```
tt()
```

Tl;dr explain basic API architecture, async coding and fetching

Schedule

1. Sync vs Async
2. Async in JavaScript
3. Data formats
4. API Architecture (public, keys, oAuth)
5. Client vs Server fetch
6. What’s next?

Sync vs Async
(non-blocking)

Sync vs async

why?

Getting data from a resource (API) takes time. It needs to **fetch** the resource, parse it etc. But also, what if the data isn’t available (no internet connection e.g.) how should *errors* be handled?

Sync vs async

Only one thing can happen at a time, on a single main thread, and everything else is *blocked* until an operation completes.

MDN - Introducing asynchronous JavaScript

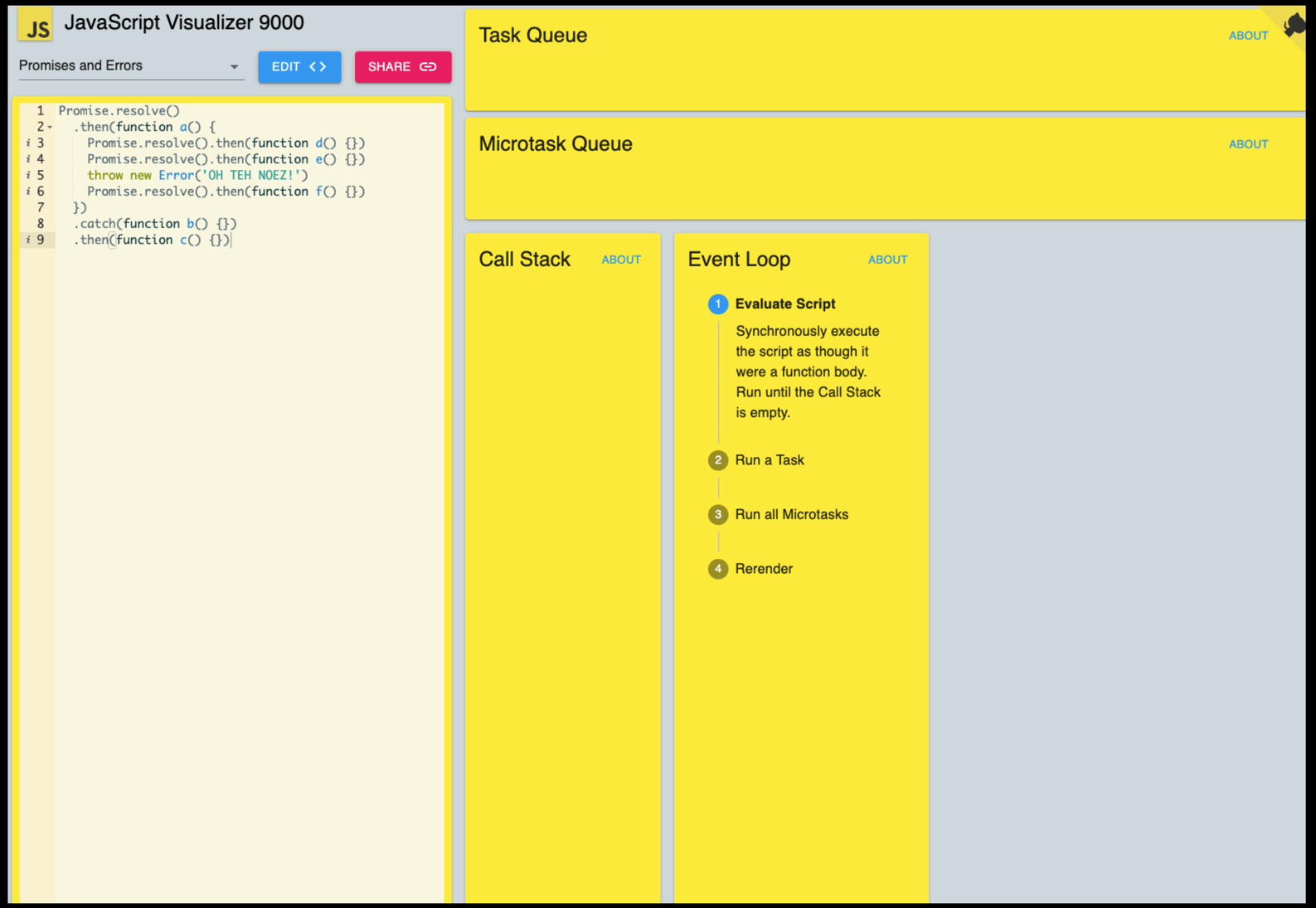
SYNCHRONOUS LOAD

ASYNCHRONOUS LOAD

Sync vs async

Many Web API features now use **asynchronous code to run,** especially those that access or *fetch some kind of resource from an external* device, such as fetching a file from the network, accessing a database and returning data from it.

MDN - Introducing asynchronous JavaScript



https://www.jsv9000.app/

Async in JavaScript

(Pasta promises)

Async in JavaScript

- Callbacks
- Promises
- Async / Await

Fetching in JavaScript

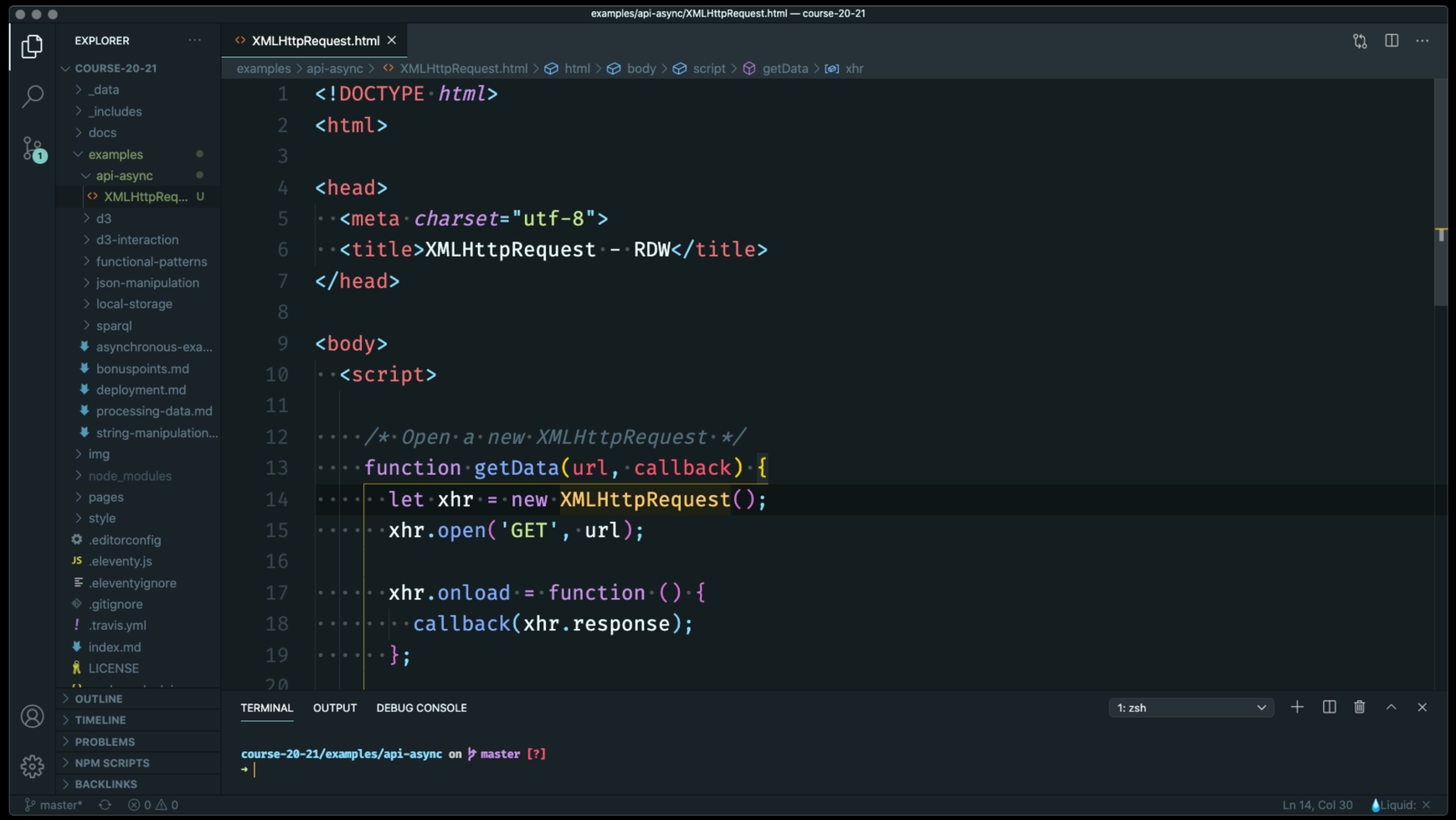
- Callbacks (XMLHTTPrequest)
- Promises (Fetch)
- Async / Await (Fetch)

