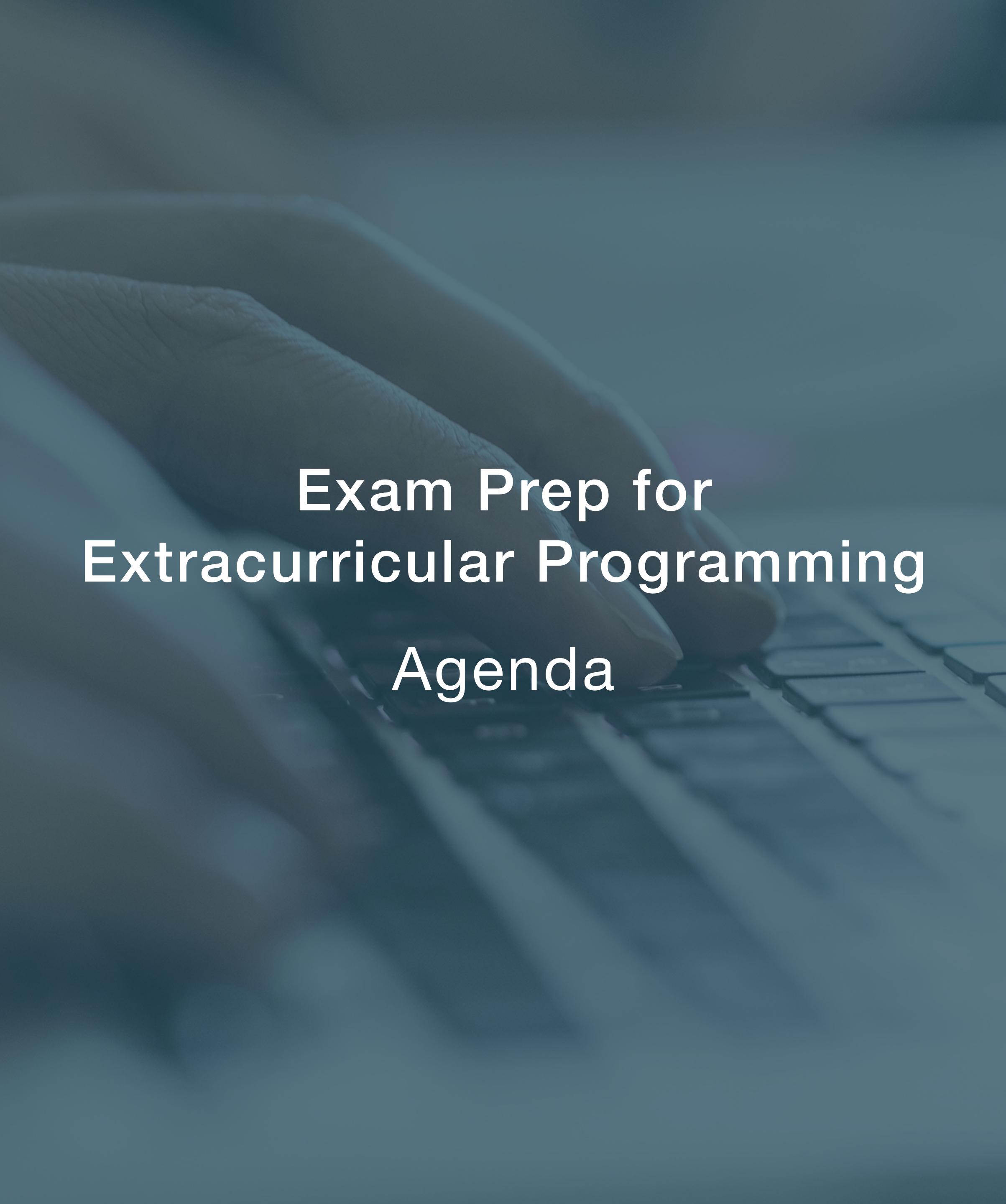




ERHVERVSAKADEMI AARHUS
BUSINESS ACADEMY AARHUS

Exam Prep for Extracurricular Programming

Welcome Back Programming 😊

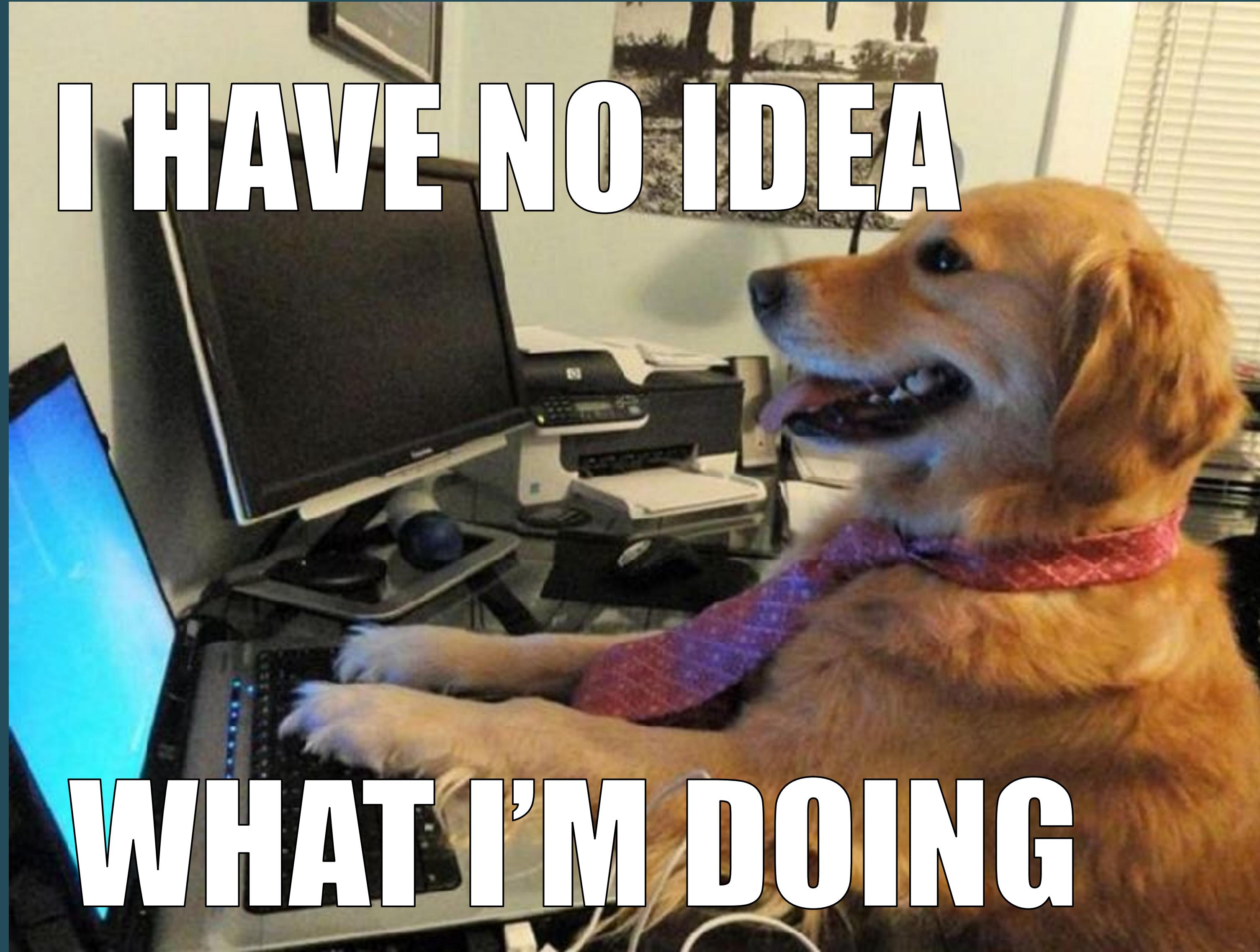


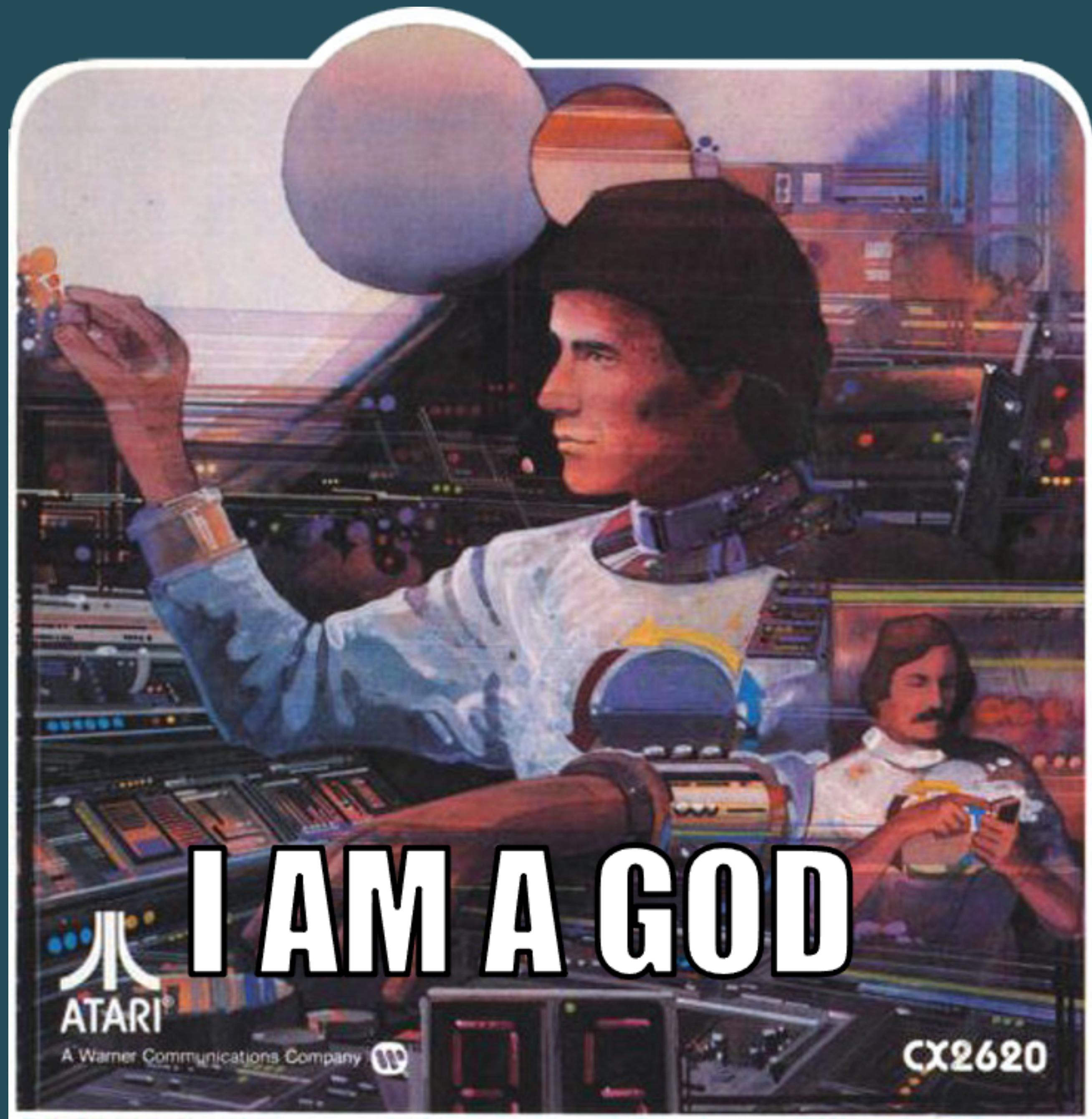
Exam Prep for Extracurricular Programming Agenda

1. Client Server
2. Fetch, JSON & DOM Manipulation
3. Headless WordPress
4. Recap on JS terms & concepts
5. Exam Prep

I HAVE NO IDEA

WHAT I'M DOING





I'm Rasmus Cederdorff (RACE)

Senior Lecturer

Freelance Web App Developer

- Programming with an eye for UI and UX.
- Web Development, JavaScript, React, BaaS & Node.js
- “I speak JavaScript”.
- Websites, Webshops, Web Apps, Mobile Apps, Server App and BaaS.







I'm Rasmus Cederdorff
Aarhus (Tilst)
Alicia & Ida

From Holstebro
I'm into sports
Love (apple) gadgets, to take
pictures & interior design
projects

What's with my arm?

