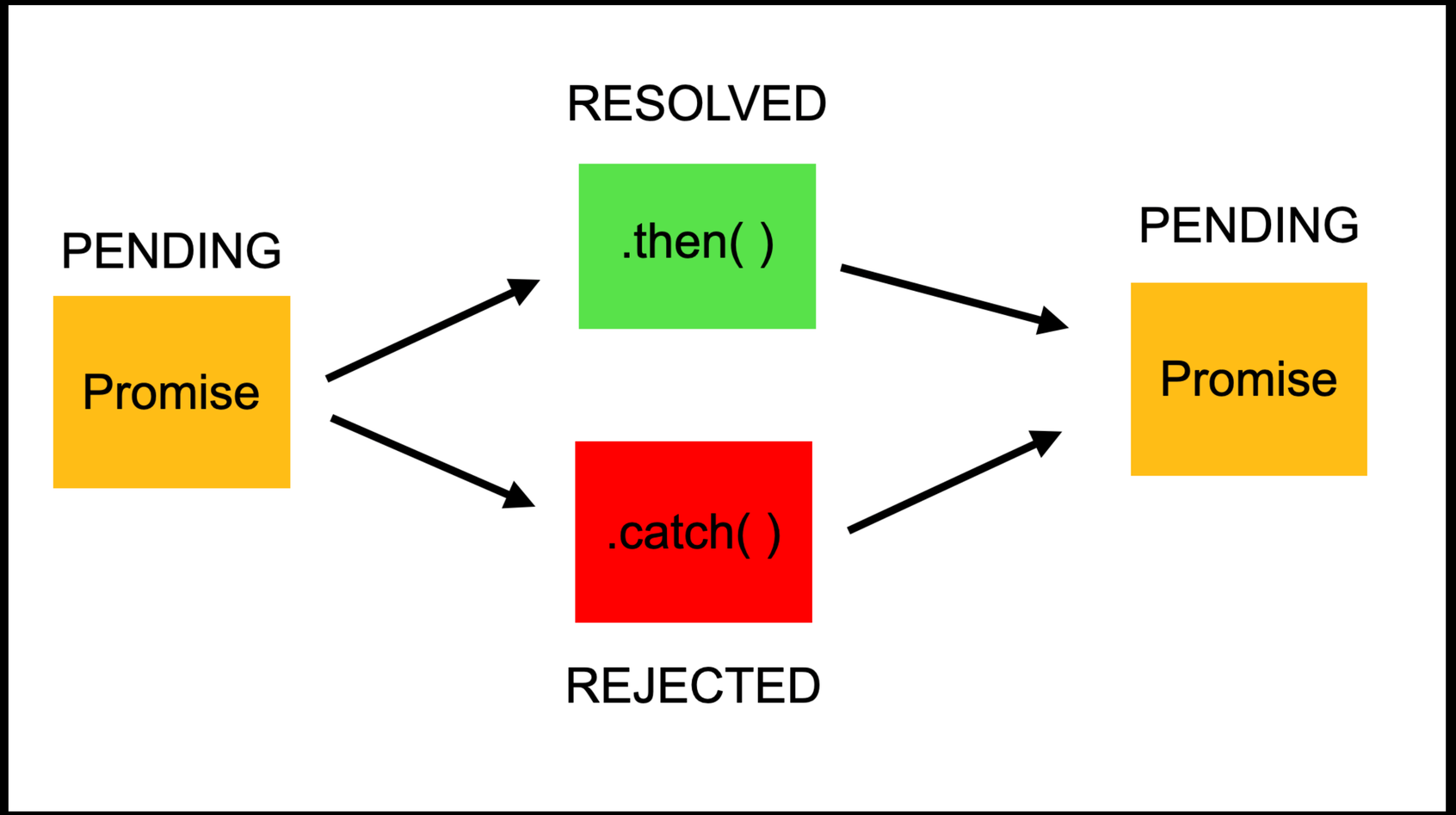
Promises

What ?

The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

Promises

What ?



Promises

What ?

```
const aPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('foo');
  }, 300);
});

aPromise
  .then(handleResolvedA, handleRejectedA)
  .then(handleResolvedB, handleRejectedB)
  .then(handleResolvedC, handleRejectedC);
```

Async & Await What ?

An async function is a function declared with the **async** keyword, and the **async** keyword is permitted within it. The **async** and **await** keywords enable asynchronous, promise-based behavior to be written in a cleaner style, avoiding the need to explicitly configure promise chains.