Callbacks

definition

Async callbacks are functions that are specified as arguments when calling a function which *will start executing code in the background.* **When the background code finishes running, it calls the callback function.**

Callbacks



[XMLHttpRequest Example]

Promises

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

Promises

A promise can only succeed or fail once. It cannot succeed or fail twice, neither can it switch from success to failure or vice versa. A pending promise can either be **fulfilled with a value or rejected with a reason (error).**

WEB.DEV - Promises

Promises

A common need is to execute two or more asynchronous operations back to back, where each subsequent operation starts when the previous operation succeeds, with the result from the previous step. We accomplish this by creating a **promise chain.**

WEB.DEV - Promises

Promises

versus

Callback ‘hell’ (christmas tree)

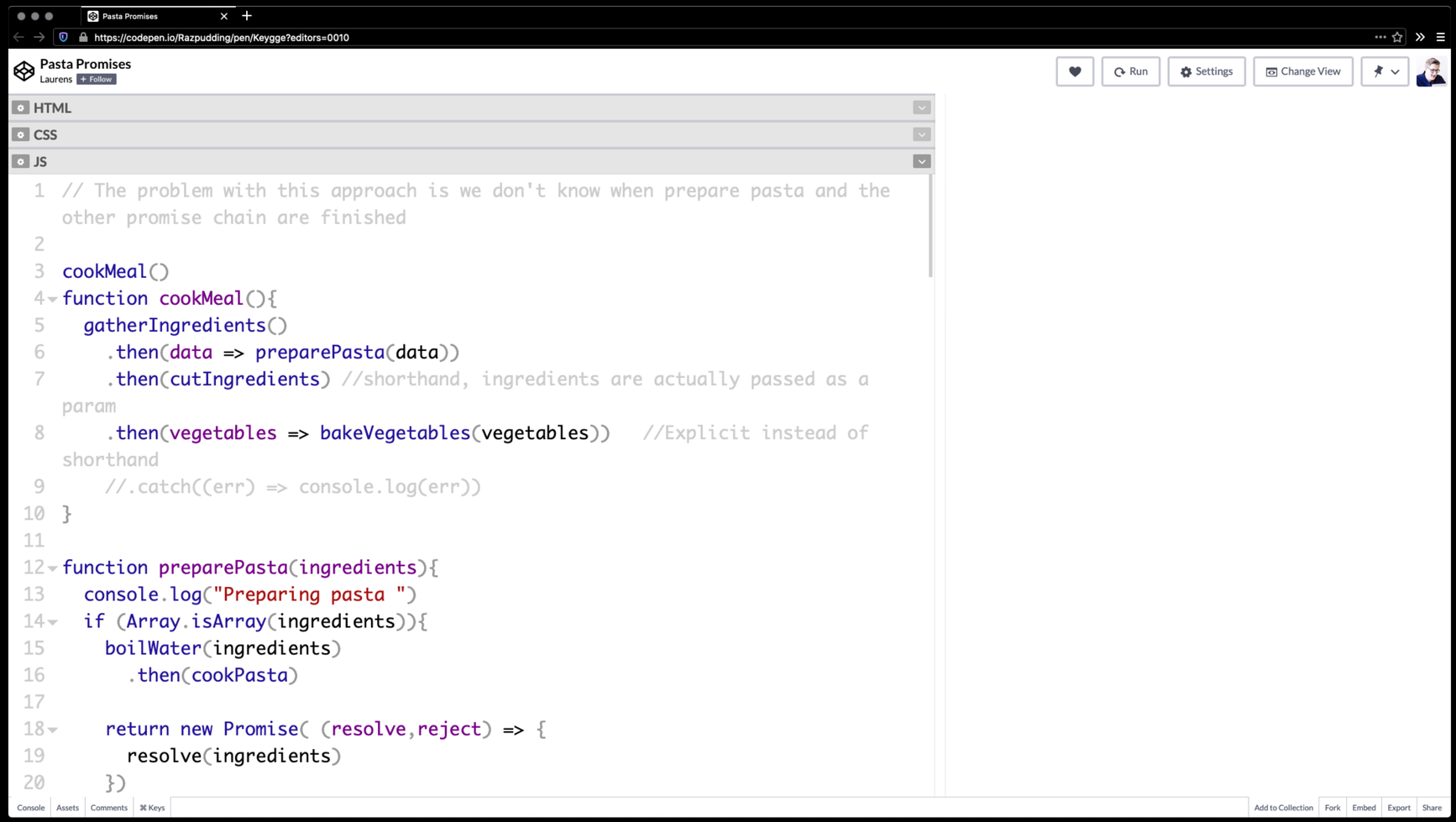
```
doSomething(function(result) {
  doSomethingElse(result, function(newResult) {
    doThirdThing(newResult, function(finalResult) {
      console.log('Got the final result: ' + finalResult);
    }, failureCallback);
  }, failureCallback);
}, failureCallback);
```

Promise chain

```
doSomething()
.then(function(result) {
  return doSomethingElse(result);
})
.then(function(newResult) {
  return doThirdThing(newResult);
})
.then(function(finalResult) {
  console.log('Got the final result: ' + finalResult);
})
.catch(failureCallback);
```

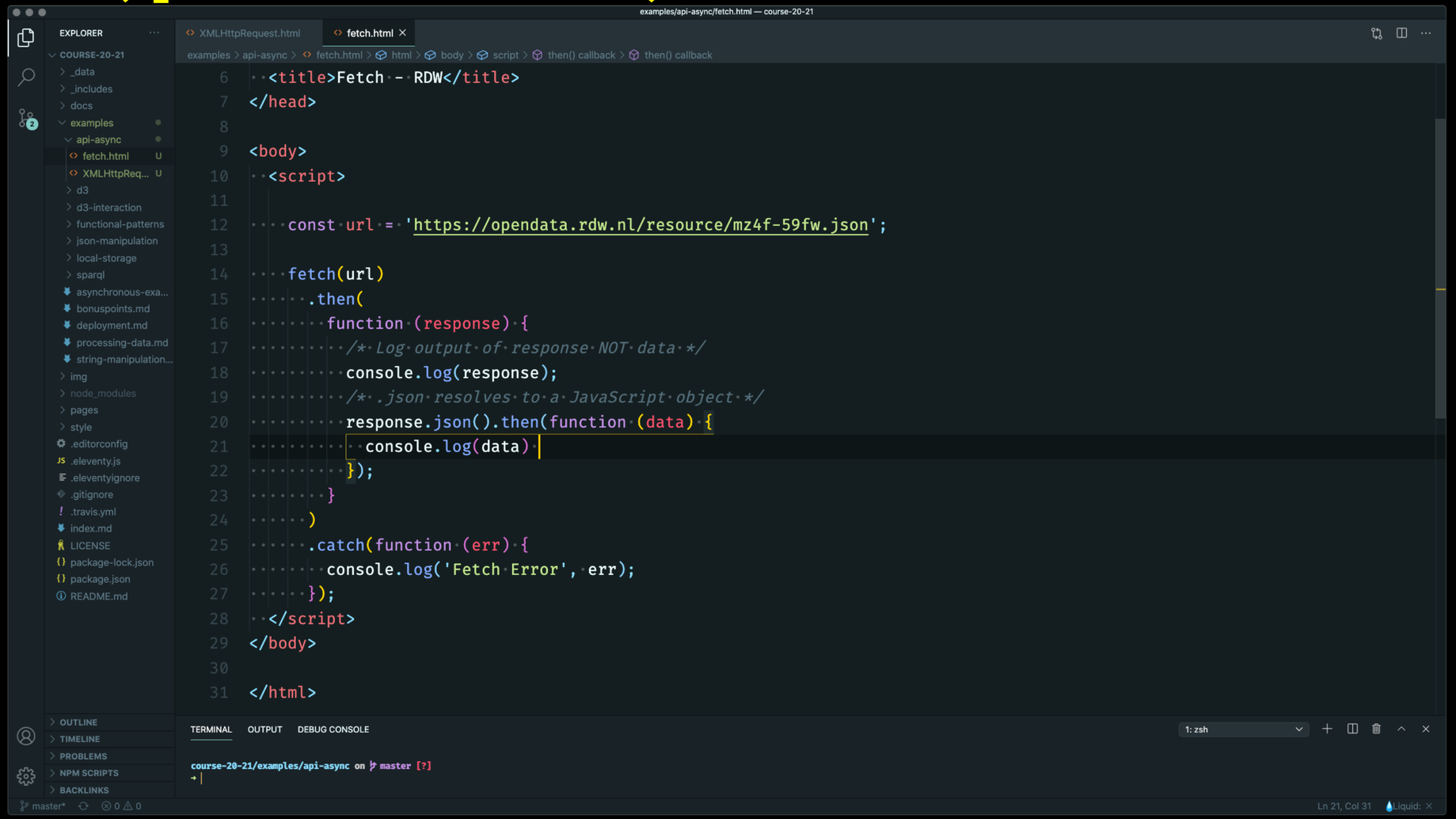
WEB.DEV - Promises

Promises



[Pasta Promise Example]