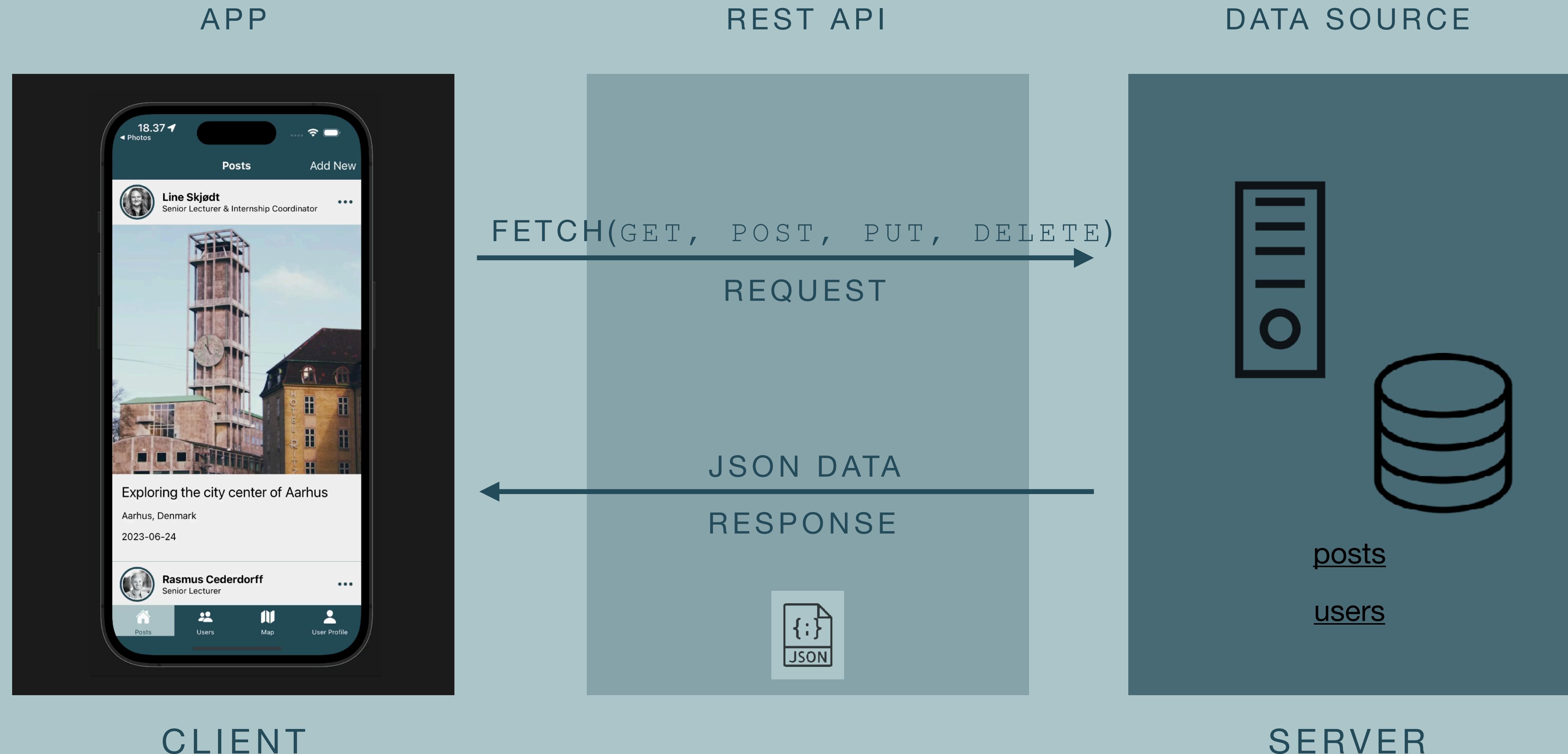




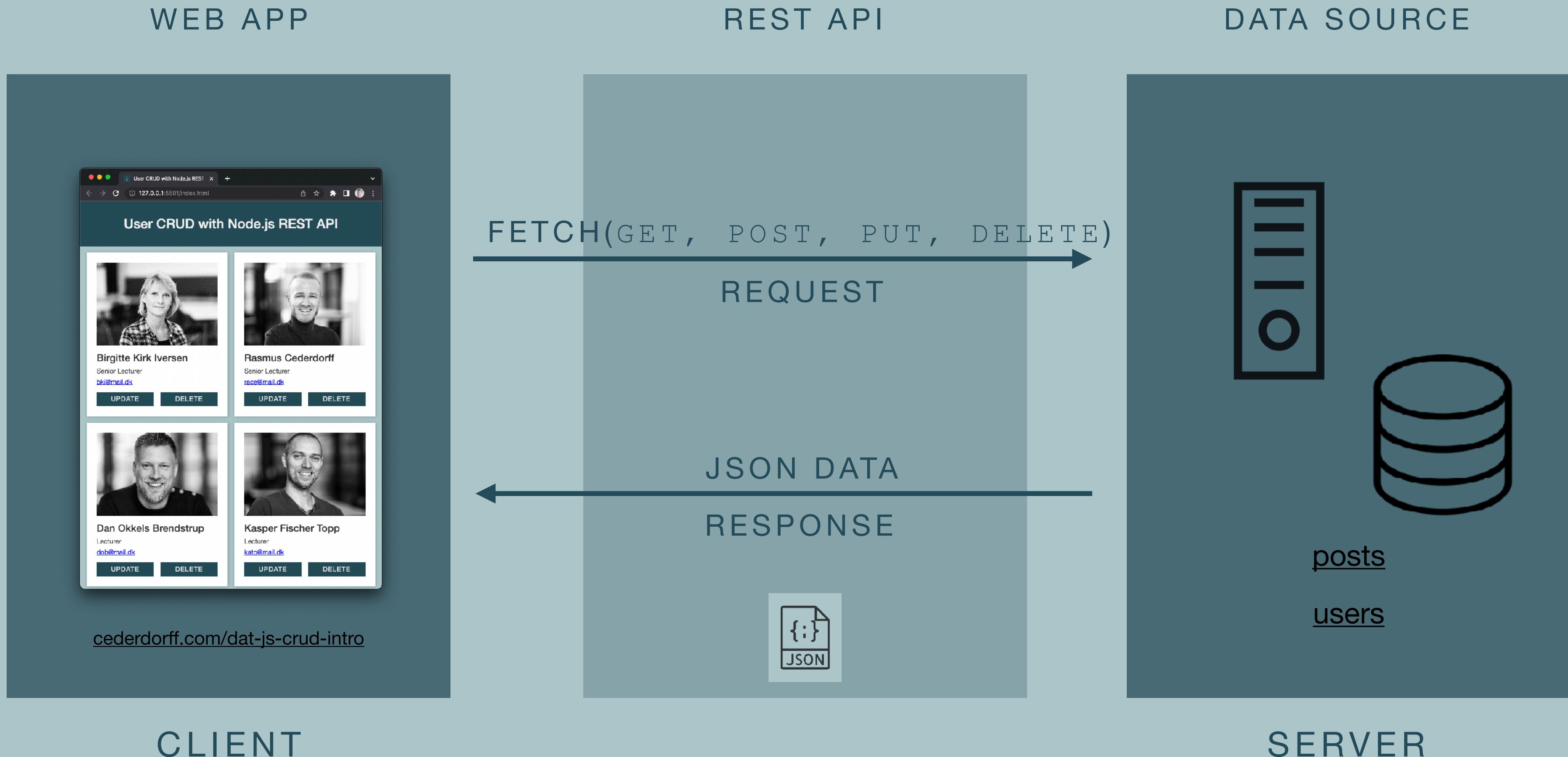
Client Server

The Basic Architecture of the Web

Client Server Architecture

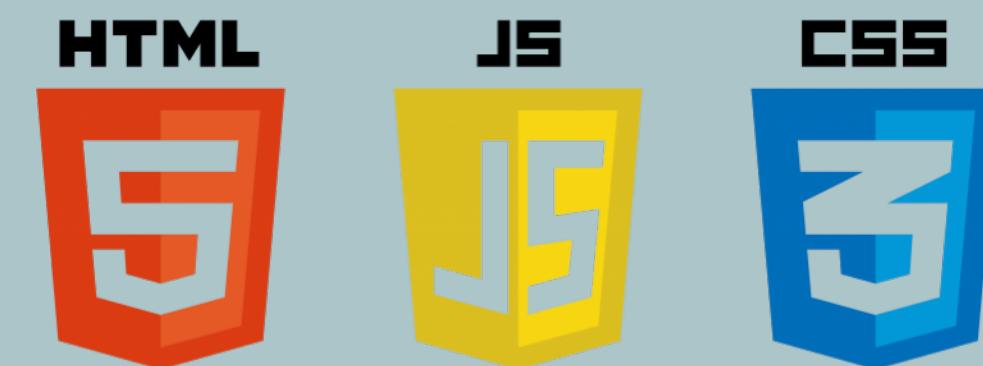


Web Development



Webudvikling

FRONTEND

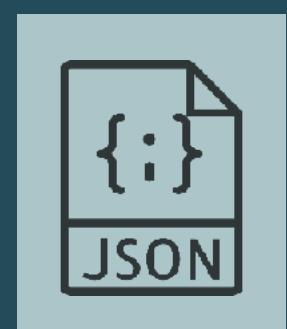


REST API

FETCH(GET, POST, PUT, DELETE)

REQUEST

JSON DATA
RESPONSE

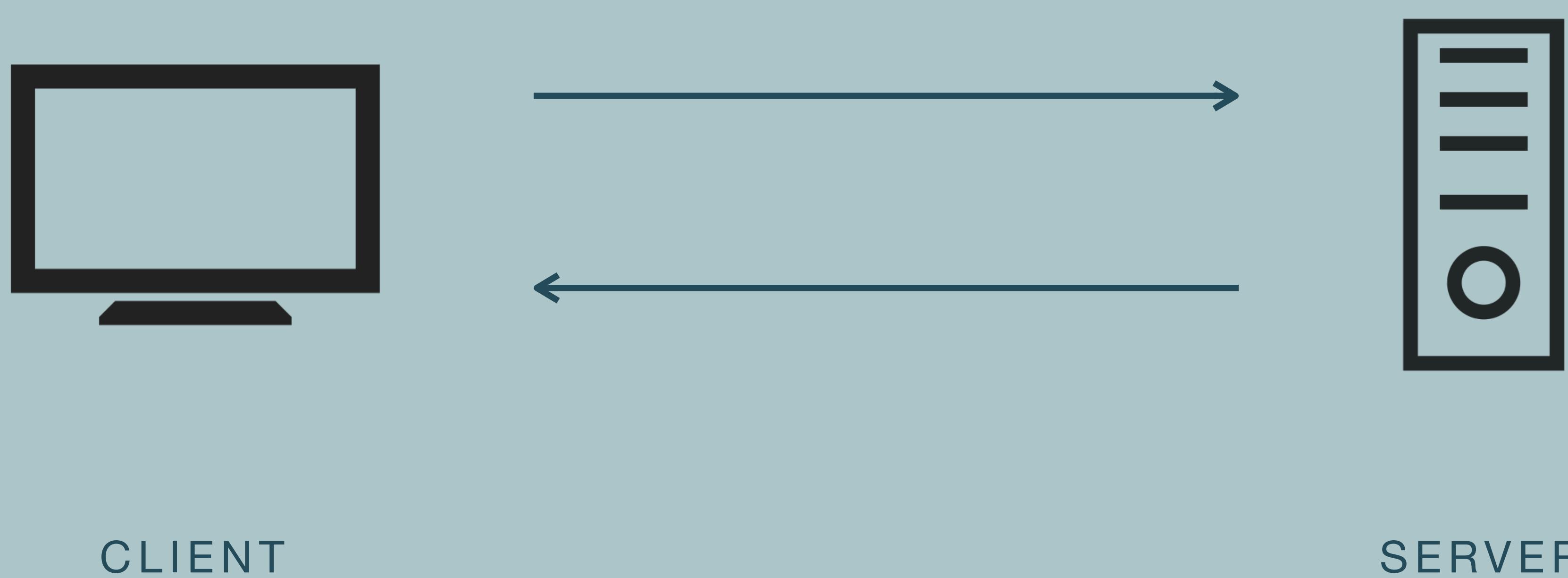


BACKEND



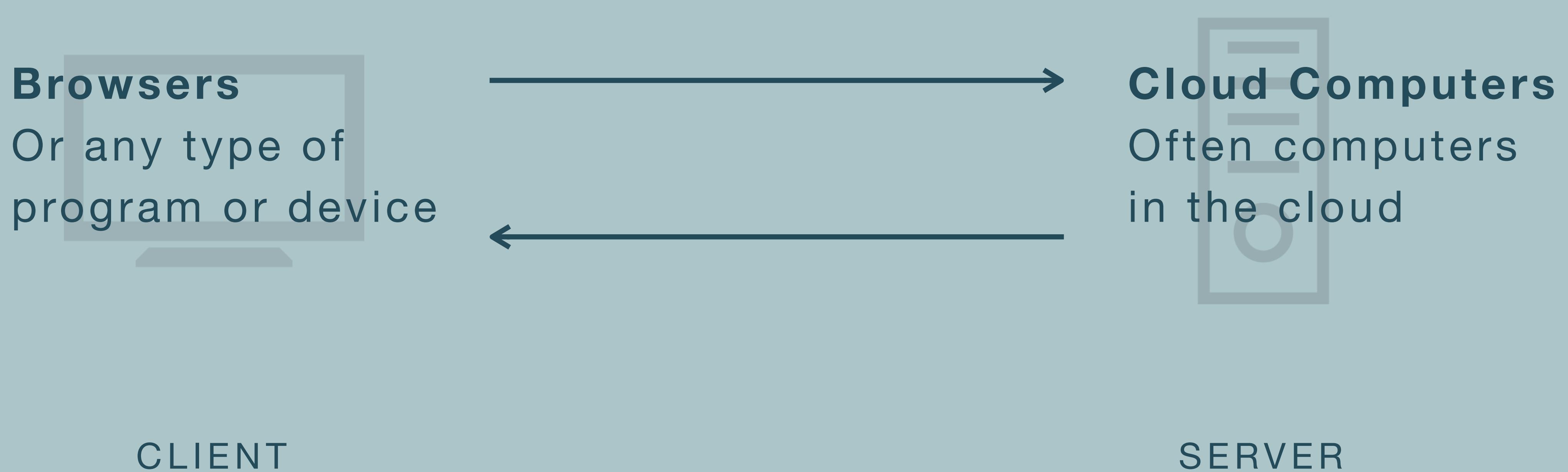
Client-Server Model

Communication between web **clients** and web **servers**.



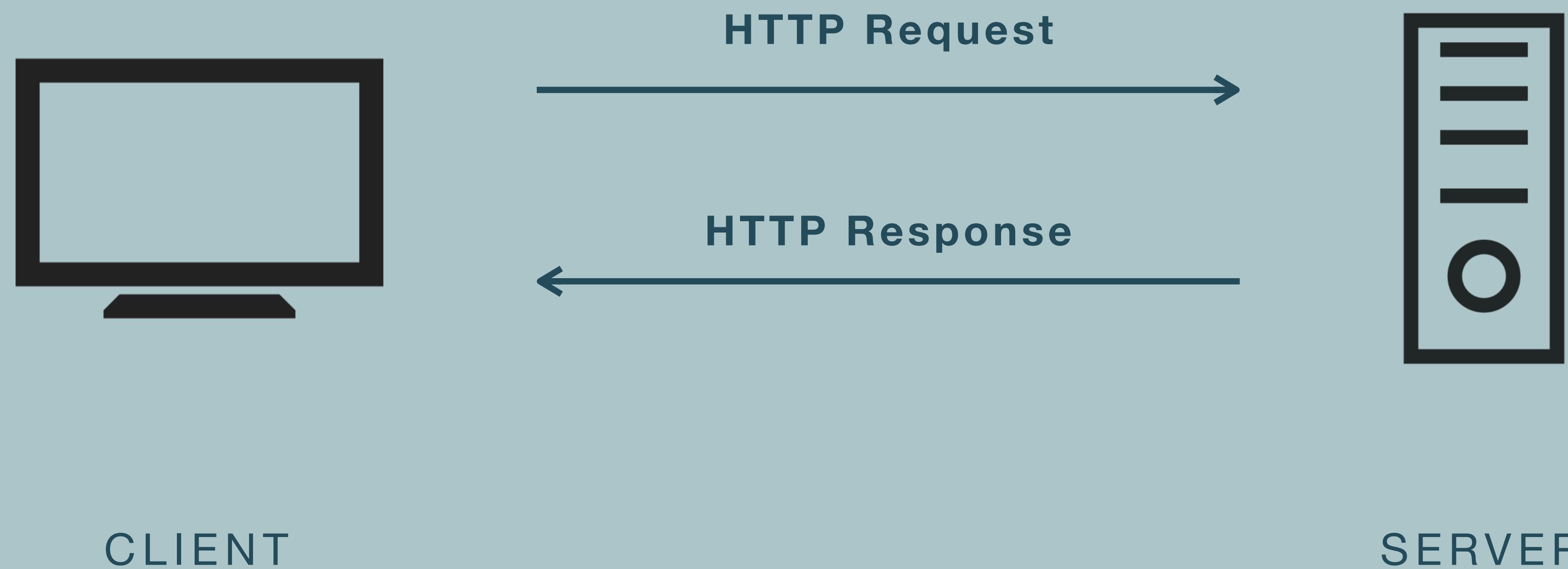
Client-Server Model

Communication between web **clients** and web **servers**.



Client-Server Model

Communication between web **clients** and web **servers**.



Hyper Text Transfer Protocol

- A protocol and standard for fetching data, HTML and other resources (text, images, videos, scripts, JSON).
- The foundation of the web.



What is HTTP

Not Secure | w3schools.com/whatis/whatis_http.asp

HTML CSS JAVASCRIPT SQL PYTHON

HTTP Request / Response

Communication between clients and servers is done by **requests** and **responses**:

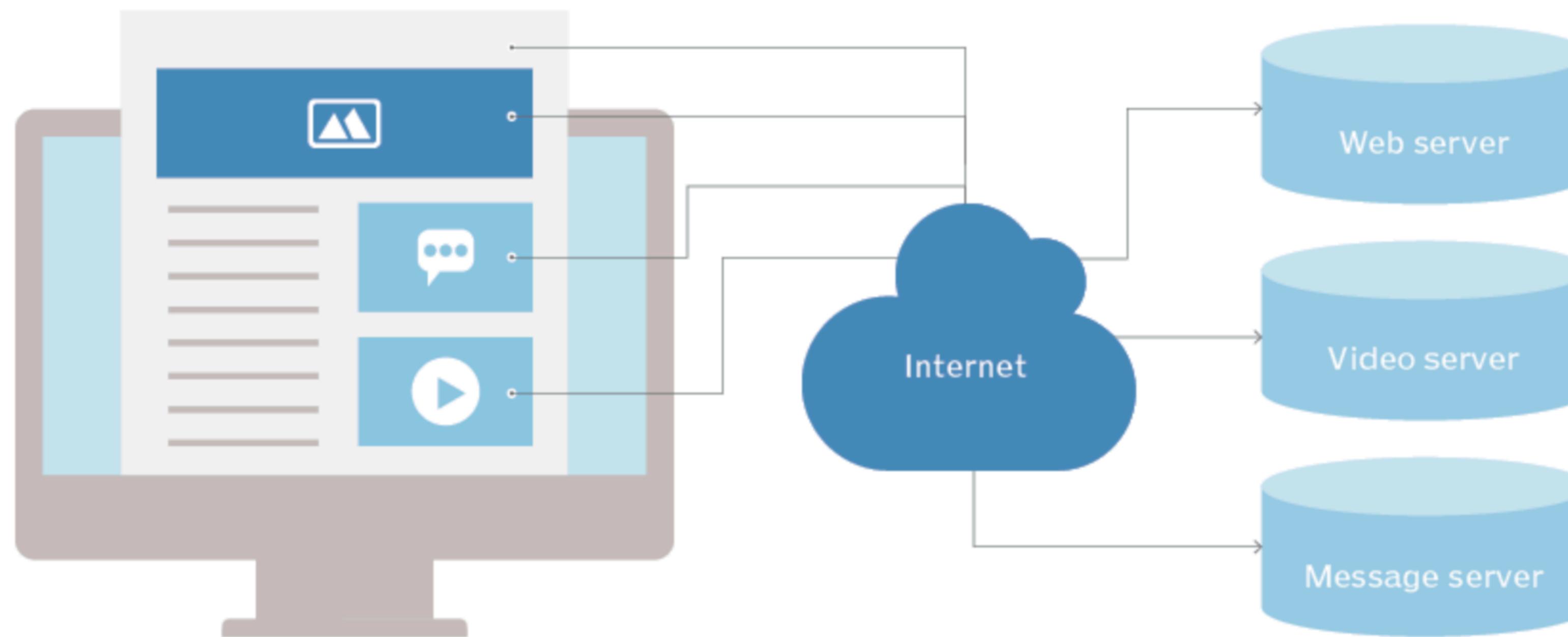
1. A client (a browser) sends an **HTTP request** to the web
2. A web server receives the request
3. The server runs an application to process the request
4. The server returns an **HTTP response** (output) to the browser
5. The client (the browser) receives the response

The HTTP Request Circle

A typical HTTP request / response circle:

1. The browser requests an HTML page. The server returns an HTML file.
2. The browser requests a style sheet. The server returns a CSS file.
3. The browser requests an JPG image. The server returns a JPG file.
4. The browser requests JavaScript code. The server returns a JS file
5. The browser requests data. The server returns data (in XML or JSON).