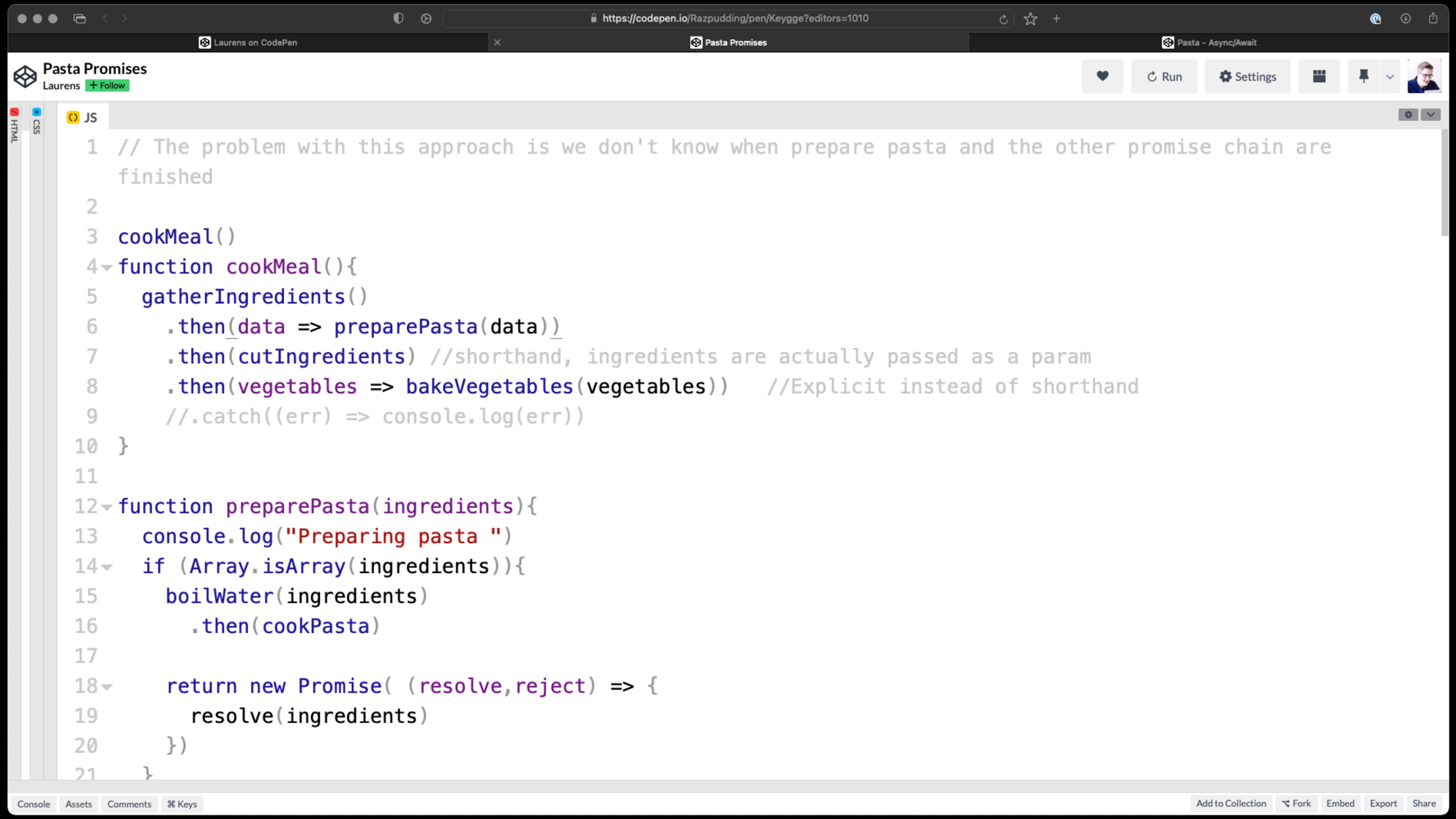
Async & Await

What ?

```
async function getData() {
  const res = await fetch('https://robertspier.nl/v1/api');
  return await res.json();
}

const theData = getData();

// Returns the result from the API url
```