Fetch (promises)



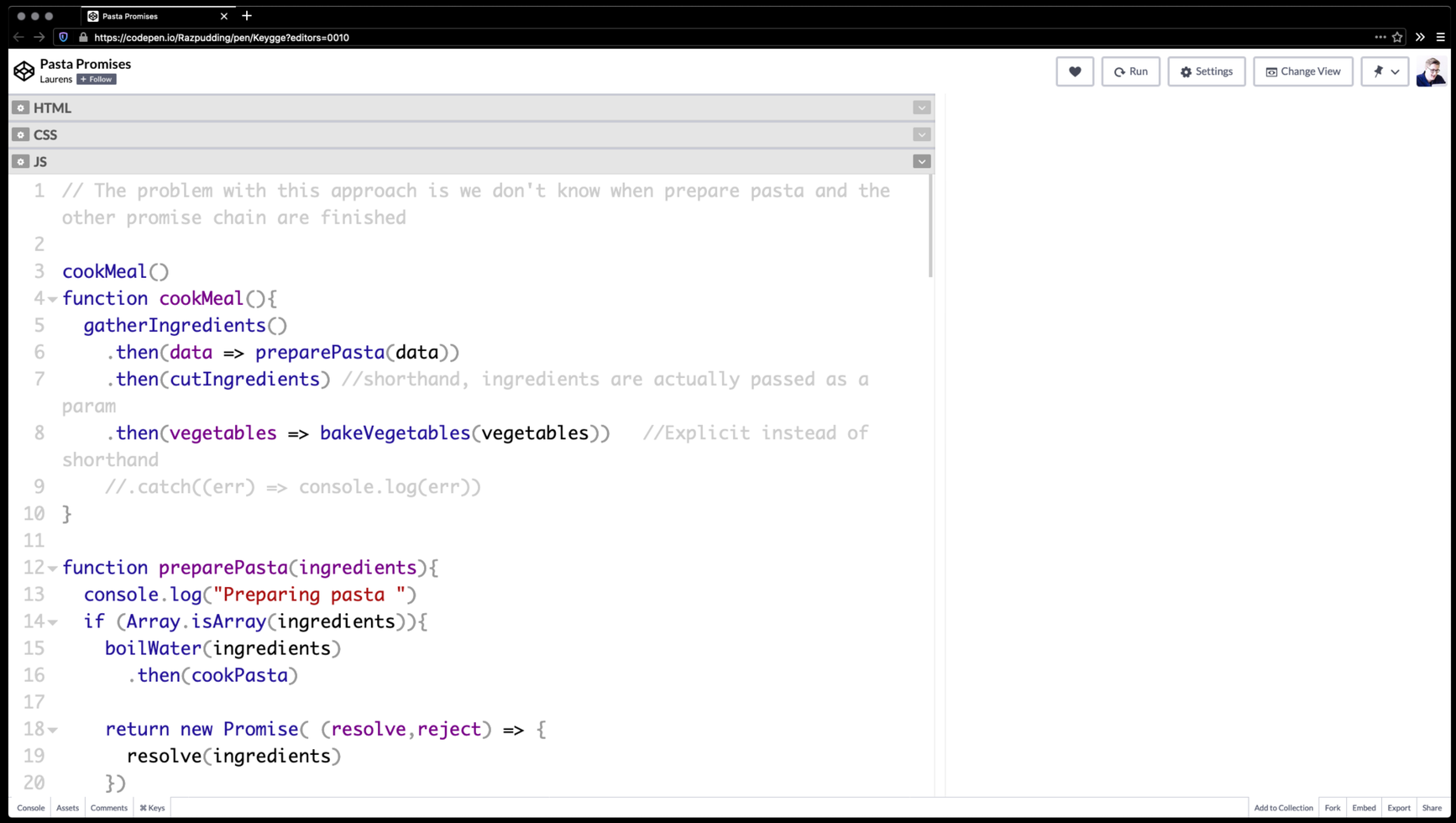
[Fetch Example]

Async Await

An async function is a function that knows how to e**xpect the possibility of the await keyword** being used to invoke asynchronous code. Act as *syntactic sugar* on top of promises, making asynchronous code easier to write and to read afterwards.

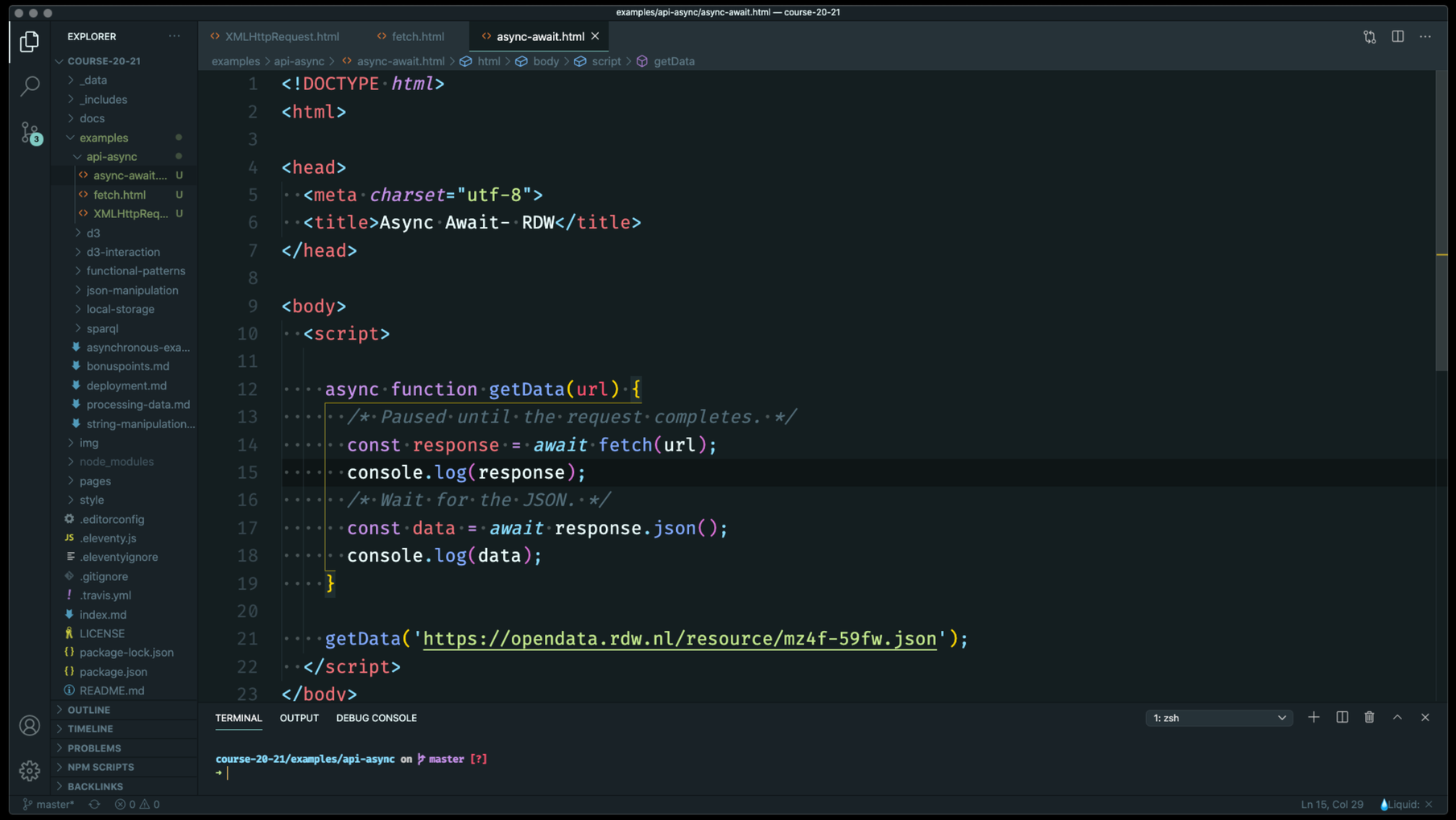
MDN - Making asynchronous programming

Async Await



[Pasta Async/Await Example]

Fetch (async / await)



[Async Await Example]

Error handling

With either method you choose, make sure you handle errors.