How HTTP Works



<https://www.techtarget.com/whatis/definition/HTTP-Hypertext-Transfer-Protocol>

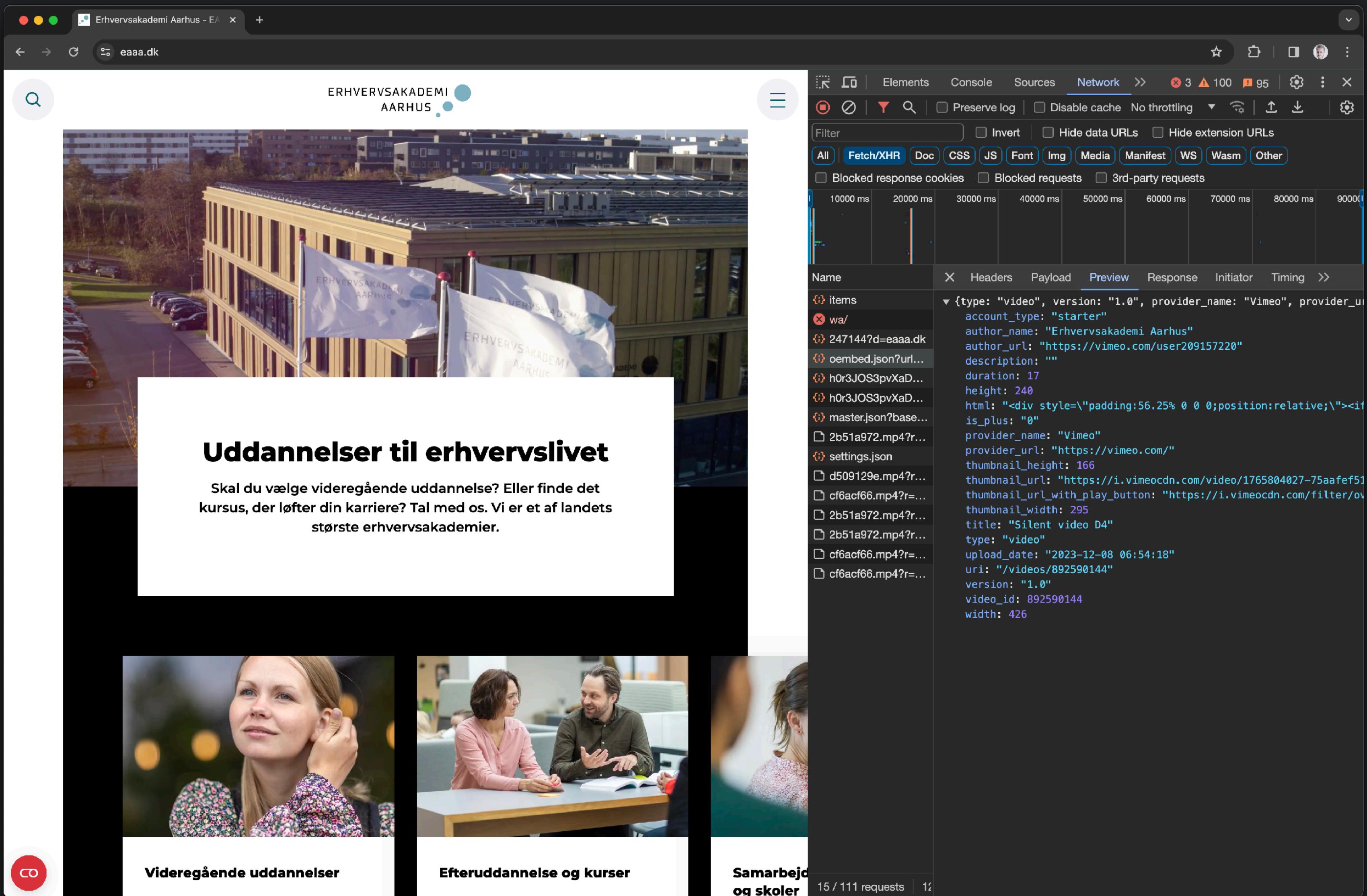
Network Tab

The screenshot illustrates the Network tab of a browser's developer tools, overlaid on a live website. The website's content includes a header for 'ERHVERVSAKADEMI AARHUS', a main image of two people working, and sections for 'Uddannelser til erhvervslivet' and 'Efteruddannelse og kurser'. The Network tab displays a timeline and a detailed list of network requests.

Name	Method	Status	Type	Initiator	Size	T..	Waterfall
1.gif?dgi=70fb8fd... GET 200 gif uc.js?cbid...							
heatmaps.js	GET	200	script	monsido-s...	(disk...	2...	
page-correct.js	GET	200	script	monsido-s...	(disk...	2...	
?a=h0r3JOS3pvXaDa...	GET	200	gif	monsido-s...	57 B	1...	
favicon.ico	GET	200	x-icon	Other	(disk...	1...	
h0r3JOS3pvXaDabSjt...	GET	200	xhr	heatmaps....	(disk...	0...	
h0r3JOS3pvXaDabSjt...	GET	200	xhr	page-corr...	(disk...	0...	
cast_sender.js?loadC...	GET	200	script	vendor.mo...	(disk...	0...	
1765804027-75aafef...	GET	200	avif	Other	(disk...	0...	
master.json?base64_i...	GET	200	xhr	vendor.mo...	3.0 kB	1...	
1765804027-75aafef...	GET	200	avif	vendor.mo...	(me...	0...	
cast_framework.js	GET	200	script	cast_send...	14 B	1...	
cast_sender.js	GET	200	script	cast_send...	(disk...	1...	
settings.json	GET	200	xhr	uc.js?cbid...	(disk...	0...	
widgetIcon.min.js	GET	200	script	uc.js?cbid...	(disk...	0...	
2b51a972.mp4?r=dX...	GET	200	xhr	vendor.mo...	5.4 kB	2...	
d509129e.mp4?r=dX...	GET	200	xhr	vendor.mo...	264 kB	3...	
cf6acf66.mp4?r=dXM...	GET	200	xhr	vendor.mo...	4.3 ...	2...	
2b51a972.mp4?r=dX...	GET	200	xhr	vendor.mo...	5.4 kB	1...	
2b51a972.mp4?r=dX...	GET	200	xhr	vendor.mo...	5.1 kB	2...	
cf6acf66.mp4?r=dXM...	GET	200	xhr	vendor.mo...	4.6 ...	2...	
cf6acf66.mp4?r=dXM...	GET	200	xhr	vendor.mo...	3.5 ...	3...	
collect?v=2&tid=G-D...	POST	204	ping	js?id=G-D...	17 B	6...	

91 requests | 12.8 MB transferred | 16.2 MB resources | Finish: 5.47 s | DOMContentLoaded: 290 ms

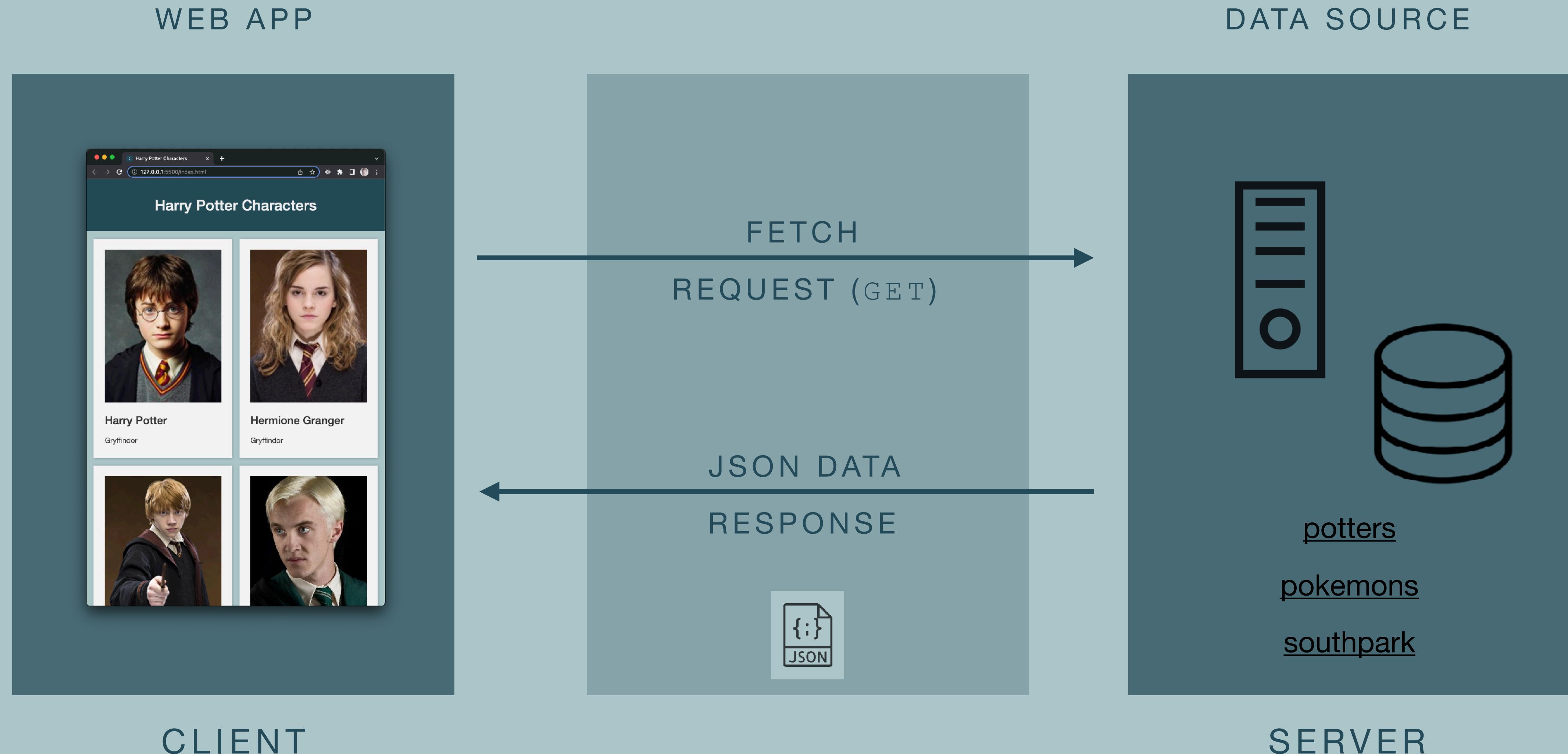
Network Tab



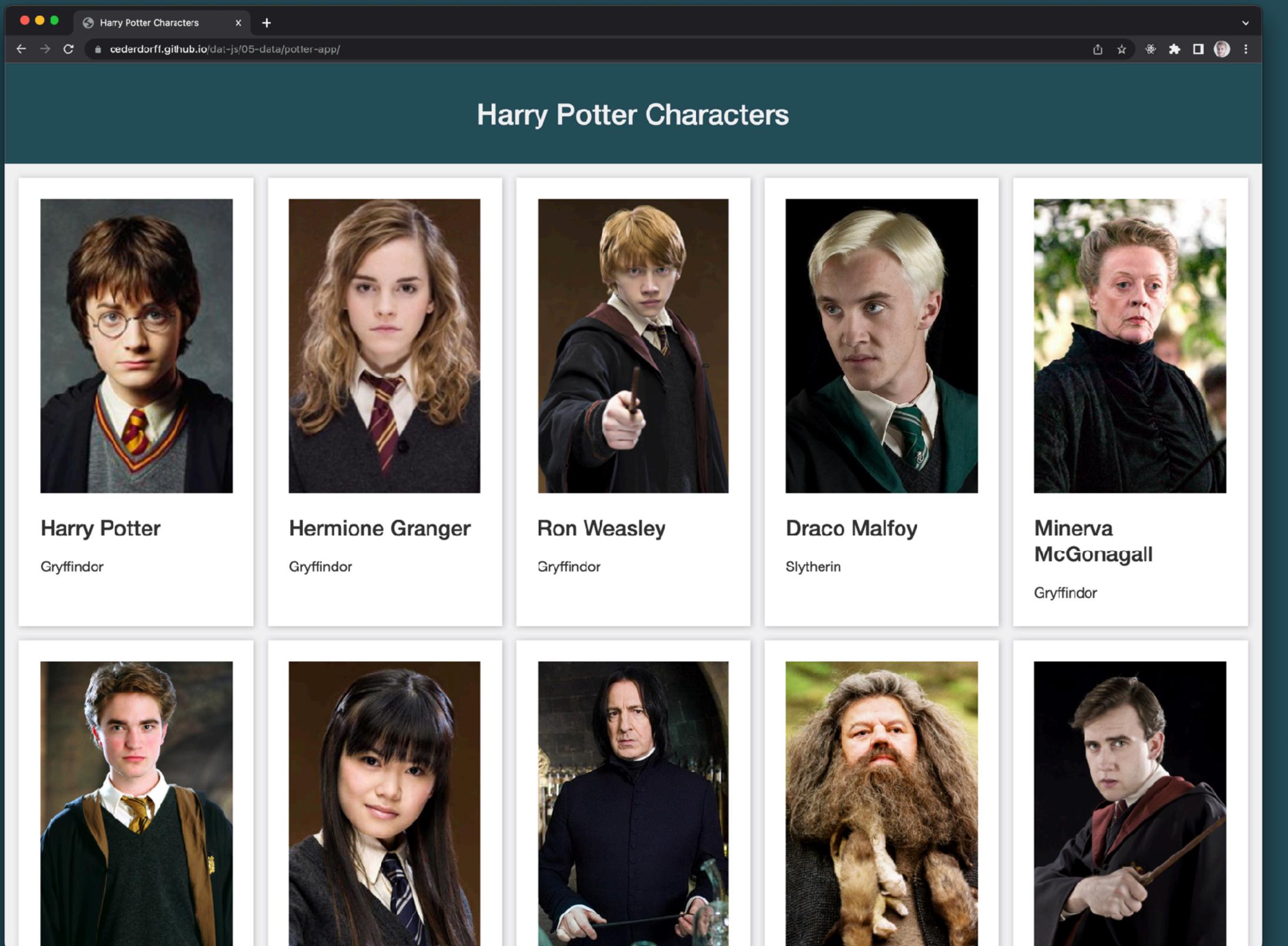
- Use the Network tab to investigate a website (e.g. kea.dk, dr.dk, google.dk, react-rest-and-auth.web.app or any other).
- Open the website in Google Chrome (or another browser).
- Open the Developer Tool (see: [How to open the dev tool in your browser](#)) and go to Netværk/Network.
- Reload the page while you are in the Network tab and check which resources are being downloaded.
- What type(s) of resources are they?
- Consider how the resources are being downloaded and why I tell you to do all this.

Network Tab

Fetch, HTTP Request & Response



Frontend (client)



JSON Data Source (Server)

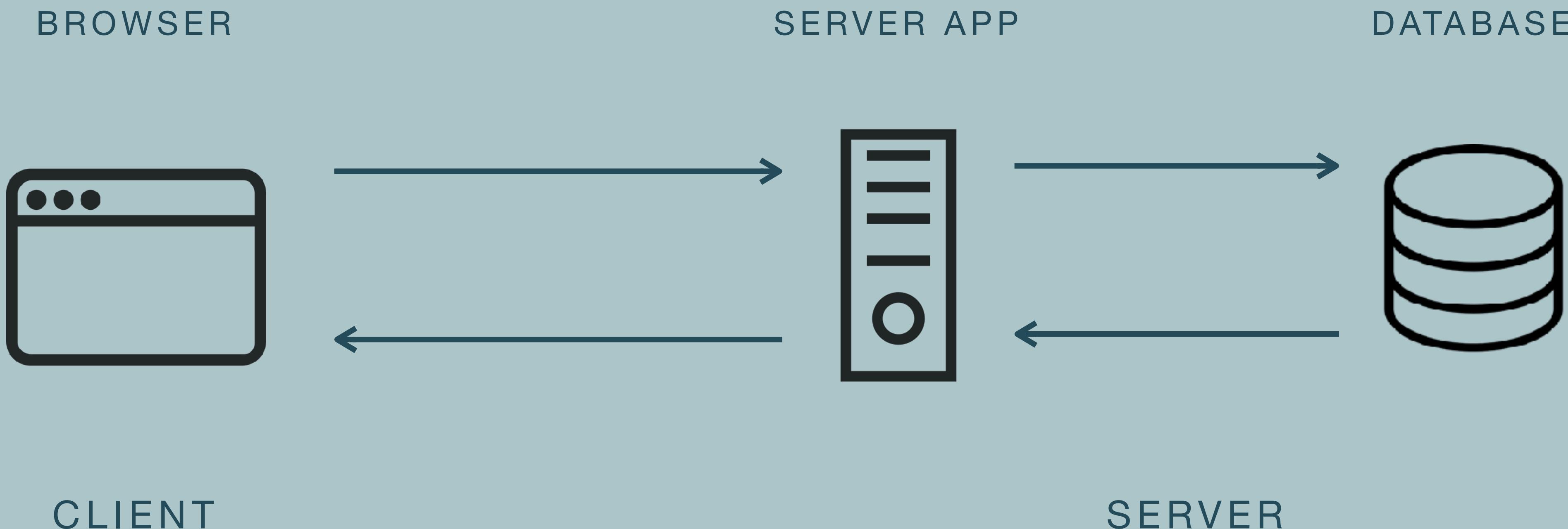
A screenshot of a browser window showing a JSON object representing a Harry Potter character. The JSON is displayed in two tabs: "Raw" and "Parsed". The "Parsed" tab shows the human-readable JSON, which includes details like name, species, gender, house, date of birth, year of birth, ancestry, eye color, hair color, and wand information. The "Raw" tab shows the raw JSON code.

```
Raw
{
  "name": "Harry Potter",
  "species": "human",
  "gender": "male",
  "house": "Gryffindor",
  "dateOfBirth": "31-07-1980",
  "yearOfBirth": 1980,
  "ancestry": "half-blood",
  "eyeColour": "green",
  "hairColour": "black",
  "wand": {
    "wood": "holly",
    "core": "phoenix feather",
    "length": 11
  },
  "patronus": "stag",
  "hogwartsStudent": true,
  "hogwartsStaff": false,
  "actor": "Daniel Radcliffe",
  "alive": true,
  "image": "http://hp-api.herokuapp.com/images/harry.jpg"
},
{
  "name": "Hermione Granger",
  "species": "human",
  "gender": "female",
  "house": "Gryffindor",
  "dateOfBirth": "19-09-1979",
  "yearOfBirth": 1979,
  "ancestry": "muggleborn",
  "eyeColour": "brown",
  "hairColour": "brown",
  "wand": {
    "wood": "vine",
    "core": "dragon heartstring",
    "length": ""
  },
  "patronus": "otter",
  "hogwartsStudent": true,
  "hogwartsStaff": false,
  "actor": "Emma Watson",
  "alive": true,
  "image": "http://hp-api.herokuapp.com/images/hermione.jpeg"
},
{
  "name": "Ron Weasley",
  "species": "human",
  "gender": "male",
  "house": "Gryffindor",
  "dateOfBirth": "01-03-1980",
  "yearOfBirth": 1980,
  "ancestry": "pure-blood",
  "eyeColour": "blue",
  "hairColour": "red",
  "wand": {
    "wood": "willow",
    "core": "unicorn hair",
    "length": 12
  }
}
```