https://codepen.io/Razpudding/pen/Keygge?editors=1010
Laurens on CodePen
Pasta Promises
Pasta - Async/Await
Pasta Promises
Laurens
+ Follow
Run
Settings
HTML
CSS
JS
// The problem with this approach is we don't know when prepare pasta and the other promise chain are finished
cookMeal()
function cookMeal(){
gatherIngredients()
.then(data => preparePasta(data))
.then(cutIngredients) //shorthand, ingredients are actually passed as a param
.then(vegetables => bakeVegetables(vegetables)) //Explicit instead of shorthand
//.catch((err) => console.log(err))
}
function preparePasta(ingredients){
console.log("Preparing pasta ")
if (Array.isArray(ingredients)){
boilWater(ingredients)
.then(cookPasta)
return new Promise((resolve,reject) => {
resolve(ingredients)
})
Console
Assets
Comments
⌘ Keys
Add to Collection
Fork
Embed
Export
Share

Short Break!

Third-party API's

Web Api’s

Two types

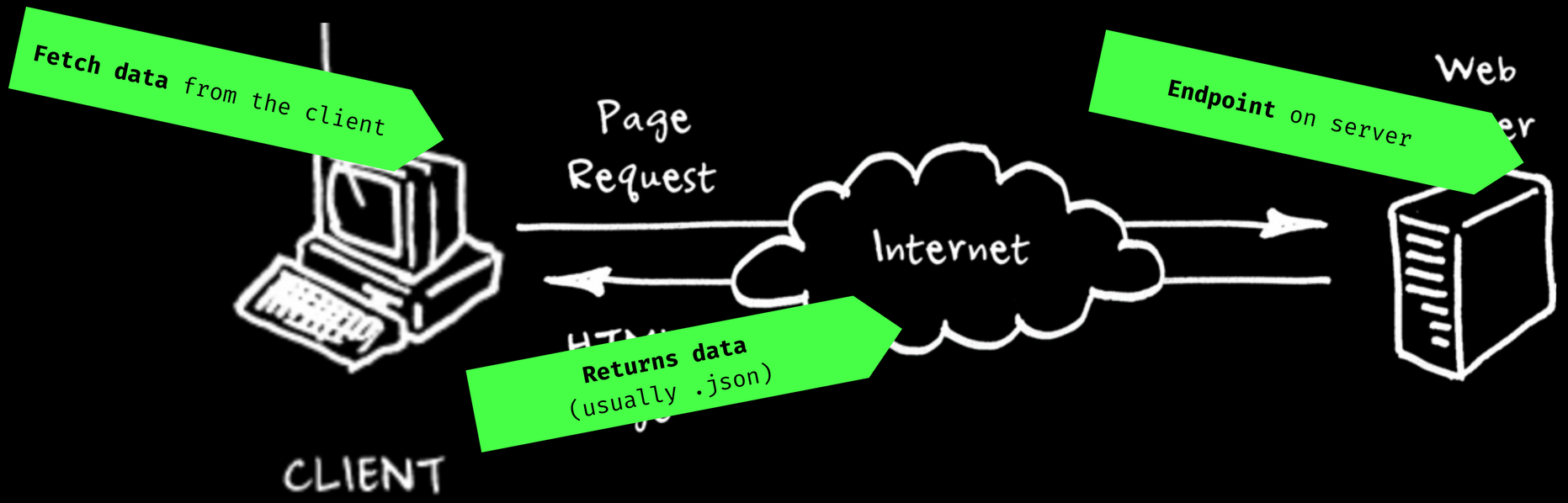
* **Browser API’s;** native and build into your web browser to expose data from the browser and device.

* **Third Party;** usually used for retrieving data. Fetching data from a server.

Api's

Endpoints

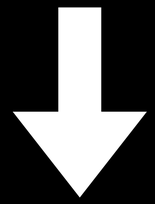
[..] a programmatic interface consisting of one or more **publicly exposed endpoints** to a defined request–response message system


Fetch data from the client
Page
Request
Internet
Endpoint on server
Web
Returns data
(usually .json)
CLIENT