- .catch
- if statements (status codes)
- throw

Data Formats

(Transform vs Clean)

Gather Data

- **Get the data;** Use digital tools for automating. Put it into a spreadsheet.
- **Explore the data;** map it, count words. Use the graphic tools in spreadsheet.
- **Extend the data;** combine information from other sources. Fill in the blanks.

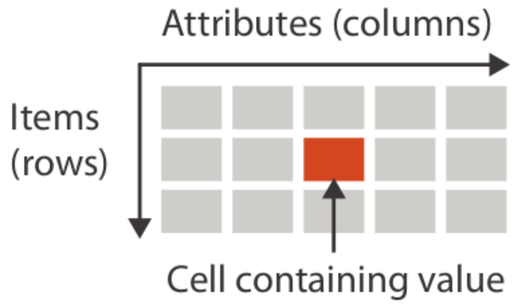
Gather Data

- **json** — **J**ava**S**cript **O**bject **N**otation
  like JavaScript, but no programming, just data
- **xml** — **Ex**tensible **M**arkup **L**anguage
  like HTML, but for anything
- **dsv** — **D**elimiter-**s**eparated **v**alues
  every row is a record, delimiter between fields

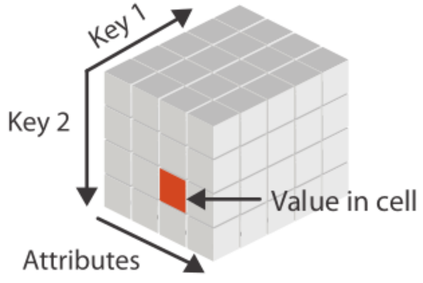
Three major datatypes

Dataset Types

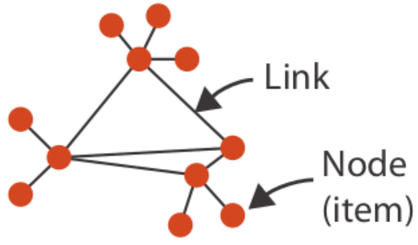
Tables



Multidimensional Table



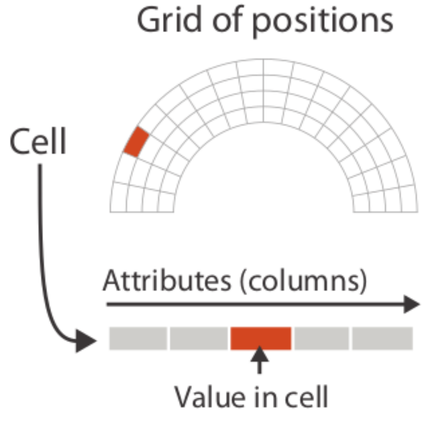
Networks



Trees

Spatial

Fields (Continuous)



Geometry (Spatial)

Attribute Types

- Categorical

- Ordered
  - *Ordinal*

  - *Quantitative*

	Point	Region
Space	(Latitude, Longitude) e.g. “My current location” **Quantitative**	Geographic Identifier e.g. Countries, States, Counties **Categorical**
Time	Instant e.g. “Right now” **Quantitative**	Interval e.g. “This year” **Ordinal**
Quantity	Value e.g. 5.2 **Quantitative**	Interval e.g. “5 - 10 years old” **Ordinal**

Gather Data

- **Data cleansing:**
  detecting and correcting (or removing)
  inaccurate records from a record set
  higher order functions (map etc.)

JavaScript

- **Data transforming:**
  converting data from one format or s
  another format or structure
  csv ⇥ json

D3.js

Maybe a small break?

API Architecture

(Data formats)

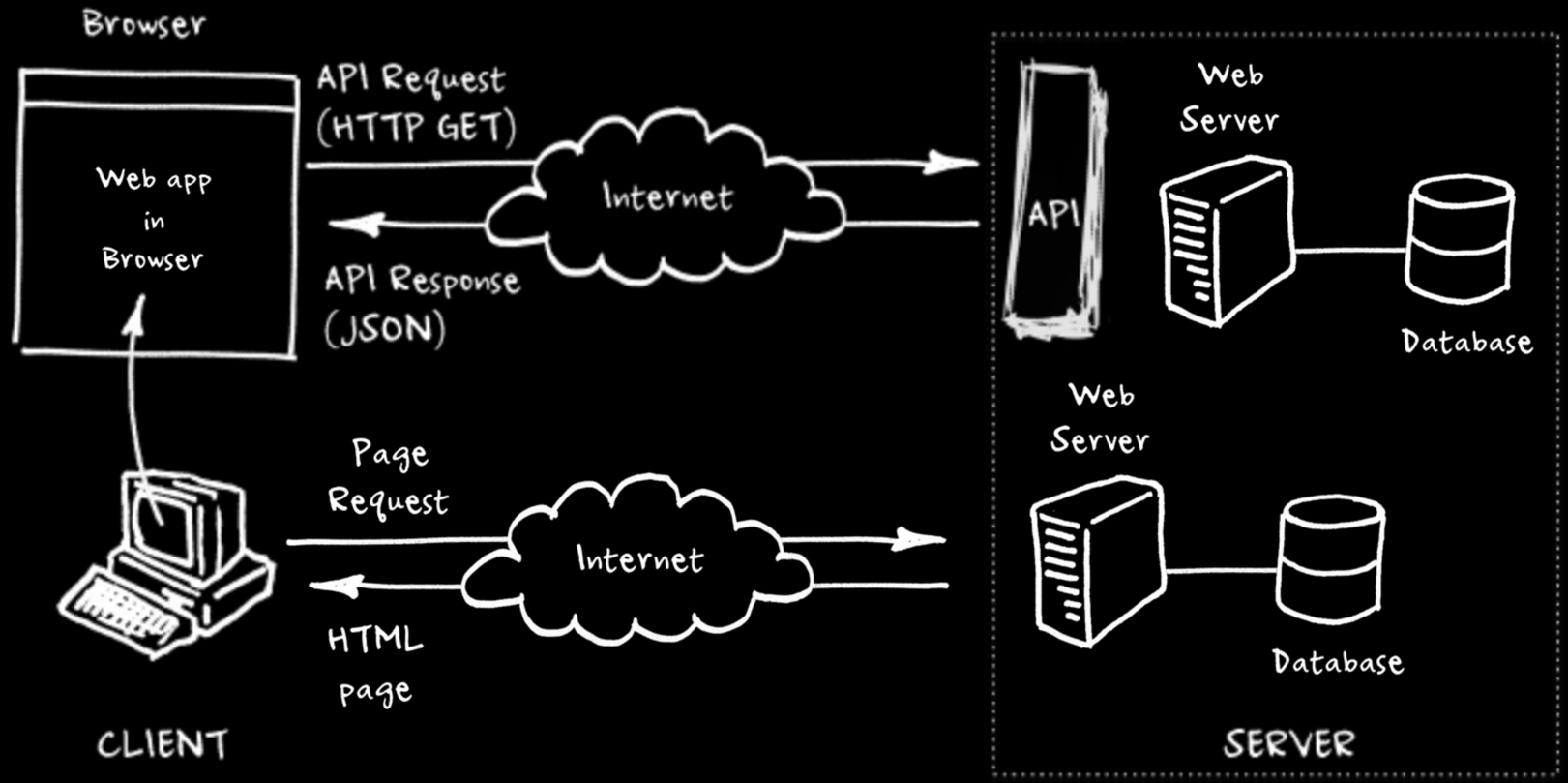
API definition

An API is an application programming interface. **It is a set of rules that allow programs to talk to each other.** The developer creates the API on the server and allows the client to talk to it.