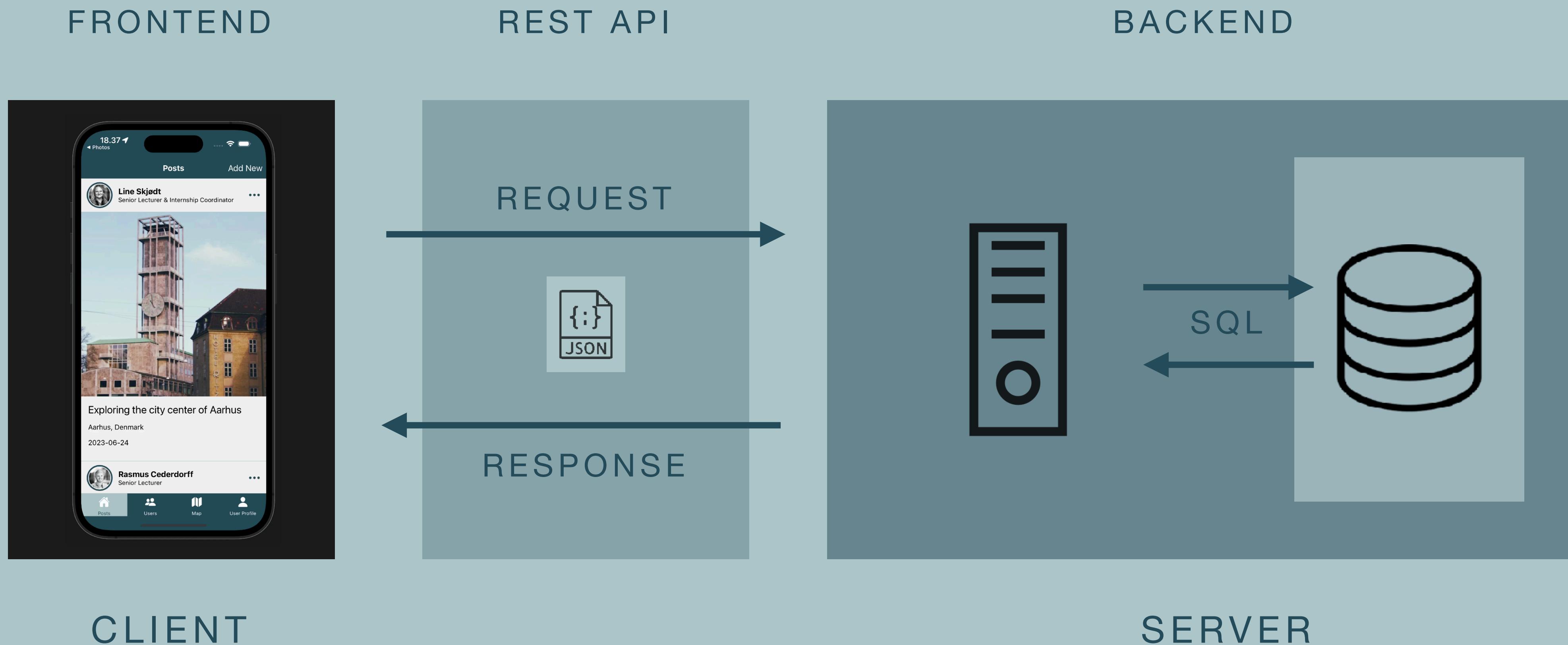
<https://cederdorff.com/potter-app/>

<https://raw.githubusercontent.com/cederdorff/dat-js/main/data/potter.json>

Web Dev Architecture



Web Dev Architecture



Payment Gateways

Social Media Integration

Healthcare Integration

Maps and Location Services

IoT (Internet of Things)

API

Weather Data

News and Content Aggregation

Financial Data

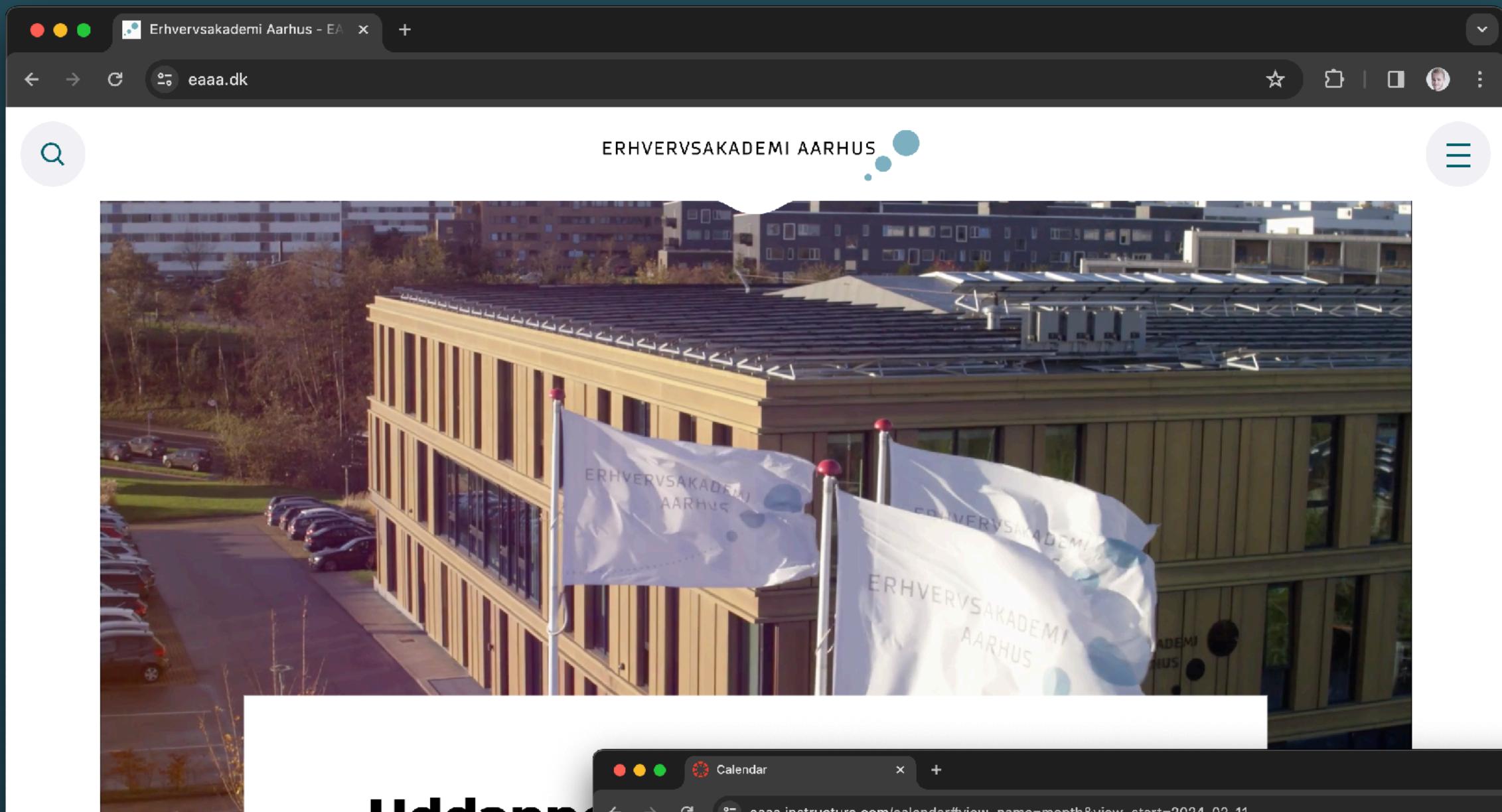
Cloud Services

E-commerce Platforms

The screenshot shows a web browser window titled "City Info App" with the URL "127.0.0.1:5500/index.html". The browser has a dark mode theme. On the left, there is a mobile application interface with a rounded rectangular background. It displays the title "Zip Code City App" and two input fields: "Zip Code:" with the value "8000" and "City:" with the value "Aarhus C". On the right, the browser's developer tools are open, showing the "Console" tab. The console output is a JSON object from "app.js:10" with the following structure:

```
app.js:10
{
  href: 'https://api.dataforsyningen.dk/postnumre/8000',
  nr: '8000',
  navn: 'Aarhus C',
  stormodtageradresser: null,
  bbox: Array(4),
  ...
}

bbox: [10.17165345, 56.11626813, 10.33170603, ...]
dagi_id: "198000"
geo_version: 3
geo_endret: "2023-03-24T22:28:38.837Z"
href: "https://api.dataforsyningen.dk/postnumre/8000"
kommuner: [...]
navn: "Aarhus C"
nr: "8000"
stormodtageradresser: null
visueltcenter: [10.27802097, 56.15051496]
endret: "2023-03-24T22:28:38.837Z"
[[Prototype]]: Object
```



A screenshot of a digital calendar interface for February 2024. The calendar is displayed in a grid format with days of the week (MON-SUN) and dates (29-Jan to 3-Mar). Each cell contains a small thumbnail of an event and its details. The sidebar on the left includes links for Account, Admin, Dashboard, Courses, Calendar (with a red notification dot), Inbox, History, Commons, and Student links.

A screenshot of a Google search results page for the query "house of vincent ring". The results are sponsored ads for various gold rings from the brand House of Vincent. The top result is a "TILBUD" for a "GUL CITRIN FORGYLDT. 699 kr. SUN GAZER RING GUL CITRIN FORGYLDT." ring. To the right of the search results is a map of Northern Europe with a blue dot indicating Denmark. Below the map is the brand's logo and some text: "HOUSE OF VINCENT", "Website", "Gem", "Åbningstider: Åben 24 timer", and "Foreslå en ændring".

Twitter API

Stripe API

Google Maps API

Microsoft HealthVault API

Fitbit API

API

OpenWeatherMap API

The New York Times API

Amazon S3 API

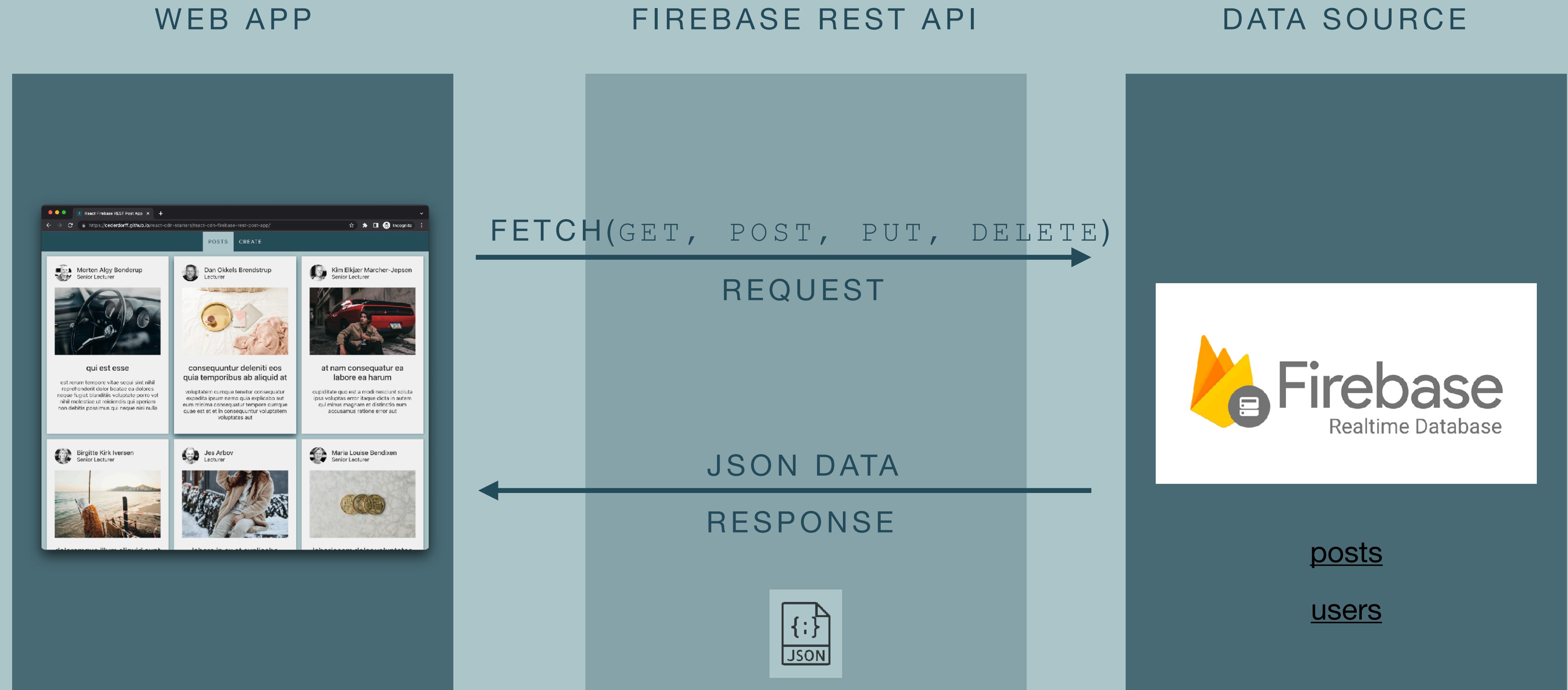
Alpha Vantage API

Shopify API

Fetch

HTTP Requests in JavaScript

Fetch, HTTP Request & Response



fetch(...)

HTTP Requests in
JavaScript.

A way to get and post data
from and to data sources.

```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/callback.js")
  .then(function (response) {
    return response.json();
  })
  .then(function (data) {
    console.log(data);
  });
// or with promises
const response = fetch("https://cederdorff.github.io/web-frontend/promise.js");
const data = response.json();
console.log(data);
```

fetch(...)

HTTP Requests in
JavaScript.

A way to get and post data
from and to data sources.

getCharacters fetches a list of characters
from a JSON data source, parses the JSON
to JS and returns the data.

```
async function getCharacters() {  
  const response = await fetch(url);  
  const data = await response.json();  
  console.log(data);  
  return data;  
}
```

fetch(...)

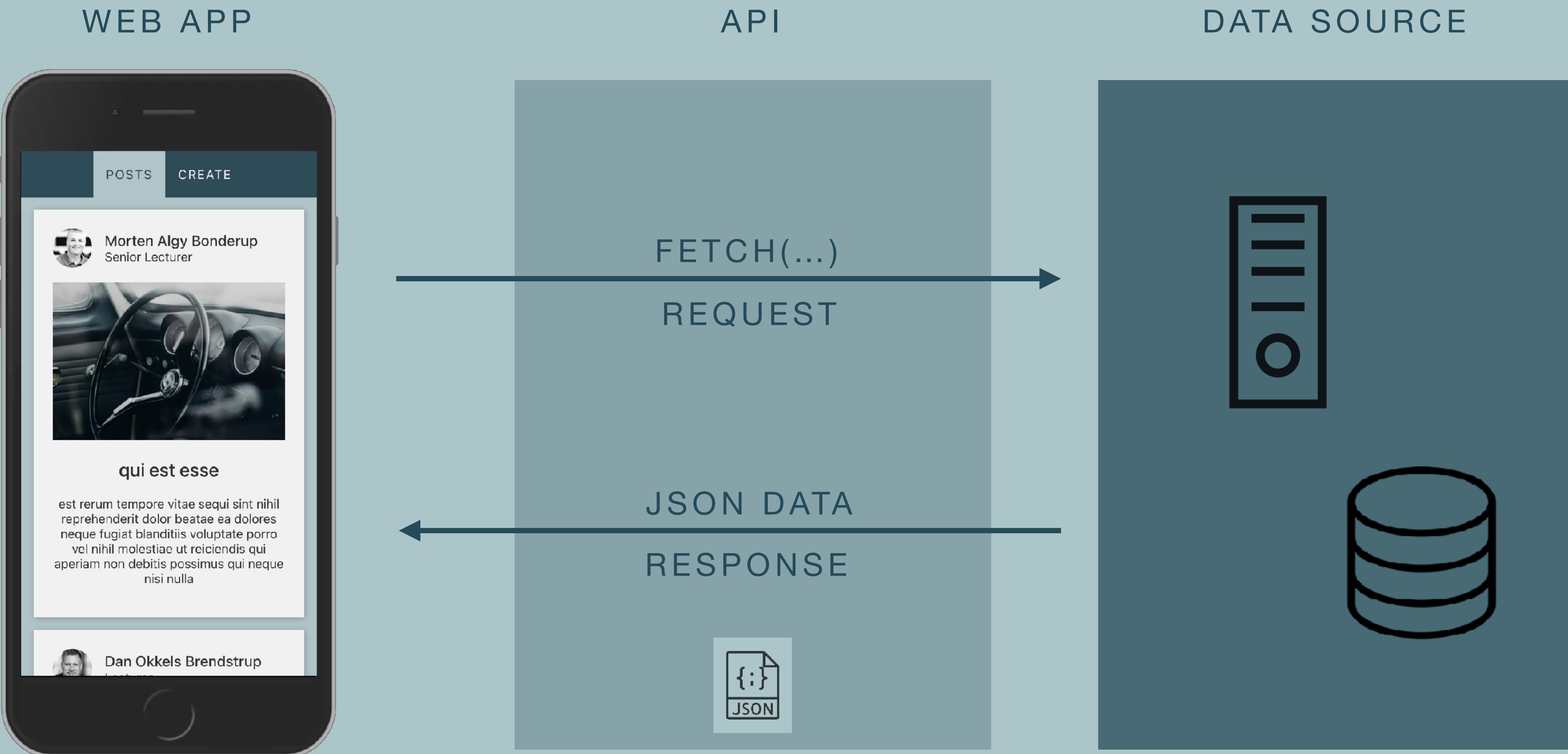
... get & post data from and to a data source
... can perform network requests to a server

```
1 let promise = fetch(url, [options])
```