Api’s

Endpoints

`/api/location/(woeid)/`

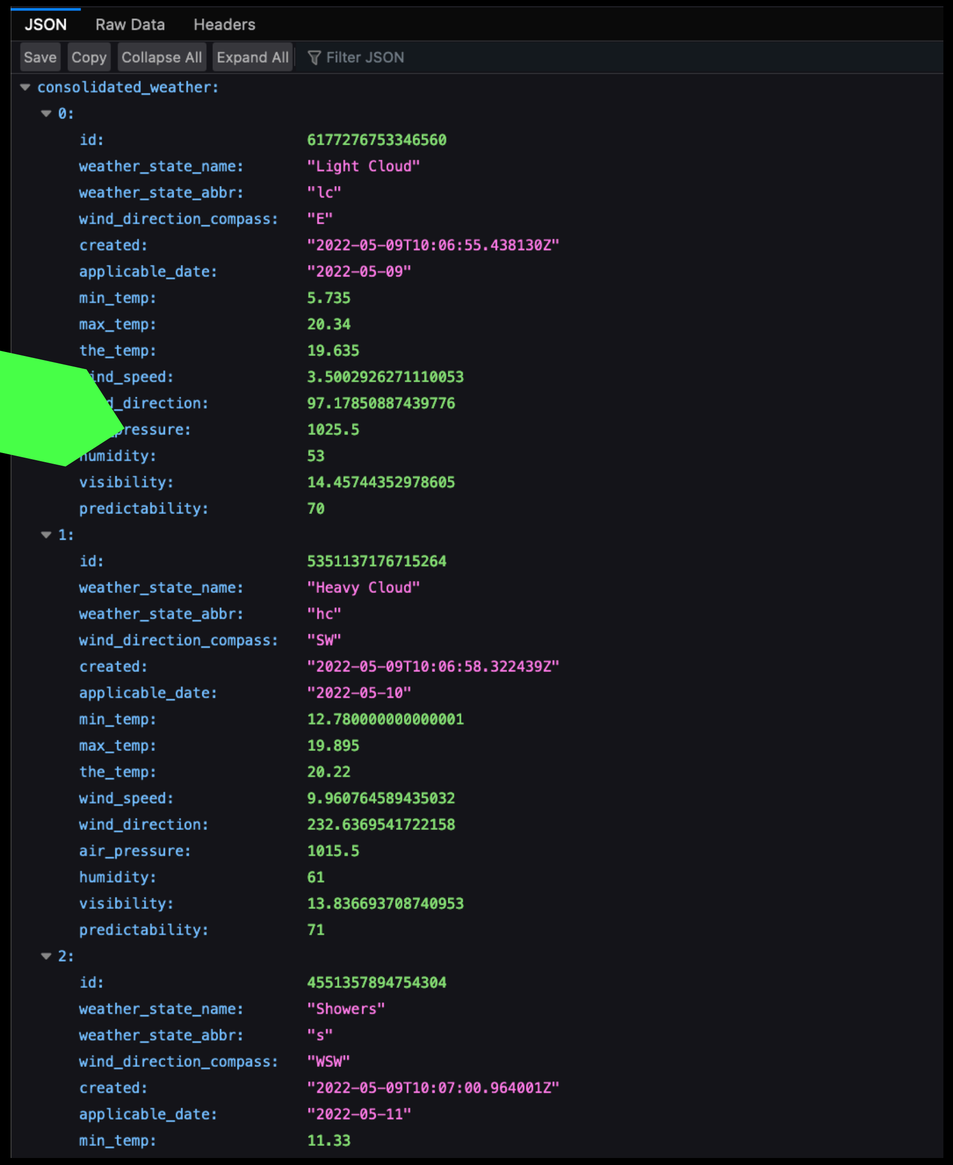


https://www.metaweather.com/api/location/727232/

Api's

Json

Returned data
(usually .json)



Api’s

Fetch

The Fetch API provides an interface for **fetching resources** (including across the network). […] returning a **promise which is fulfilled** once the response is available.

Api’s

Fetch

```
fetch('https://api.github.com/repos/cmda-bt/pt-course-21-22/stargazers')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(function(err) {
    console.log('Fetch Error :-S', err);
 });
```

Api’s

Fetch

```
async function fetchData() {
    const url = 'https://api.github.com/repos/cmda-bt/pt-course-21-22/stargazers';
    let response = await fetch(url);
    let stargazers = await response.json();
    console.log(stargazers);
}

fetchData();
```