Smashing - Understanding REST API’s

grand scheme

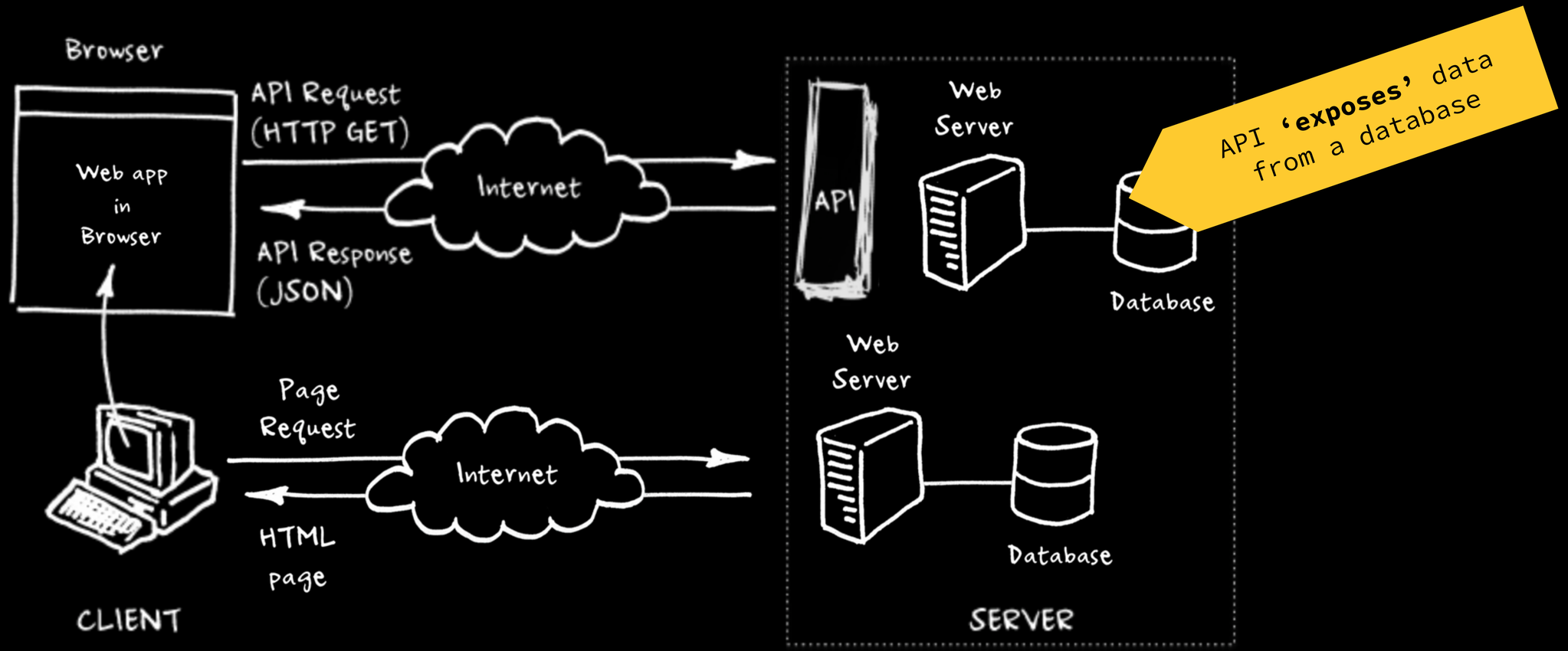
API



http://www.robert-drummond.com/2013/05/08/how-to-build-a-restful-web-api-on-a-raspberry-pi-in-javascript-2/

grand scheme

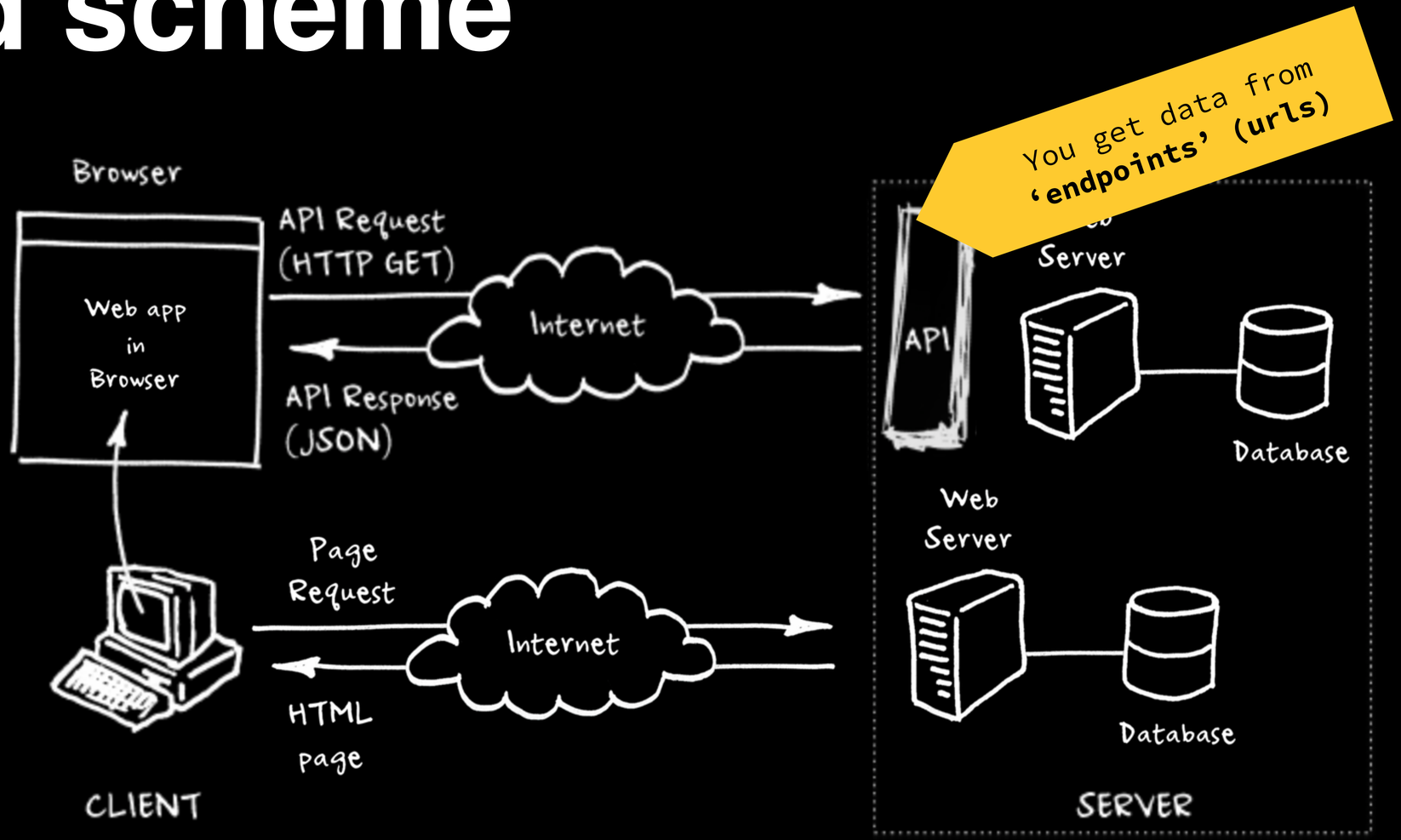
API



http://www.robert-drummond.com/2013/05/08/how-to-build-a-restful-web-api-on-a-raspberry-pi-in-javascript-2/