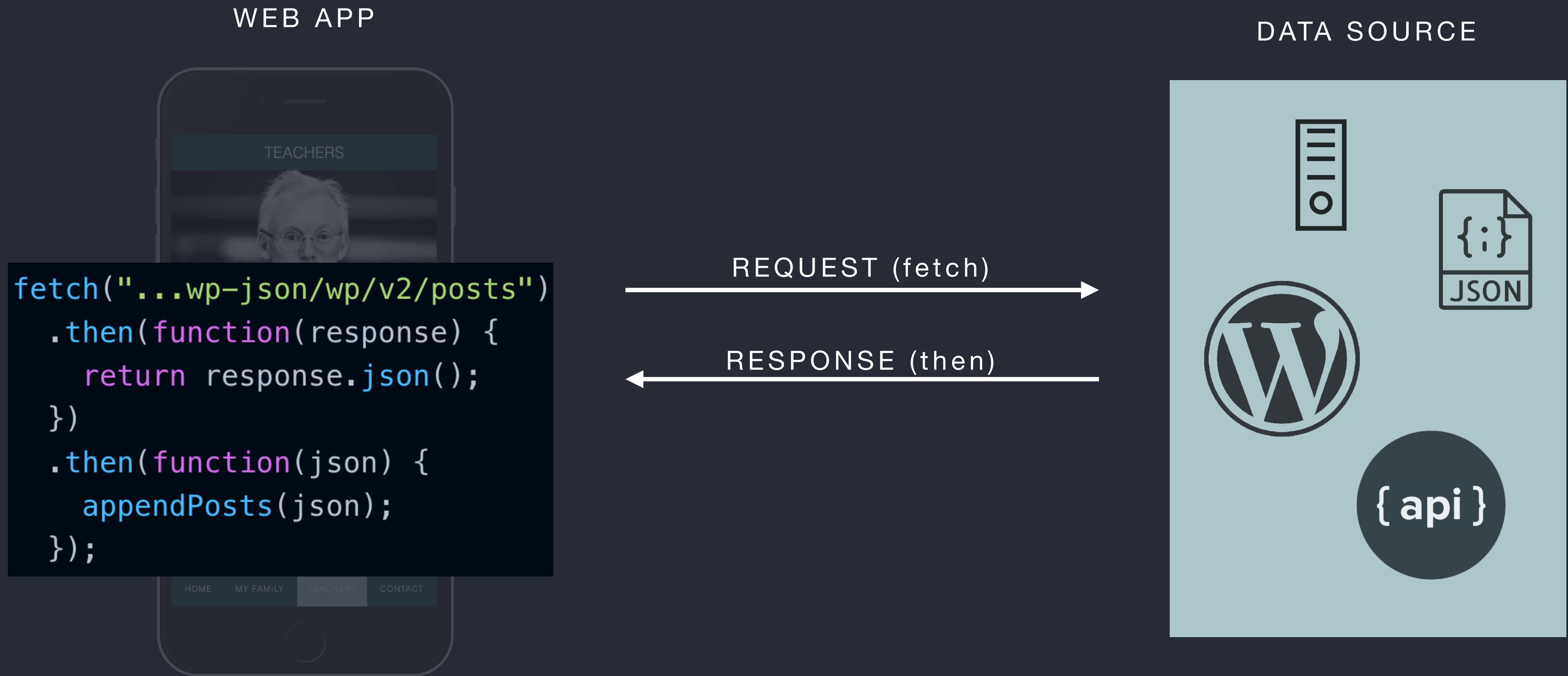
- `url` – the URL to access.
- `options` – optional parameters: method, headers etc.

Without `options`, this is a simple GET request, downloading the contents of the `url`.

Fetch, fetch, fetch



Fetch



```

/*
Fetches json data from the file persons.json
*/
fetch('json/persons.json')
  .then(function (response) {
    return response.json();
  })
  .then(function (jsonData) {
    console.log(jsonData);
    appendPersons(jsonData)
  });

/*
Appends json data to the DOM
*/
function appendPersons(persons) {
  let htmlTemplate = "";
  for (let person of persons) {
    htmlTemplate += /*html*/
      `


        <h4>${person.name}</h4>
        <p>${person.age} years old</p>
        <p>Hair color: ${person.hairColor}</p>
        <p>Relation: ${person.relation}</p>

`;
  }
  document.querySelector("#persons").innerHTML = htmlTemplate;
}

```

```

[
  {
    "name": "Peter Madsen",
    "age": 52,
    "hairColor": "blonde",
    "relation": "dad",
    "img": "img/dad.jpg"
  },
  {
    "name": "Ane Madsen",
    "age": 51,
    "hairColor": "brown",
    "relation": "mom",
    "img": "img/ane.jpg"
  },
  {
    "name": "Rasmus Madsen",
    "age": 28,
    "hairColor": "blonde",
    "relation": "brother",
    "img": "img/IMG_0526_kvadrat.jpg"
  },
  {
    "name": "Mie Madsen",
    "age": 25,
    "hairColor": "brown",
    "relation": "blonde",
    "img": "img/mie.jpg"
  },
  {
    "name": "Mads Madsen",
    "age": 18,
    "hairColor": "dark",
    "relation": "blonde",
    "img": "img/mads.jpg"
  },
  {
    "name": "Jens Madsen",
    "age": 14,
    "hairColor": "blonde",
    "relation": "uncle",
    "img": "img/jenspeter.jpg"
  }
]

```

```

/*
Fetches json data from the file persons.json
*/
fetch('json/persons.json')
  .then(function (response) {
    return response.json();
  })
  .then(function (jsonData) {
    console.log(jsonData);
    appendPersons(jsonData);
  });

/*
Appends json data to the DOM
*/
function appendPersons(persons) {
  let htmlTemplate = "";
  for (let person of persons) {
    htmlTemplate += /*html*/
      <article>
        
        <h4>${person.name}</h4>
        <p>${person.age} years old</p>
        <p>Hair color: ${person.hairColor}</p>
        <p>Relation: ${person.relation}</p>
      </article>
  }
  document.querySelector("#persons").innerHTML = htmlTemplate;
}

```

REQUEST (fetch)

RESPONSE (then)

```
[
  {
    "name": "Peter Madsen",
    "age": 52,
    "hairColor": "blonde",
    "relation": "dad",
    "img": "img/dad.jpg"
  },
  {
    "name": "Ane Madsen",
    "age": 51,
    "hairColor": "brown",
    "relation": "mom",
    "img": "img/ane.jpg"
  },
  {
    "name": "Rasmus Madsen",
    "age": 28,
    "hairColor": "blonde",
    "relation": "brother",
    "img": "img/IMG_0526_kvadrat.jpg"
  },
  {
    "name": "Mie Madsen",
    "age": 25,
    "hairColor": "brown",
    "relation": "blonde",
    "img": "img/mie.jpg"
  },
  {
    "name": "Mads Madsen",
    "age": 18,
    "hairColor": "dark",
    "relation": "blonde",
    "img": "img/mads.jpg"
  },
  {
    "name": "Jens Madsen",
    "age": 14,
    "hairColor": "blonde",
    "relation": "uncle",
    "img": "img/jenspeter.jpg"
  }
]
```

```
/*
Fetches json data from the file persons.json
*/
fetch('json/persons.json')
  .then(function (response) {
    return response.json();
  })
  .then(function (jsonData) {
    console.log(jsonData);
    appendPersons(jsonData)
  });

/*
Appends json data to the DOM
*/
function appendPersons(persons) {
  let htmlTemplate = "";
  for (let person of persons) {
    htmlTemplate += /*html*/
      `


        <h4>${person.name}</h4>
        <p>${person.age} years old</p>
        <p>Hair color: ${person.hairColor}</p>
        <p>Relation: ${person.relation}</p>

`;
  }
  document.querySelector("#persons").innerHTML = htmlTemplate;
}
```

```
},
{
  "name": "Rasmus Madsen",
  "age": 28,
  "hairColor": "blonde",
  "relation": "brother",
  "img": "img/IMG_0526_kvadrat.jpg"
},
{
  "name": "Mie Madsen",
  "age": 25,
  "hairColor": "brown",
  "relation": "blonde",
  "img": "img/mie.jpg"
},
{
  "name": "Mads Madsen",
  "age": 18,
  "hairColor": "dark",
  "relation": "blonde",
  "img": "img/mads.jpg"
},
{
```

request (fetch)

The diagram illustrates the flow of data from the `app.js` code to the resulting JSON response. A curved arrow originates from the URL in the `fetch` statement in the `getPerson` function and points to the JSON data on the right. Another curved arrow originates from the `persons` variable in the `displayPersons` function and points back to the `app.js` code.

```
app.js — web-diplom-frontend
JS app.js ×
fetch-persons-grid > JS app.js > ...
1  let persons = [] // global variable
2
3  async function getPerson() {
4      const response = await fetch(
5          "https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json"
6      ); // fetch request - fetch data from a given url
7      persons = await response.json(); // setting global variable with fetched data
8      displayPersons(persons); // calling displayPersons with persons as parameter
9  }
10
11 function displayPersons(listOfPersons) {
12     let html = ""; // variable to store html
13     //loop through all persons and create an article with content for each
14     for (const person of listOfPersons) {
15         html += /*html*/
16             `

17                 
18                 <h2>${person.name}</h2>
19                 <p>${person.title}</p>
20                 <a href="mailto:${person.mail}">${person.mail}</a>
21             </article>
22         `; // generate and save html for every person in html variable
23     }
24     // set grid container content with person <article> elements
25     // saved in html
26     document.querySelector("#content").innerHTML = html;
27 }
28
29 getPerson(); // execute get persons to start the fun
30


```

Ln 1, Col 1 Spaces: 4 UTF-8 LF {} JavaScript Go Live ✓ Prettier

fetch-persons-grid

A screenshot of a browser window showing the JSON data received from the URL. The URL is `https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json`. The JSON array contains eight objects, each representing a person with properties: name, mail, title, and img.

```
[{"name": "Birgitte Kirk Iversen", "mail": "bkj@mail.dk", "title": "Senior Lecturer", "img": "https://www.eaaa.dk/media/u4gorzsd/birgitte-kirk-iversen2.jpg?width=800&height=450"}, {"name": "Martin Aagaard N\u00f8hr", "mail": "mnor@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/oayjq02h/martin-n%C3%B8hr.jpg?width=800&height=450"}, {"name": "Rasmus Cederdorff", "mail": "race@mail.dk", "title": "Senior Lecturer", "img": "https://www.eaaa.dk/media/devlvgj/rasmus-cederdorff.jpg?width=800&height=450"}, {"name": "Dan Okkels Brendstrup", "mail": "dob@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/bdojel41/dan-okkels-brendstrup.jpg?width=800&height=450"}, {"name": "Line Skj\u00f8dt", "mail": "lskj@mail.dk", "title": "Senior Lecturer & Internship Coordinator", "img": "https://www.eaaa.dk/media/14qpfq4/line-skj%C3%B8dt.jpg?width=800&height=450"}, {"name": "Kasper Fischer Topp", "mail": "kato@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/lxzcybme/kasper-topp.jpg?width=800&height=450"}, {"name": "Anne Kirketerp", "mail": "anki@mail.dk", "title": "", "img": ""}]
```

request (fetch)

The diagram illustrates the flow of data from the `app.js` code to the resulting JSON response. A curved arrow labeled "request (fetch)" points from the `fetch` call in the `getPerson` function to the browser screenshot. Another curved arrow labeled "response (JSON)" points from the `response.json` call to the JSON data shown in the browser.

```
app.js — web-diplom-frontend
JS app.js ×
fetch-persons-grid > JS app.js > ...
1 let persons = [] // global variable
2
3 async function getPerson() {
4     const response = await fetch(
5         "https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json"
6     ); // fetch request - fetch data from a given url
7     persons = await response.json(); // setting global variable with fetched data
8     displayPersons(persons); // calling displayPersons with persons as parameter
9 }
10
11 function displayPersons(listOfPersons) {
12     let html = ""; // variable to store html
13     //loop through all persons and create an article with content for each
14     for (const person of listOfPersons) {
15         html += /*html*/
16             `

17                 
18                 <h2>${person.name}</h2>
19                 <p>${person.title}</p>
20                 <a href="mailto:${person.mail}">${person.mail}</a>
21             </article>
22         `; // generate and save html for every person in html variable
23     }
24     // set grid container content with person <article> elements
25     // saved in html
26     document.querySelector("#content").innerHTML = html;
27 }
28
29 getPerson(); // execute get persons to start the fun
30


```

Ln 1, Col 1 Spaces: 4 UTF-8 LF {} JavaScript Go Live ✓ Prettier

A screenshot of a browser window showing the JSON response from the URL `https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json`. The browser title bar shows the URL and the page content area displays the JSON data.

```
[{"name": "Birgitte Kirk Iversen", "mail": "bkj@mail.dk", "title": "Senior Lecturer", "img": "https://www.eaaa.dk/media/u4gorzsd/birgitte-kirk-iversen2.jpg?width=800&height=450"}, {"name": "Martin Aagaard N\u00f8hr", "mail": "mnor@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/oayjq02h/martin-n%C3%B8hr.jpg?width=800&height=450"}, {"name": "Rasmus Cederdorff", "mail": "race@mail.dk", "title": "Senior Lecturer", "img": "https://www.eaaa.dk/media/devlvgj/rasmus-cederdorff.jpg?width=800&height=450"}, {"name": "Dan Okkels Brendstrup", "mail": "dob@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/bdojel41/dan-okkels-brendstrup.jpg?width=800&height=450"}, {"name": "Line Skj\u00f8dt", "mail": "lskj@mail.dk", "title": "Senior Lecturer & Internship Coordinator", "img": "https://www.eaaa.dk/media/14qpfeq4/line-skj%C3%B8dt.jpg?width=800&height=450"}, {"name": "Kasper Fischer Topp", "mail": "kato@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/lxzcybme/kasper-topp.jpg?width=800&height=450"}, {"name": "Anne Kirketerp", "mail": "anki@mail.dk", "title": "", "img": ""}]
```

fetch-persons-grid

The diagram illustrates the data flow in a web application. On the left, a screenshot of a code editor shows the file `app.js` with the following code:

```
1 let persons = [] // global variable
2
3 async function getPerson() {
4     const response = await fetch(
5         "https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json"
6     ); // fetch request - fetch data from a given url
7     persons = await response.json(); // setting global variable with fetched data
8     displayPersons(persons); // calling displayPersons with persons as parameter
9 }
10
11 function displayPersons(listOfPersons) {
12     let html = ""; // variable to store html
13     //loop through all persons and create an article with content for each
14     for (const person of listOfPersons) {
15         html += /*html*/
16             `

17                 
18                 <h2>${person.name}</h2>
19                 <p>${person.title}</p>
20                 <a href="mailto:${person.mail}">${person.mail}</a>
21             </article>
22         `; // generate and save html for every person in html variable
23     }
24     // set grid container content with person <article> elements
25     // saved in html
26     document.querySelector("#content").innerHTML = html;
27 }
28
29 getPerson(); // execute get persons to start the fun
30


```

On the right, a screenshot of a browser window shows the JSON data source at `https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json`. The JSON array contains the following objects:

```
[{"name": "Birgitte Kirk Iversen", "mail": "bki@mail.dk", "title": "Senior Lecturer", "img": "https://www.eaaa.dk/media/u4gorzsd/birgitte-kirk-iversen2.jpg?width=800&height=450"}, {"name": "Martin Aagaard N\u00f8hr", "mail": "mnor@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/oayjq02h/martin-n%C3%B8hr.jpg?width=800&height=450"}, {"name": "Rasmus Cederdorff", "mail": "race@mail.dk", "title": "Senior Lecturer", "img": "https://www.eaaa.dk/media/devlvgj/rasmus-cederdorff.jpg?width=800&height=450"}, {"name": "Dan Okkels Brendstrup", "mail": "dob@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/bdojel41/dan-okkels-brendstrup.jpg?width=800&height=450"}, {"name": "Line Skj\u00f8dt", "mail": "lskj@mail.dk", "title": "Senior Lecturer & Internship Coordinator", "img": "https://www.eaaa.dk/media/14qpfq4/line-skj%C3%B8dt.jpg?width=800&height=450"}, {"name": "Kasper Fischer Topp", "mail": "kato@mail.dk", "title": "Lecturer", "img": "https://www.eaaa.dk/media/lxzcybme/kasper-topp.jpg?width=800&height=450"}, {"name": "Anne Kirketerp", "mail": "anki@mail.dk", "title": "", "img": ""}]
```

fetch-persons-grid

index.html — web-diplom-frontend

JS app.js

```
fetch-persons-grid > JS app.js > ...
1 let persons = [] // global variable
2
3 async function getPerson() {
4     const response = await fetch(
5         "https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json"
6     ); // fetch request - fetch data from a given url
7     persons = await response.json(); // setting global variable with fetched data
8     displayPersons(persons); // calling displayPersons with persons as parameter
9 }
10
11 function displayPersons(listOfPersons) {
12     let html = ""; // variable to store html
13     //loop through all persons and create an article with content for each
14     for (const person of listOfPersons) {
15         html += /*html*/ `
16             <article>
17                 
18                 <h2>${person.name}</h2>
19                 <p>${person.title}</p>
20                 <a href="mailto:${person.mail}">${person.mail}</a>
21             </article>
22         `; // generate and save html for every person in html variable
23     }
24     // set grid container content with person <article> elements
25     // saved in html
26     document.querySelector("#content").innerHTML = html;
27 }
28
29 getPerson(); // execute get persons to start the fun
30
```

index.html

```
fetch-persons-grid > index.html > ...
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7     <meta name="viewport" content="width=device-width, initial-scale=1.0">
8     <link rel="stylesheet" href="app.css">
9     <title>Fetch Persons</title>
10
11 </head>
12 <body>
13     <header>
14         <h1>Fetch Persons</h1>
15     </header>
16     <main>
17         <section id="content" class="grid-container"></section>
18     </main>
19     <script src="app.js"></script>
20
21 </body>
22 </html>
```

DOM Manipulation

Ln 17, Col 9 Spaces: 4 UTF-8 LF HTML ⚡ Go Live ✅ Prettier

fetch-persons-grid

Fetch

... get & post data from and to a data source

```
// Simple javascript 😞  
  
//Synchronous fetch using async/await.  
  
// Usual way  
✓ const jsonData = fetch('URL')  
    .then(response => response.json())  
    .then(json => console.log(json));  
  
// Using await  
✓ const jsonData = await fetch('URL').then(res => res.json())  
  
// Shorter syntax 😊  
✓ const jsonData = await (await fetch('URL')).json();
```

<https://www.instagram.com/p/B0nxQjXj9Zi/>

Async JS

JavaScript reads and runs the script from top to bottom.