Api’s

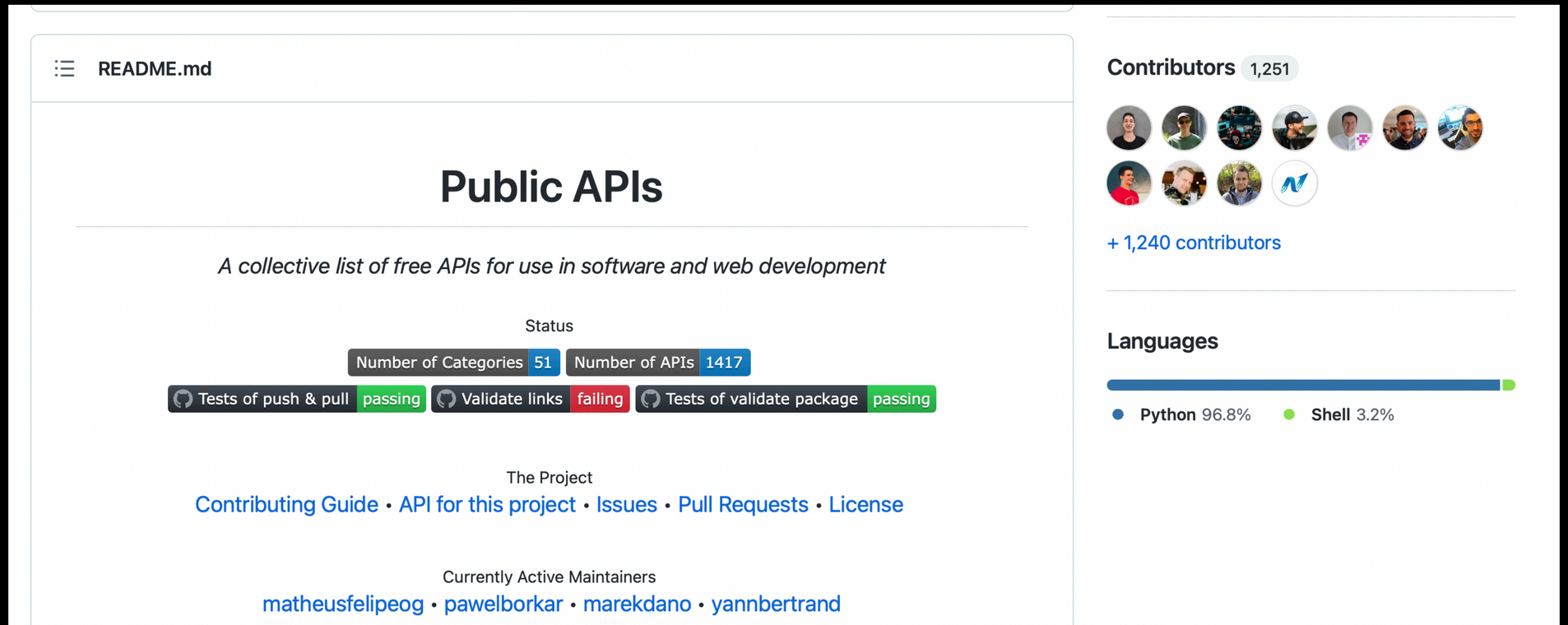
Authentication

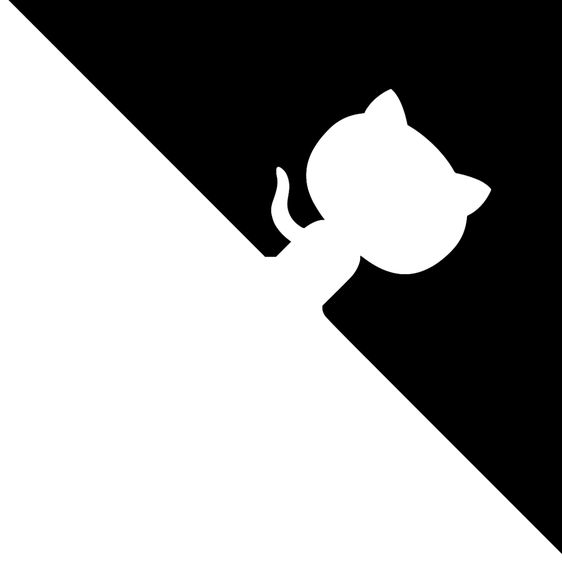
Third-party API’s usually require you to authenticate in some way.

- None (public)
- apiKey
- 0AUTH

Assignment (50m)

Search online for a public API's fetch that data and display it in the console or the browser.





Synopsis

- **Regular lesson**
- **Time:** 1:40h

Table of Contents

Practicum

continue working on **week-3**

exit;

see you in **lab-4**!