grand scheme

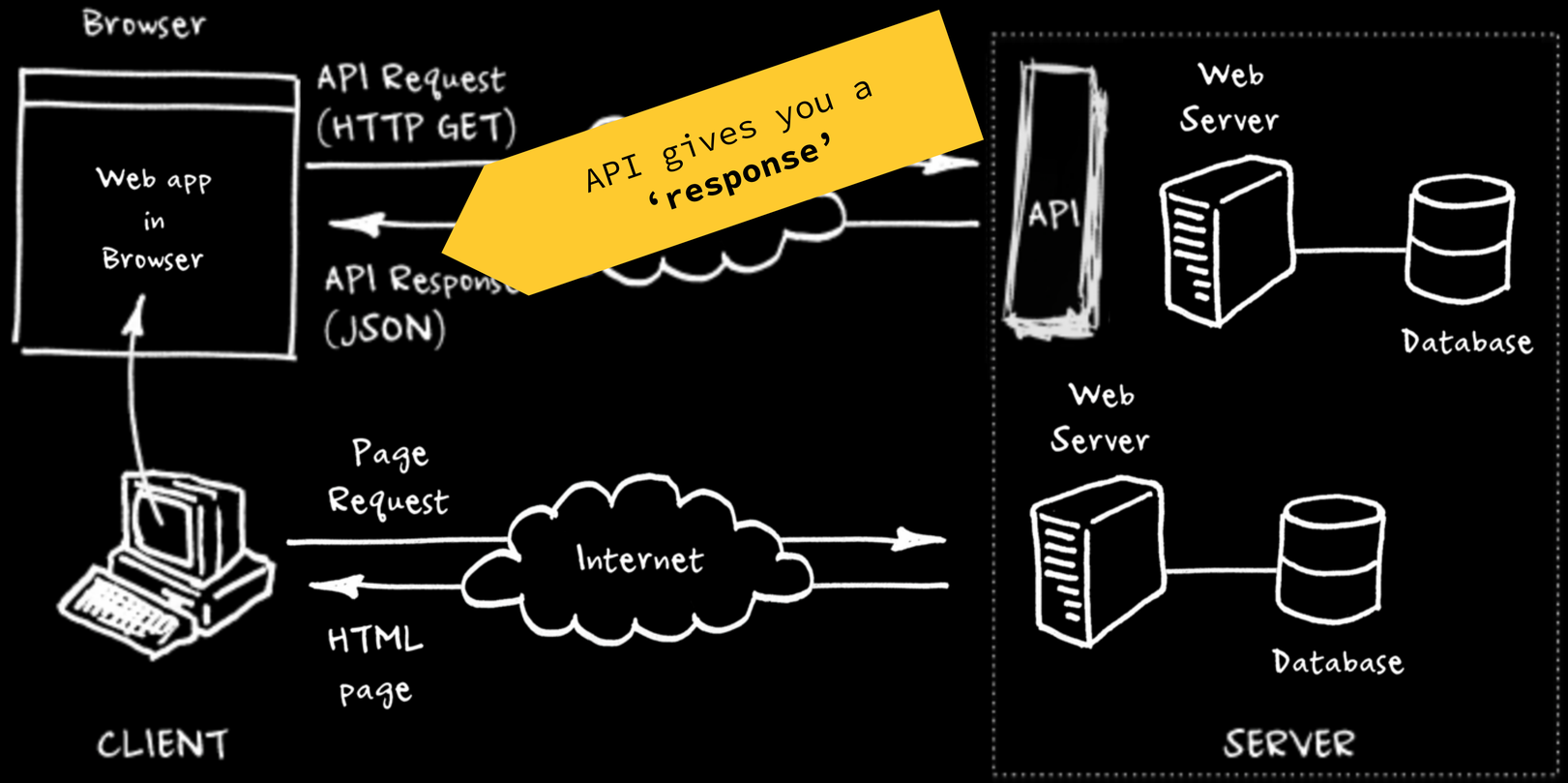
API



http://www.robert-drummond.com/2013/05/08/how-to-build-a-restful-web-api-on-a-raspberry-pi-in-javascript-2/

grand scheme

API



http://www.robert-drummond.com/2013/05/08/how-to-build-a-restful-web-api-on-a-raspberry-pi-in-javascript-2/

API

rest

REST determines how the API looks like. It stands for “Representational State Transfer”. It is a set of rules that developers follow when they create their API.

Smashing - Understanding REST API’s

Open Data Parkeren: PARKEERGEBIED

Parkeren

Data bekijken | Visualiseren en ontdekken | Exporteer | API | ...

Deze tabel legt een koppeling tussen de gebieden zoals de
de gebieden zoals deze voor Open Data Parkeren volgens
gepubliceerd worden.

Krijg toegang tot deze Dataset via SODA API

De Socrata Open Data API (SODA) voorziet een programmatische toegang tot deze dataset en de mogelijkheid om gegevens te filteren, op te vragen en te combineren.

API-documenten | Ontwikkelingsport

Endpoint and data format

API-eindpunt

https://opendata.rdw.nl/resource/mz4f-59 | JSON | Kopiëren

Blokkeren

Betreffende deze dataset

Bijgewerkt
25 oktober 2020

Laatst bijgewerkt op
25 oktober 2020

Metadata laatst bijgewerkt op
25 oktober 2020

Gemaakt op
28 oktober 2014

Weergaves
4.800

Downloads
6.776

Data voorzien door
(geen)

Eigenaar dataset
Open data team RDW

Licentie

Licentie	

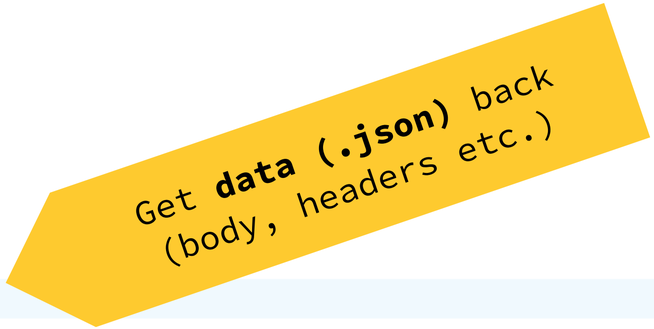
Onderwerpen

Categorie	Parkeren
Tags	parkeergebied, parkeren

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON


```
0:
    areamanagerid:   "344"
    areaid:          "3300"
    uuid:            "57d6f361-ffea-4186-a5a0-80a122c06fc3"
1:
    areamanagerid:   "796"
    areaid:          "020411"
    uuid:            "b6a5905f-c79f-4669-a43a-68eccef1f3c3"
2:
    areamanagerid:   "202"
    areaid:          "BC2"
    uuid:            "58de2c3d-2c38-4bb4-b653-30e2c9b90363"
3:
    areamanagerid:   "344"
    areaid:          "B7200"
    uuid:            "eab21b04-bcea-4ff4-858e-eedd6a24e5b7"
4:
    areamanagerid:   "262"
    areaid:          "1"
    uuid:            "f51fa39b-dda9-4e06-8374-1f6791c8be7c"
5:
    areamanagerid:   "599"
    areaid:          "599_10"
    uuid:            "dca24e79-98d6-46e7-980d-b14b461cfdd8"
6:
    areamanagerid:   "826"
    areaid:          "2"
    uuid:            "63f645d1-c2c0-4042-809e-4a217a6f2445"
7:
    areamanagerid:   "200"
    areaid:          "PAS2"
    uuid:            "5c89ef00-b5a2-4601-82f5-38090bea1351"
8:
    areamanagerid:   "141"
```

API (auths)

- **Public;** *open url* endpoint which can be freely ‘fetched’ (rate-limit).

chucknorris.io is a free JSON API for hand curated Chuck Norris facts. Read more

Subscribe for new Chuck Facts

Enter your email | Subscribe

Usage

Retrieve a random chuck joke in JSON format.

```
GET https://api.chucknorris.io/jokes/random
```

Example response:


```
{
"icon_url" : "https://assets.chucknorris.host/img/avatar/chuck-
norris.png",
"id" : "LBpfYFjlSfSfmw4cpJS6IA",
"url" : "",
"value" : "Chuck Norris can play slide guitar with a beer bottle. Or,
```

API (auths)

- **Public;** *open url* endpoint which can be freely ‘fetched’ (rate-limit).
- **apiKey;** *open url* endpoint which needs a ‘identifier key’ in the header or uri

OMDb API

The Open Movie Database

The OMDb API is a RESTful web service to obtain movie information, all content and images on the site are contributed and maintained by our users.

If you find this service useful, please consider making a one-time donation or become a patron.

Poster API

The Poster API is only available to patrons.

Currently over 280,000 posters, updated daily with resolutions up to 2000x3000.

Attention Users

04/08/19 - Added support for eight digit IMDb IDs.

01/20/19 - Supressed adult content from search results.

01/20/19 - Added Swagger files (**YAML**, **JSON**) to expose current API abilities and upcoming REST functions.

Become a Patron

Sponsors

Emby, Trakt, FileBot, Reelgood, Xirvik Servers, Yidio, mi.tv, Couchpop, What's on Netflix, Edu Reviewer, Flixboss, StreamingMoviesRight, Scripts on Screen, Writers Per Hour, Medium.com, Write my paper, Ramotion.com, Phone Trackers, Property for sale in Lake Como, iStarTips, What A Room, Vibelovely, StreamToday, Property for sale in Spain, Top Casinos Reviews, Classics on DVD, Streaming App, How to make a FinTech app, Vid2 - Create movie lists with AI

Usage

Send all data requests to:

```
http://www.omdbapi.com/?apikey=[yourkey]&
```

Poster API requests:

```
http://img.omdbapi.com/?apikey=[yourkey]&
```

Parameter

http://www.omdbapi.com/?apikey=[yourkey]

```
http://www.omdbapi.com/?apikey=[yourkey]
```

Unique Identifier

API (auths)

- **Public;** *open url* endpoint which can be freely ‘fetched’ (rate-limit).
- **apiKey;** *open url* endpoint which needs a ‘identifier key’ in the header or uri
- **oAuth;** authentication protocol with user identification before fetching

Access Token

The *access token* is a string which contains the credentials and permissions that can be used to access a given resource (e.g artists, albums or tracks) or user's data (e.g your profile or your playlists).