JavaScript functions are executed in the sequence they are called. Not in the sequence they are defined.

... by default JavaScript is synchronous.

JS is Synchronous & Single-Threaded

```
function myFirst() {  
  console.log("Hello");  
}
```

```
function mySecond() {  
  console.log("Goodbye");  
}
```

```
mySecond();  
myFirst();
```

```
/* ----- Global Variables ----- */
let _users = [];
let _selectedUserId;

/* ----- */

async function fetchUsers() { ... }
function appendUsers(usersArray) { ... }
}

// ===== INIT APP =====

async function initApp() {
    await fetchUsers();
    appendUsers(_users);
}

initApp();
```

With callbacks, we can make JS Asynchronous

```
setTimeout(() => {
  console.log("Hey, I'm async!");
}, 3000);

btn.addEventListener('click', () => {
  alert("Hey, you clicked me!");
});
```

```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/canvas")
  .then(function (response) {
    return response.json();
  })
  .then(function (data) {
    console.log(data);
  });
});
```

Fetch is Asynchronous

And it's about making HTTP requests in JavaScript.
... and a way to get & post data from and to a data source.

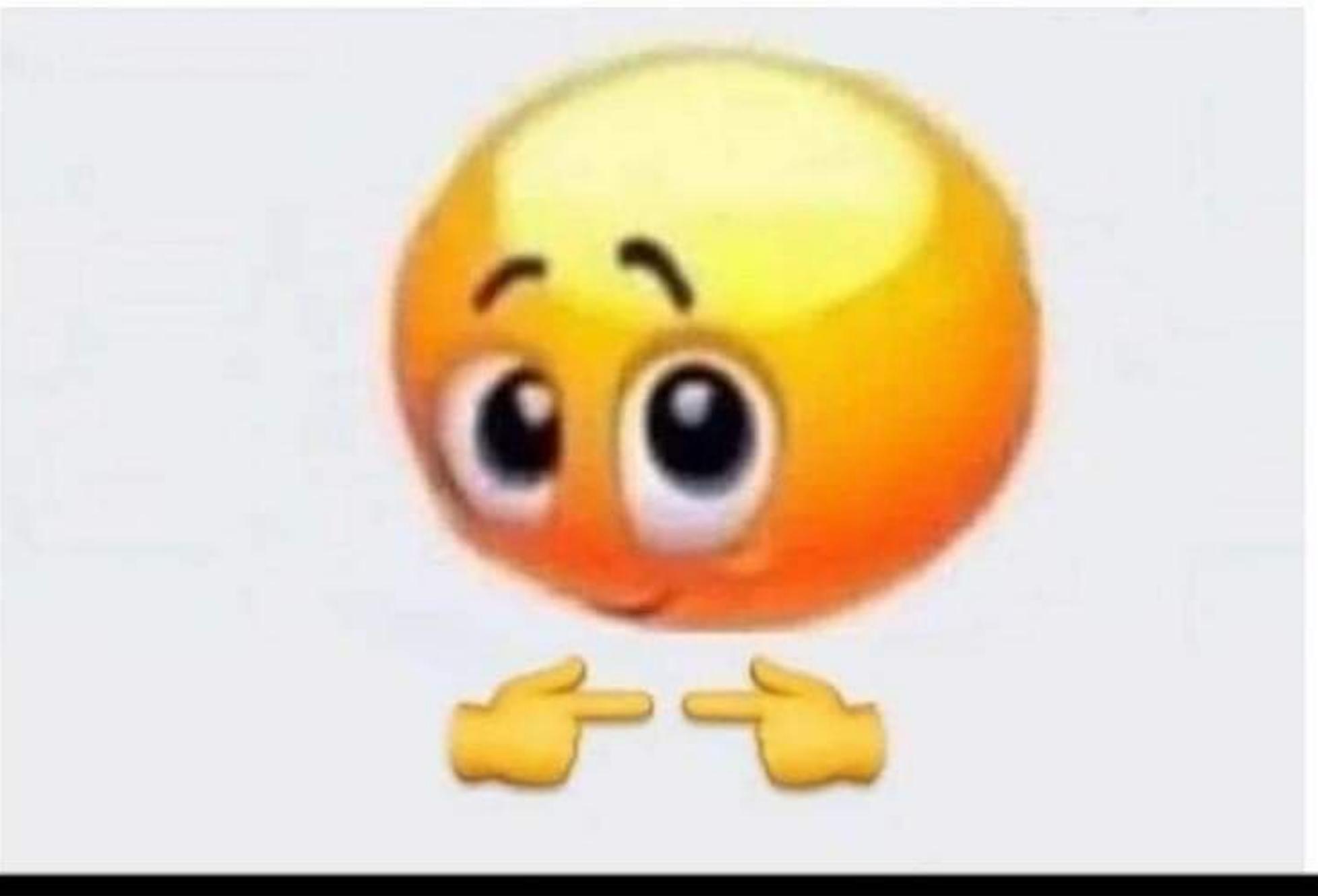
```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json")
  .then(function (response) {
    return response.json();
  })
  .then(function (data) {
    console.log(data);
  });

```

Fetch: callback (then) vs async/await

```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json")
  .then(function (response) {
    return response.json();
  })
  .then(function (data) {
    console.log(data);
  });
// 
// 
// or with async/await
const response = await fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json");
const data = await response.json();
console.log(data);
```

will you be async to my await?



async & await

- Use await to tell JS to wait for a fetch call to finish and to wait for JSON to parse.
- When using await you must tell JS that inside of the function goes some asynchronous code by wrapping it in an async function.

```
async function getPosts() {  
  const url = "https://raw.githubusercontent.com/.../data.json";  
  const response = await fetch(url);  
  const data = await response.json();  
  setPosts(data);  
}  
  
getPosts();
```

**"async and await makes promises
easier to write"**

async makes a function return a Promise

await makes a function wait for a Promise

https://www.w3schools.com/js/js_async.asp

Fetch returns a promise

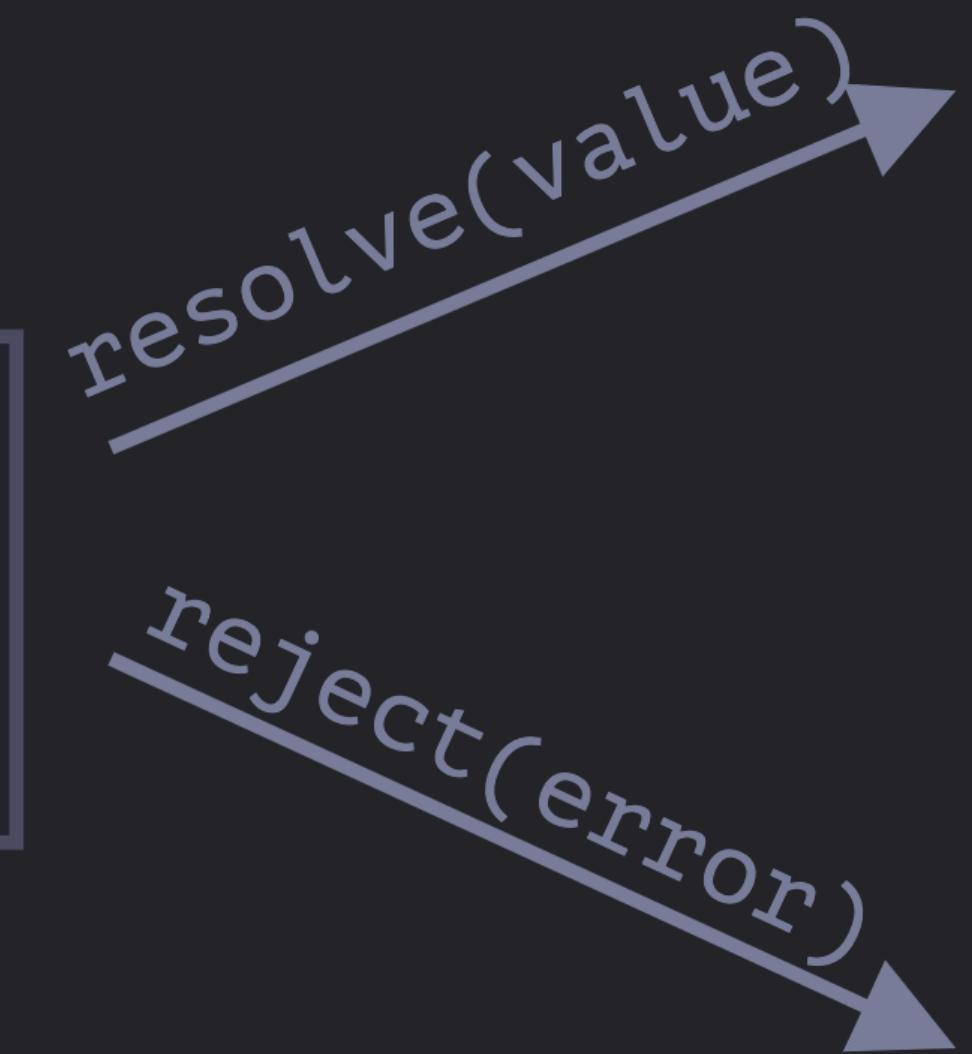
We can use `await` to wait for `fetch` to finish



https://www.w3schools.com/js/js_async.asp

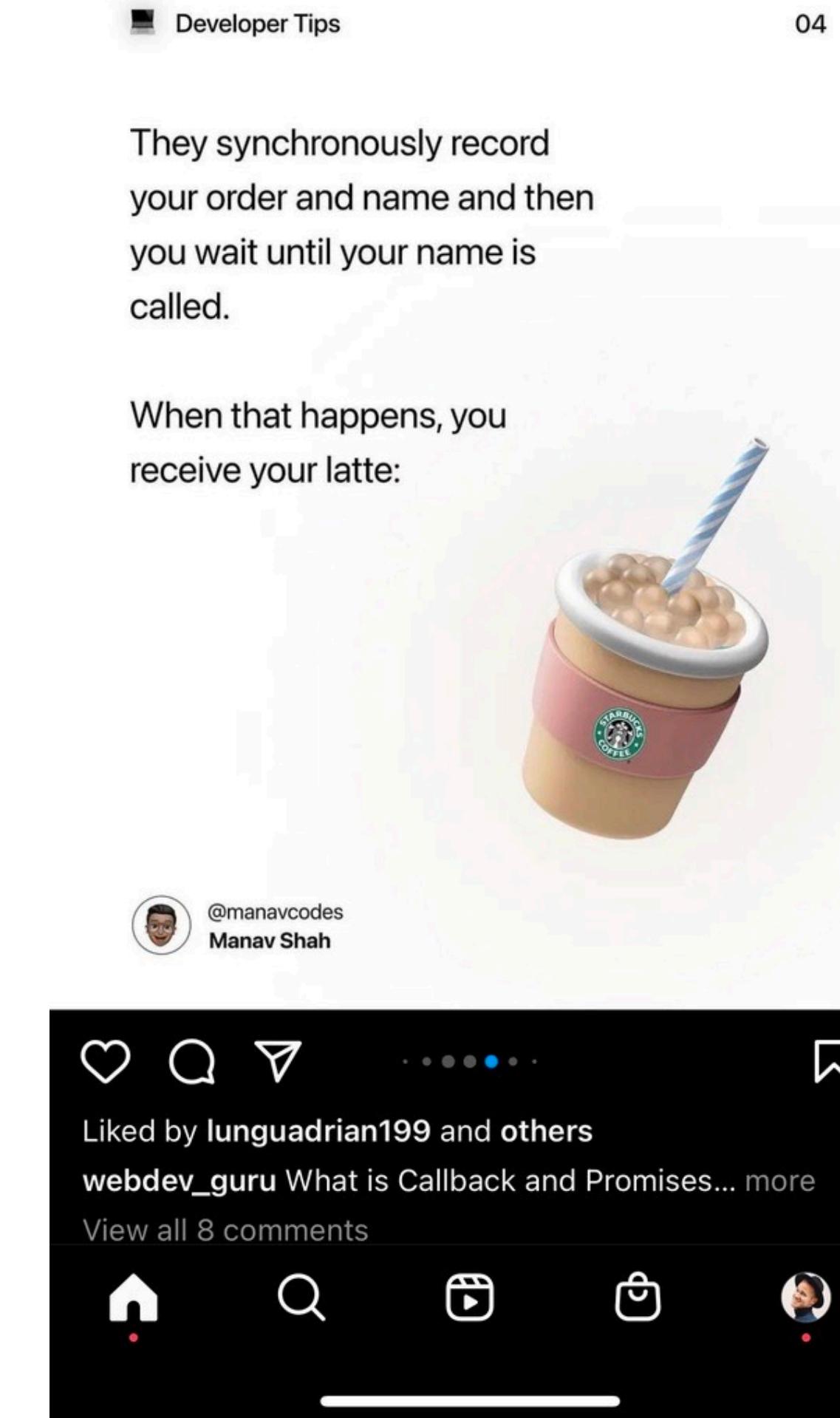
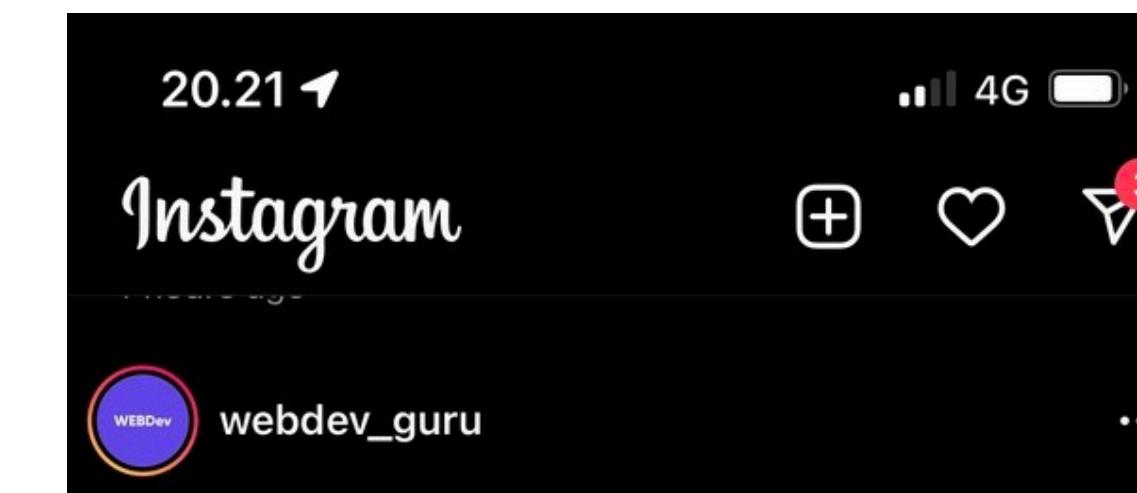
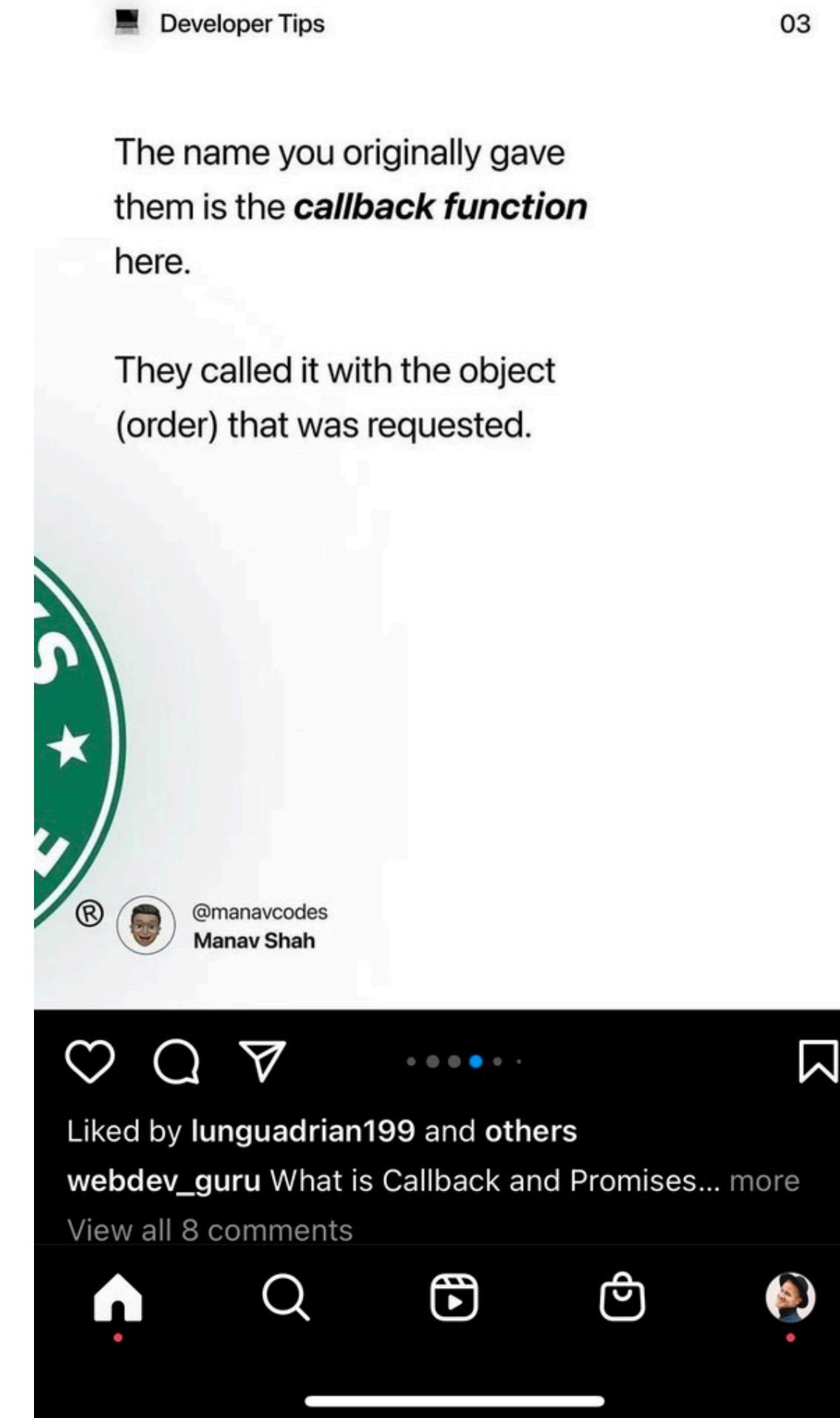
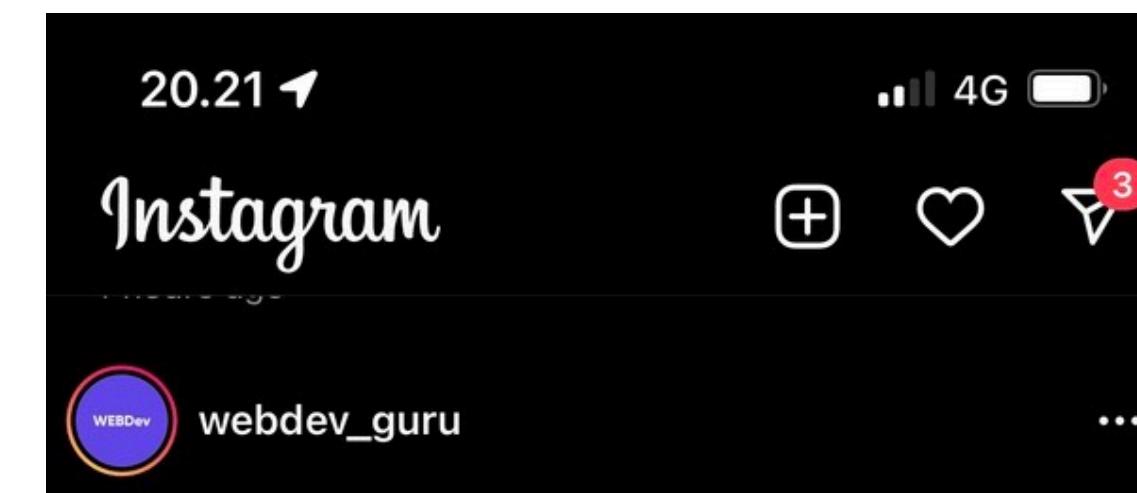
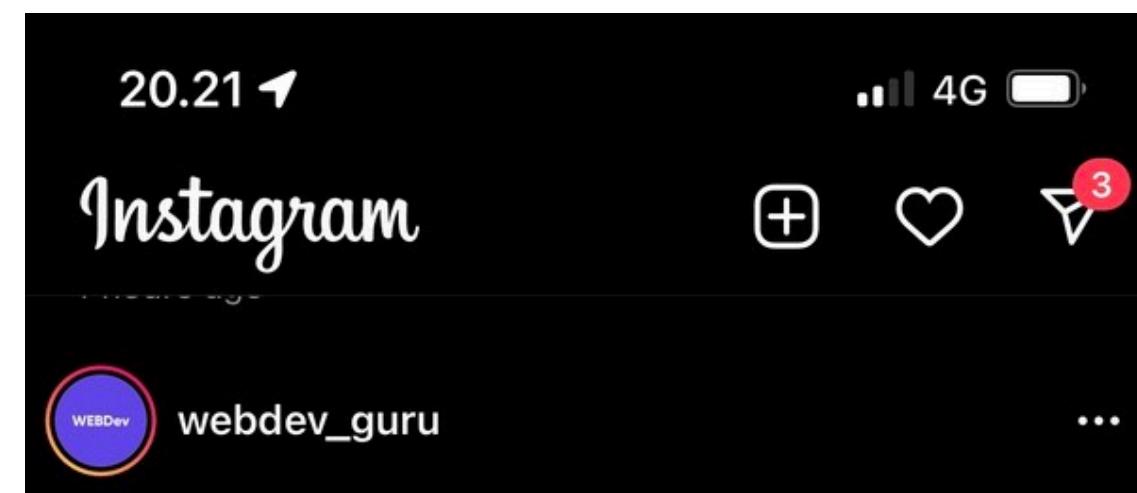
```
new Promise(executor)
```

state: "pending"
result: undefined



state: "fulfilled"
result: value

state: "rejected"
result: error



```
let myPromise = new Promise(function (resolve, reject) {
    // "Producing Code" (May take some time)
    // Making coffee ...
    resolve("Yaaay, here's your coffee"); // when successful
    reject("Ohh, f***. Something went wrong"); // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
    function (value) {
        /* code if successful */
        console.log(value); // Yaaay, here's your coffee
    },
    function (error) {
        /* code if some error */
        console.log(error); // Ohh, f***. Something went wrong
    }
);
```

All you need to know

- Use await to tell JS to wait for a fetch call to finish.
- When using await you must tell JS that here goes some asynchronous code by wrapping it in an async function.

```
function PostsPage() {
  const [posts, setPosts] = useState([]);

  useEffect(() => {
    async function getPosts() {
      const url = "https://raw.githubusercontent.com";
      const response = await fetch(url);
      const data = await response.json();
      setPosts(data);
    }
    getPosts();
  }, []);

  return (
    <section className="page">
      <h1>Posts</h1>
      <section className="grid-container">
        {posts.map(post => (
          <PostItem post={post} key={post.id} />
        ))}
    </section>
  );
}
```

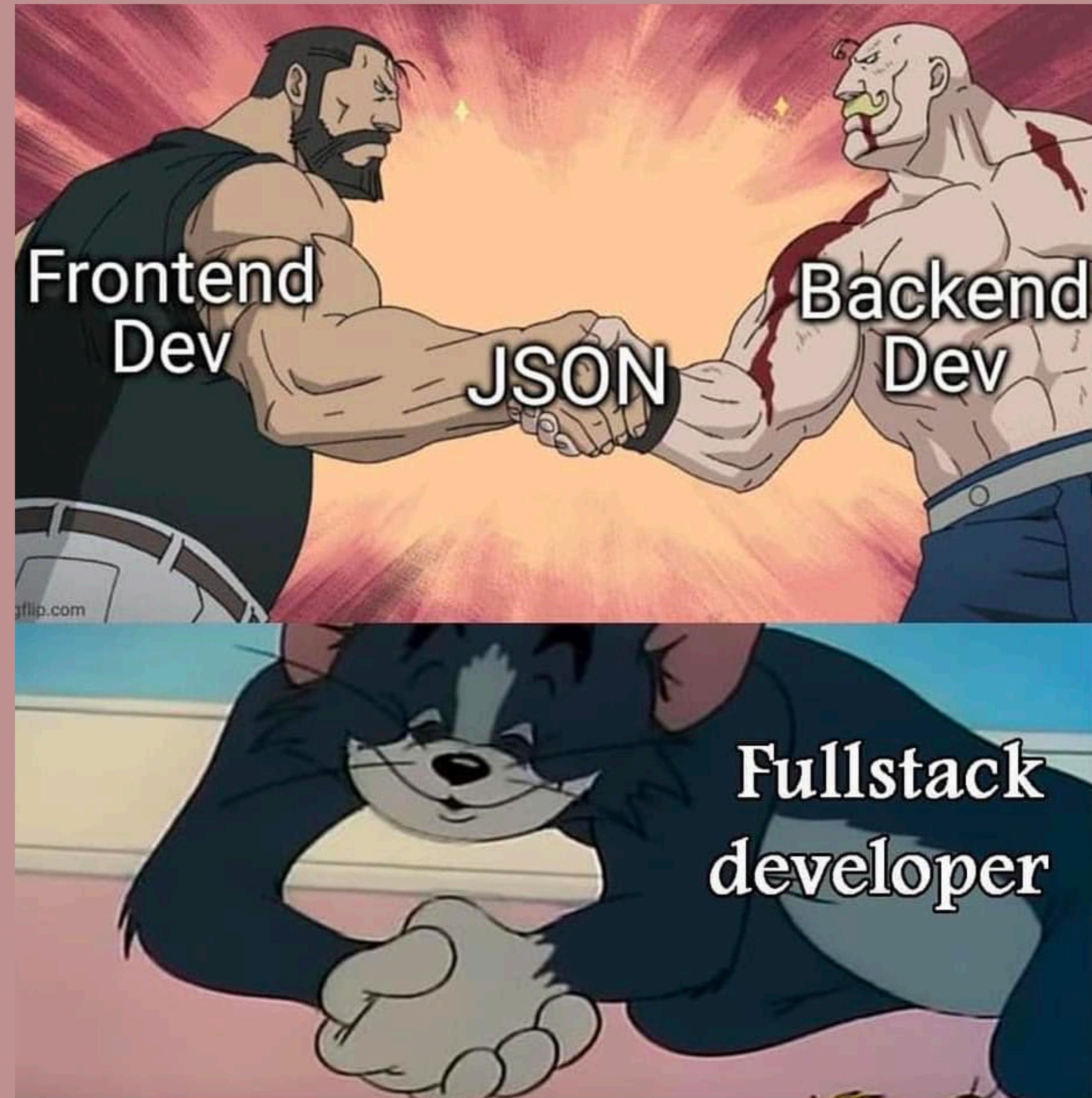
JSON

JavaScript Object Notation

JSON

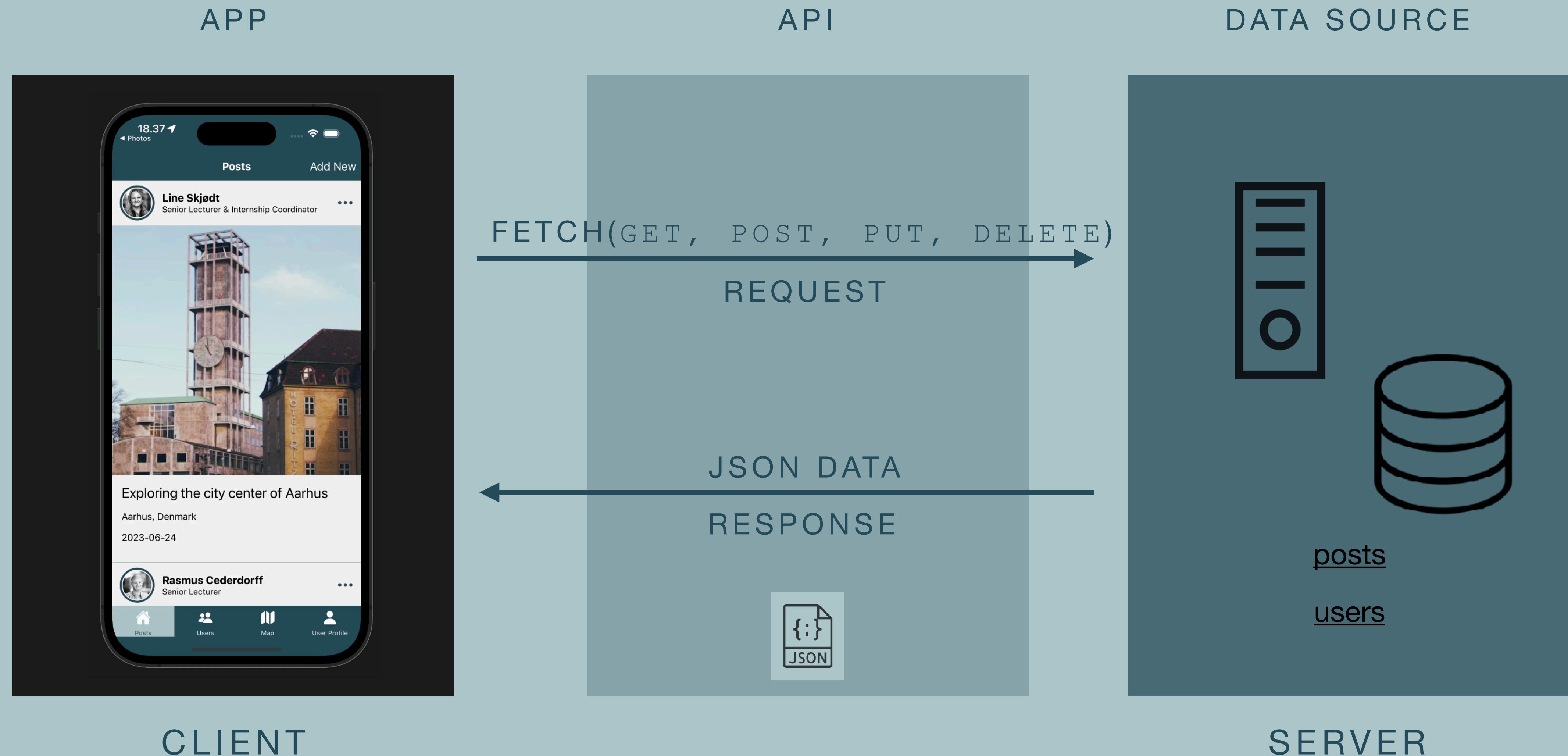
JavaScript Object Notation

... a syntax for storing & exchanging data
over the web



<https://www.instagram.com/p/CVqbCzgsZUF/>

JSON (& API) is the glue



JSON

... a syntax for storing and exchanging data over the web

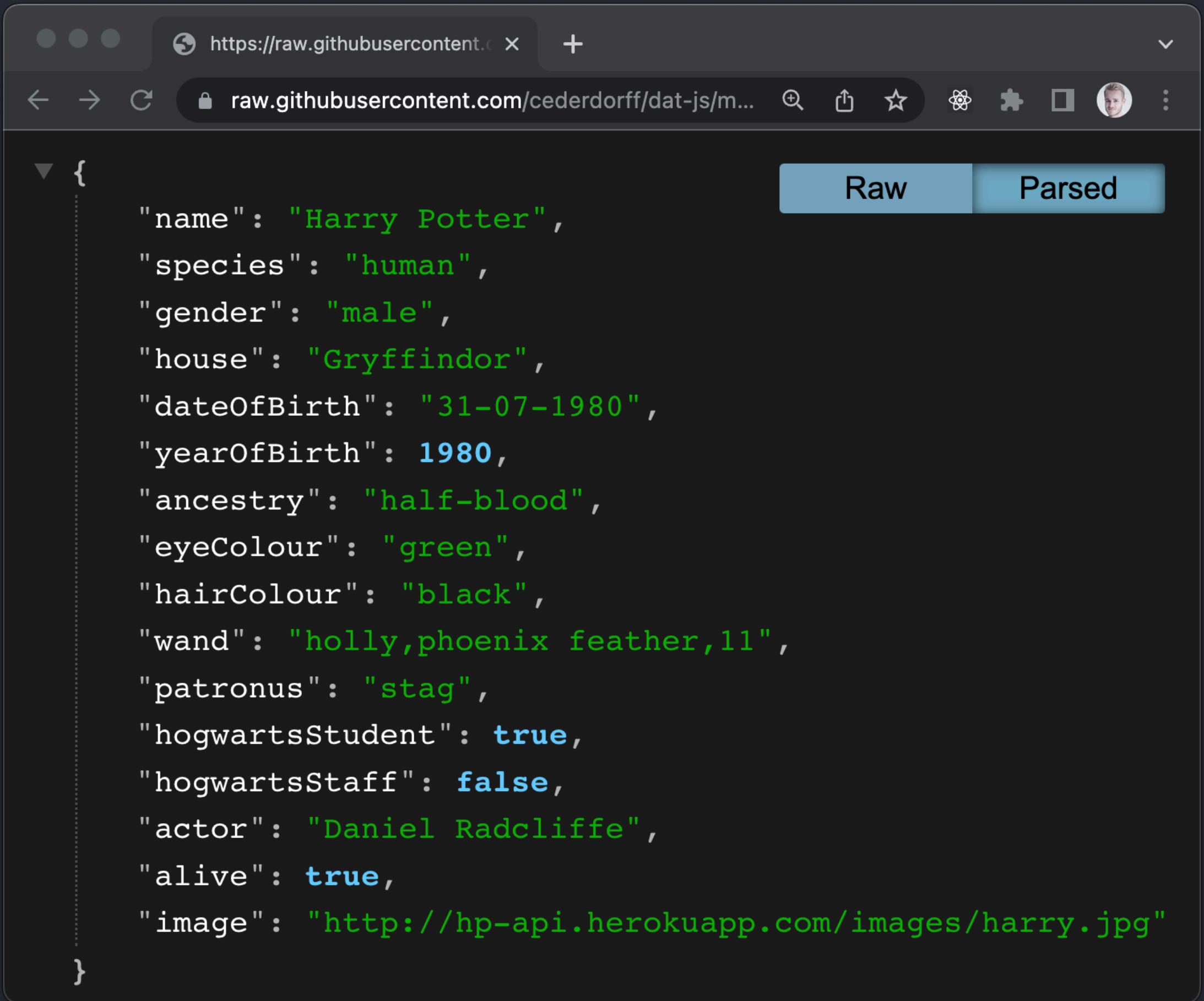
```
{  
  "name": "Alicia",  
  "age": 6  
}
```

JSON OBJECT

```
[{  
  "name": "Alicia",  
  "age": 6  
, {  
  "name": "Peter",  
  "age": 22  
}]
```