To use the *access token* you must include the following header in your API calls:

Header Parameter	Value
Authorization	Valid access token following the format: `Bearer <Access Token>`

Note that the *access token* is valid for 1 hour (3600 seconds). After that time, the token expires and you need to request a new one.

Examples

The following example uses cURL to retrieve information about a track using the Get a track endpoint:

```
curl --request GET \
    'https://api.spotify.com/v1/tracks/2TpxZ7JUBn3uw46aR7qd6V' \
     --header "Authorization: Bearer NgCXRK...MzYjw"
```

The following code implements the `getProfile()` function which performs the API call to the Get Current User's Porfile endpoint to retrieve the user profile related information:

```
async function getProfile(accessToken) {
  let accessToken = localStorage.getItem('access_token');

  const response = await fetch('https://api.spotify.com/v1/me', {
    headers: {
      Authorization: 'Bearer ' + accessToken
    }
  });

  const data = await response.json();
}
```

Multi-fetch

Often API’s uses **multiple endpoints** for different types of data (collections vs individual items) or **limit the amount of items** that get returned in each request (pagination).

RIJKS DATA

Search RijksData

RijksData on www.rijksmuseum.nl

- Home
- Object metadata
  - API
  - Harvest
  - Download
- Bibliographic data
- Controlled vocabularies
- User-generated content

Object metadata / API

Object metadata APIs

The object metadata APIs make the power of the award-winning Rijksmuseum website directly accessible to developers. Searching the collection through the API offers a wide range of interesting possibilities, as do the tiled images used to zoom in to close-ups of objects. The JSON-based service is so easy to use that you can create an application using the Rijksmuseum's rich and freely accessible content in no time.

Access to APIs

To start using the data and images, you first need to obtain an API key by registering for a Rijksstudio account. You will be given a key instantly upon request, which you can find at the advanced settings of your Rijksstudio account.

Collection API

`GET /api/[culture]/collection` gives access to the collection with brief information about each object. The results are split up in result pages. By using the `p` and `ps` parameters you can fetch more results, up to a total of 10,000. All of the other parameters are identical to the advanced search page on the Rijksmuseum website. You can use that page to find out what's the best query to use.

Parameter	Format	Default	Notes
`key`	`a-z|0-9`		Your API-key, mandatory for every request.
`format`	`json` / `jsonp` / `xml`	`json`	The format of the result.
`culture`	`nl` / `en`		The language to search in (and of the results).
`p`	`0-n`	`0`	The result page. Note that `p * ps` cannot exceed 10,000.
`ps`	`1-100`	`10`	The number of results per page.
`q`	`a-z`		The search terms that need to occur in one of the fields of the object data.
`involvedMaker`	`a-z`		Object needs to be made by this agent.
`type`	`a-z`		The type of the object.
`material`	`a-z`		The material of the object.
`technique`	`a-z`		The technique used to make the object.
`f.dating.period`	`0-21`		The century in which the object is made.

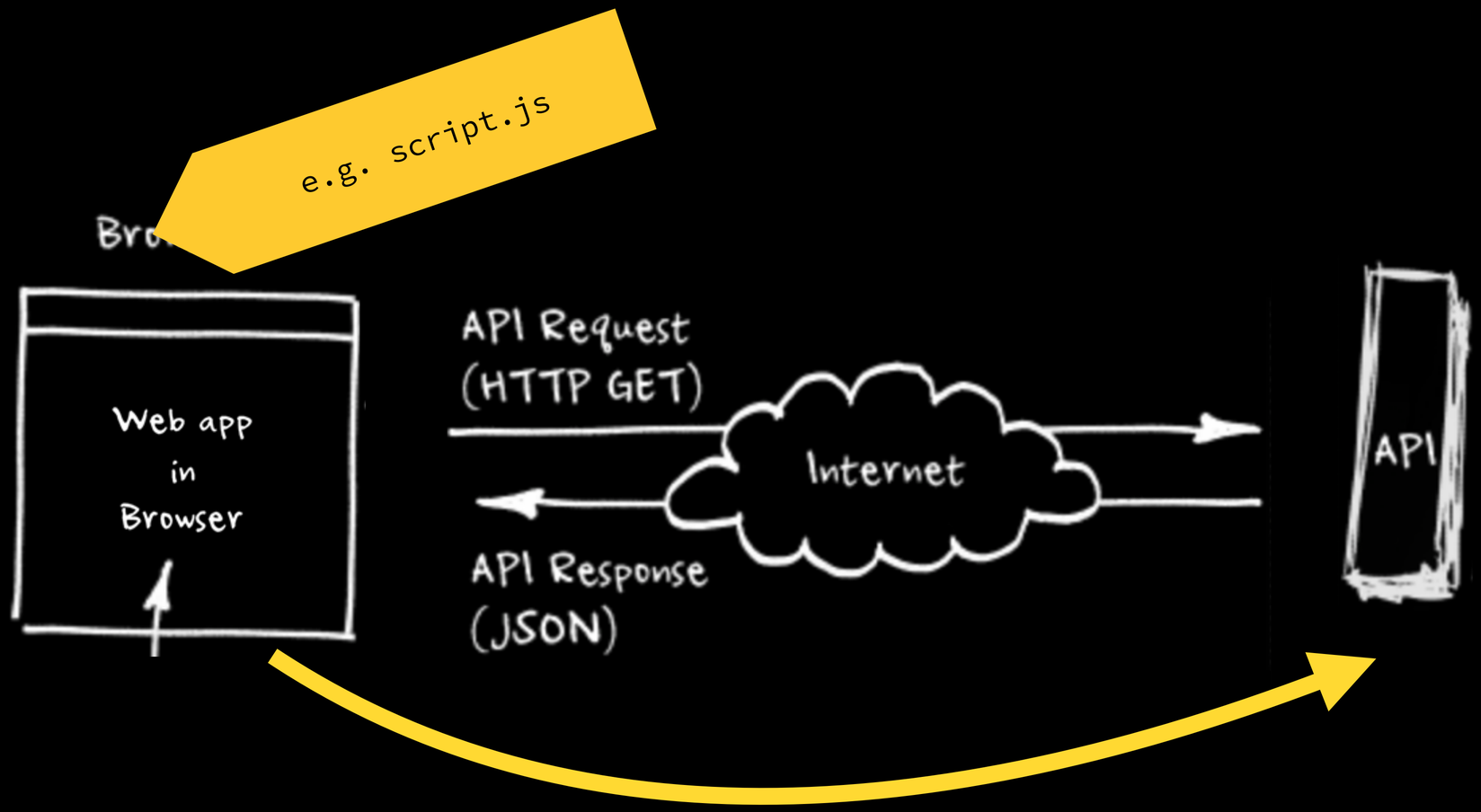
https://data.rijksmuseum.nl/object-metadata/api/

Client vs. Server

(Fetch)

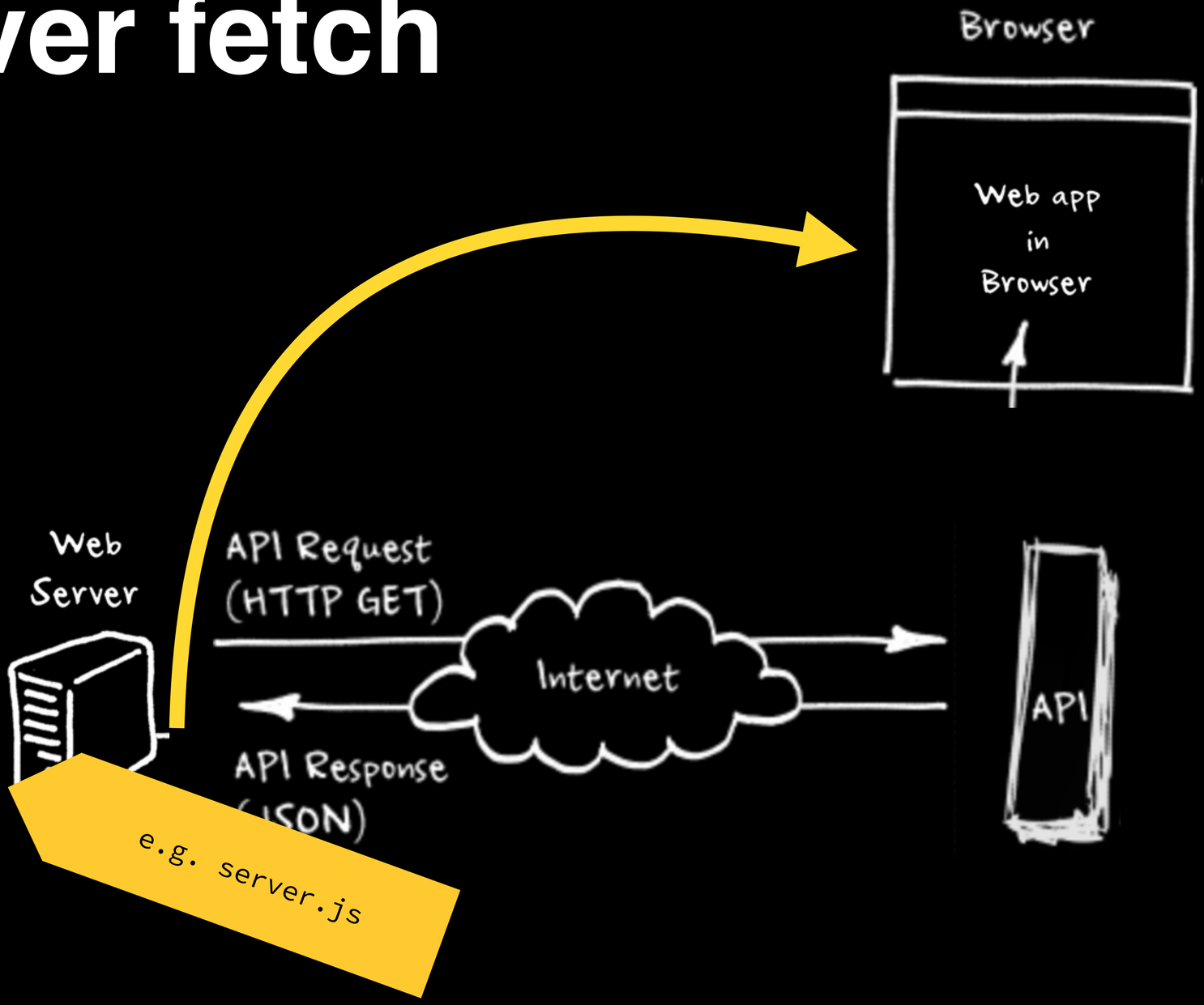
Client fetch

API

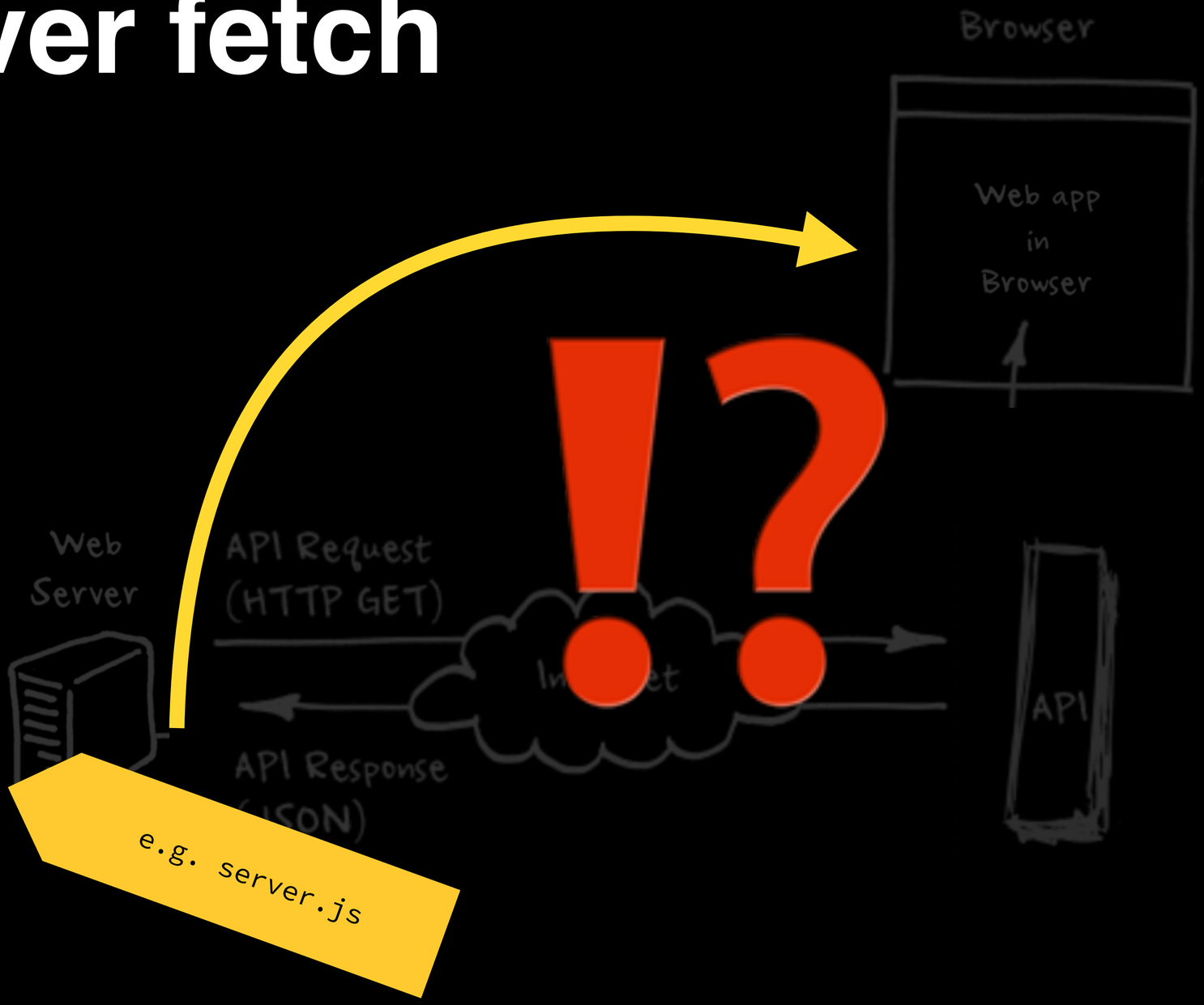


Server fetch

API



Server fetch



API

Server fetch

API keys are identifiers (unique) and acts as a password and are usually app-specific. **You don’t want your password public** (either on the client in the JavaScript) or in your GitHub repository.

.env

```
# .env file
USER_ID="239482"
USER_KEY="foobar"
NODE_ENV="development"
```

server.js

```
require('dotenv').config();

process.env.USER_ID; // "239482"
process.env.USER_KEY; // "foobar"
process.env.NODE_ENV; // "development"
```

ADVANCED

Server-only modules

Edit this page on GitHub

Like a good friend, SvelteKit keeps your secrets. When writing your backend and frontend in the same repository, it can be easy to accidentally import sensitive data into your front-end code (environment variables containing API keys, for example). SvelteKit provides a way to prevent this entirely: server-only modules.

Private environment variables

The `$env/static/private` and `$env/dynamic/private` modules, which are covered in the modules section, can only be imported into modules that only run on the server, such as hooks.server.js or +page.server.js.

Your modules

You can make your own modules server-only in two ways:

- adding `.server` to the filename, e.g. `secrets.server.js`
- placing them in `$lib/server`, e.g. `$lib/server/secrets.js`

How it works

Any time you have public-facing code that imports server-only code (whether directly or indirectly)...

```
$lib/server/secrets.js
export const atlantisCoordinates = [/* redacted */];
```

```
src/routes/utils.js
export { atlantisCoordinates } from '$lib/server/secrets.js';

export const add = (a, b) => a + b;
```

```
src/routes/+page.svelte
```

```
import fetch from 'node-fetch';
import dotenv from 'dotenv';

dotenv.config();

const url = 'https://api.schiphol.nl/public-flights/flights';
const params = {
    departure: '?flightDirection=D&route=',
    arrival: '?flightDirection=A&route='
};
const params2 = '&includedelays=false&page=0&sort=%2BscheduleTime&fromDateTime=';
const params3 = '&searchDateTimeField=scheduleDateTime';
const headers = {
    resourceversion: 'v4',
    app_id: process.env.APP_ID,
    app_key: process.env.APP_KEY,
    Accept: 'application/json'
};

const fetchData = async (iata, dateTime, direction) => {
    try {
        const res = await fetch(`${url}${params[direction]}${iata}${params2}${dateTime}${params3}`,
            method: 'GET',
            headers
        });
```

https://github.com/ninadepina/tech-track-23-24/blob/main/app/src/lib/fetchIata.js

Uncaught SyntaxError
Unexpected end of input