LIST OF JSON OBJECTS

JSON

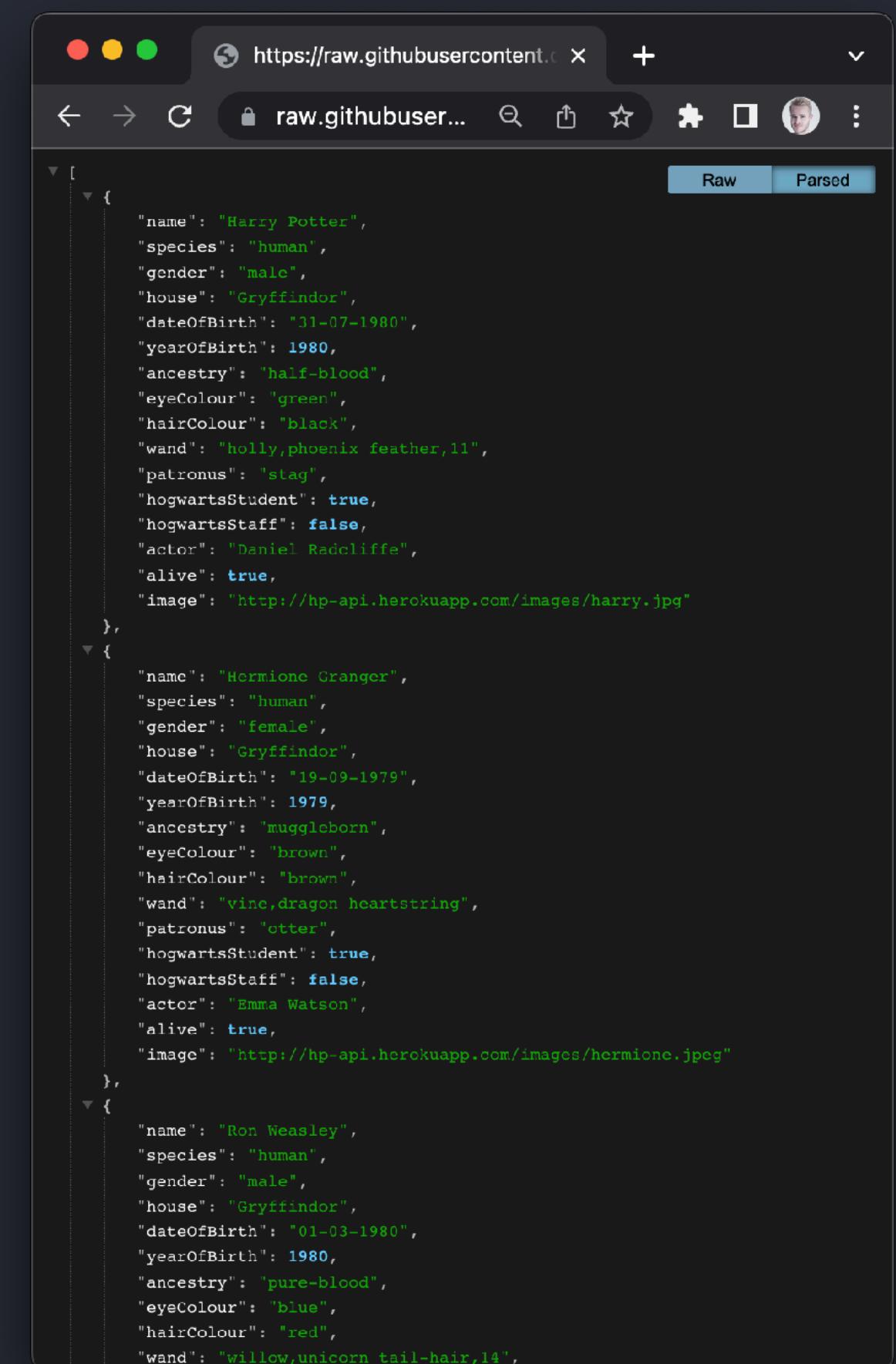


A screenshot of a web browser window displaying a single JSON object. The URL is <https://raw.githubusercontent.com/cederdorff/dat-js/master/data/harry.json>. The JSON structure is as follows:

```
{  
  "name": "Harry Potter",  
  "species": "human",  
  "gender": "male",  
  "house": "Gryffindor",  
  "dateOfBirth": "31-07-1980",  
  "yearOfBirth": 1980,  
  "ancestry": "half-blood",  
  "eyeColour": "green",  
  "hairColour": "black",  
  "wand": "holly,phoenix feather,11",  
  "patronus": "stag",  
  "hogwartsStudent": true,  
  "hogwartsStaff": false,  
  "actor": "Daniel Radcliffe",  
  "alive": true,  
  "image": "http://hp-api.herokuapp.com/images/harry.jpg"  
}
```

The JSON is displayed in two tabs: "Raw" and "Parsed". The "Parsed" tab shows the JSON structure with color-coded keywords (e.g., "name", "species", "gender") and values.

JSON OBJECT



A screenshot of a web browser window displaying a list of three JSON objects. The URL is <https://raw.githubusercontent.com/cederdorff/dat-js/master/data/characters.json>. The JSON structure is as follows:

```
[  
  {  
    "name": "Harry Potter",  
    "species": "human",  
    "gender": "male",  
    "house": "Gryffindor",  
    "dateOfBirth": "31-07-1980",  
    "yearOfBirth": 1980,  
    "ancestry": "half-blood",  
    "eyeColour": "green",  
    "hairColour": "black",  
    "wand": "holly,phoenix feather,11",  
    "patronus": "stag",  
    "hogwartsStudent": true,  
    "hogwartsStaff": false,  
    "actor": "Daniel Radcliffe",  
    "alive": true,  
    "image": "http://hp-api.herokuapp.com/images/harry.jpg"  
  },  
  {  
    "name": "Hermione Granger",  
    "species": "human",  
    "gender": "female",  
    "house": "Gryffindor",  
    "dateOfBirth": "19-09-1979",  
    "yearOfBirth": 1979,  
    "ancestry": "muggleborn",  
    "eyeColour": "brown",  
    "hairColour": "brown",  
    "wand": "vine,dragon heartstring",  
    "patronus": "otter",  
    "hogwartsStudent": true,  
    "hogwartsStaff": false,  
    "actor": "Emma Watson",  
    "alive": true,  
    "image": "http://hp-api.herokuapp.com/images/hermione.jpeg"  
  },  
  {  
    "name": "Ron Weasley",  
    "species": "human",  
    "gender": "male",  
    "house": "Gryffindor",  
    "dateOfBirth": "01-03-1980",  
    "yearOfBirth": 1980,  
    "ancestry": "pure-blood",  
    "eyeColour": "blue",  
    "hairColour": "red",  
    "wand": "willow,unicorn tail-hair,14",  
  }  
]
```

The JSON is displayed in two tabs: "Raw" and "Parsed". The "Parsed" tab shows the JSON structure with color-coded keywords (e.g., "name", "species", "gender") and values.

LIST OF JSON OBJECTS

JSON Object

The diagram illustrates the connection between a JSON object and a web application displaying Harry Potter characters.

Left Side: Web Application

A screenshot of a web browser window titled "Harry Potter Characters". The page displays three cards representing characters:

- Harry Potter**: Shows a portrait of Harry Potter and is labeled "Gryffindor".
- Hermione Granger**: Shows a portrait of Hermione Granger and is labeled "Gryffindor".
- Ron Weasley**: Shows a portrait of Ron Weasley and is labeled "Gryffindor".

Right Side: JSON Object

A screenshot of a web browser window showing the JSON representation of the Harry Potter character data. The JSON is displayed in a collapsible tree view with the "Parsed" tab selected. The data for Harry Potter is as follows:

```
{  
  "name": "Harry Potter",  
  "species": "human",  
  "gender": "male",  
  "house": "Gryffindor",  
  "dateOfBirth": "31-07-1980",  
  "yearOfBirth": 1980,  
  "ancestry": "half-blood",  
  "eyeColour": "green",  
  "hairColour": "black",  
  "wand": "holly,phoenix feather,11",  
  "patronus": "stag",  
  "hogwartsStudent": true,  
  "hogwartsStaff": false,  
  "actor": "Daniel Radcliffe",  
  "alive": true,  
  "image": "http://hp-api.herokuapp.com/images/harry.jpg"  
}
```

Red arrows point from the character cards in the application to their corresponding entries in the JSON object, indicating the data mapping.

<https://raw.githubusercontent.com/cederdorff/dat-js/main/data/harry.json>

JSON Object

The diagram illustrates the flow of data from a JSON object to a JavaScript function. A pink curved arrow originates from the JSON object on the right and points to the character variable in the showCharacter function on the left.

app.js — potter-app

```
JS app.js M X
JS app.js > ⚡ showCharacterModal
28
29  function showCharacter(character) {
30      document.querySelector("#characters").insertAdjacentHTML(
31          "beforeend",
32          /*html*/
33          `

34              
35              <h2>${character.name}</h2>
36              <p>${character.house}</p>
37          </article>
38      `;
39  );
40 }


```

Ln 48, Col 19 Spaces: 4 UTF-8 LF {} JavaScript Port : 5500 ✓ Prettier

https://raw.githubusercontent.com/cederdorff/dat-js/main/data/harry.json

```
Raw Parsed
{
    "name": "Harry Potter",
    "species": "human",
    "gender": "male",
    "house": "Gryffindor",
    "dateOfBirth": "31-07-1980",
    "yearOfBirth": 1980,
    "ancestry": "half-blood",
    "eyeColour": "green",
    "hairColour": "black",
    "wand": "holly,phoenix feather,11",
    "patronus": "stag",
    "hogwartsStudent": true,
    "hogwartsStaff": false,
    "actor": "Daniel Radcliffe",
    "alive": true,
    "image": "http://hp-api.herokuapp.com/images/harry.jpg"
}
```

<https://raw.githubusercontent.com/cederdorff/dat-js/main/data/harry.json>

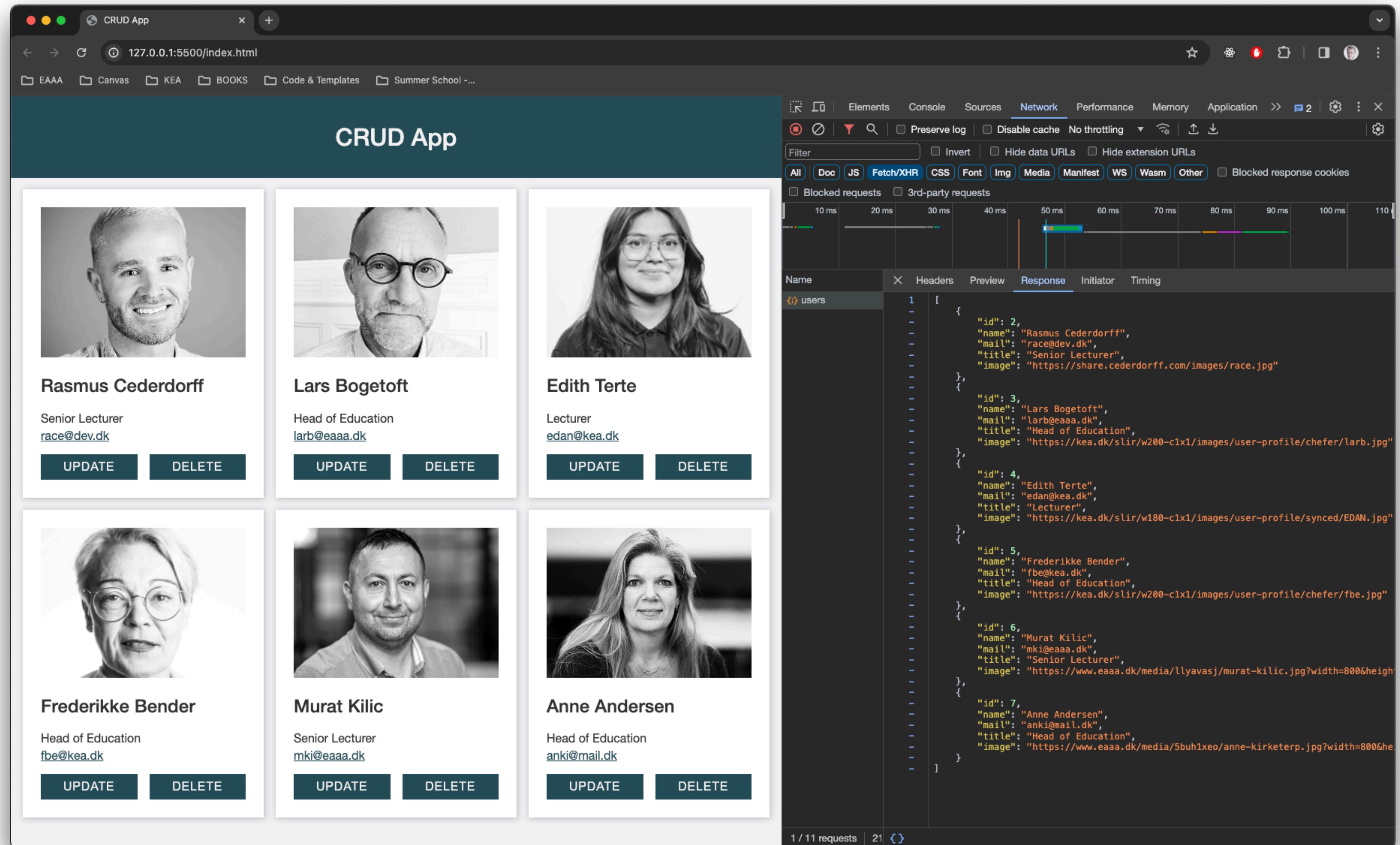
JAVASCRIPT OBJECT NOTATION

- Collection of key-value pair: “key” : “value”
- List of values, collections or objects
- Lightweight data-interchange format
- Syntax / text format for storing and exchanging data over the web
- Human and machine readable **text**: small, fast and simple
- Language independent
- Can be parsed directly to JavaScript Object
- JavaScript Objects can be converted directly to JSON
- The glue between programs (interface between frontend and backend)

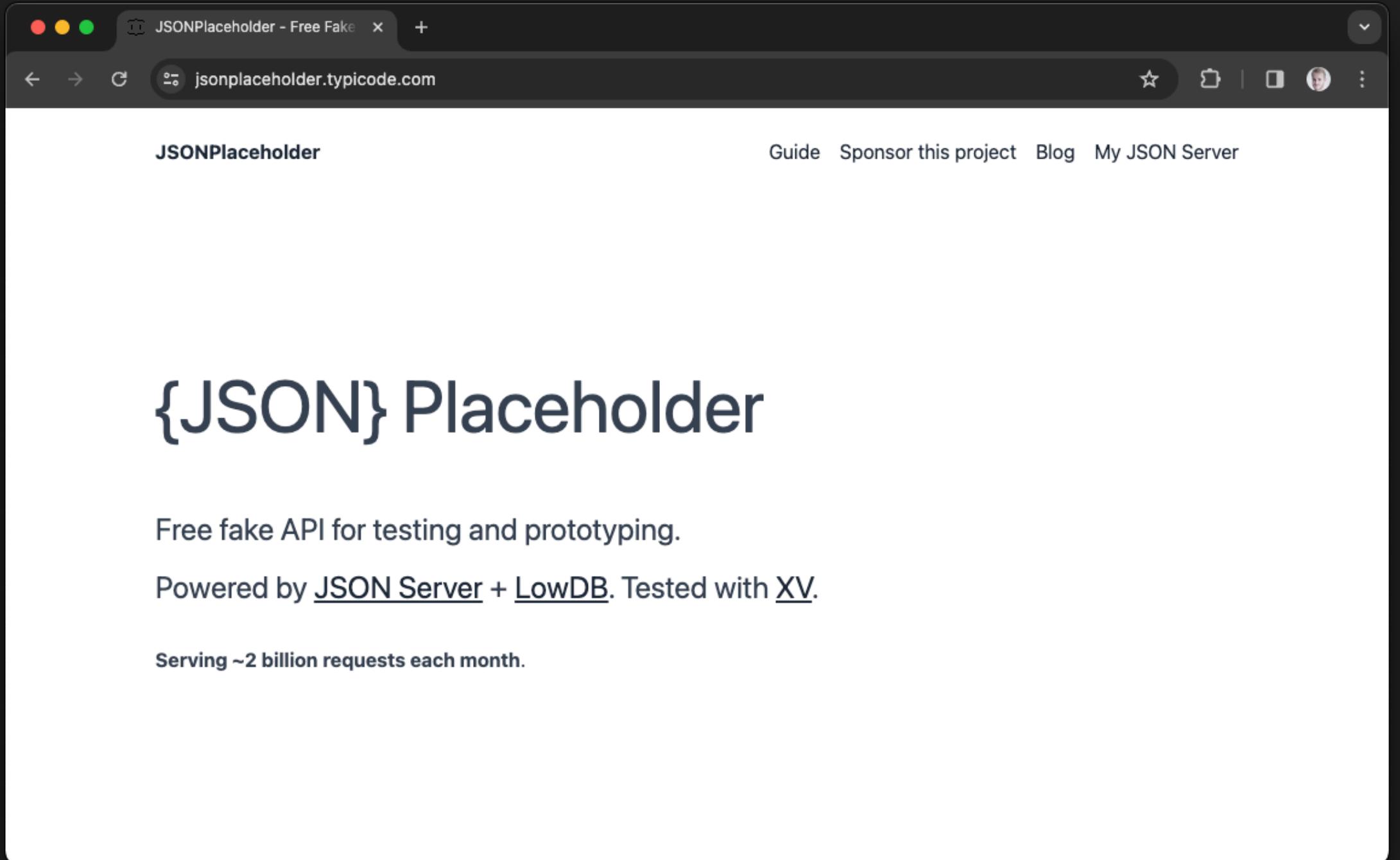
```
[  
 {  
   "id": "1",  
   "firstname": "Kasper",  
   "lastname": "Topp",  
   "age": "34",  
   "haircolor": "Dark Blonde",  
   "countryName": "Denmark",  
   "gender": "Male",  
   "lookingFor": "Female"  
 },  
 {  
   "id": "2",  
   "firstname": "Nicklas",  
   "lastname": "Andersen",  
   "age": "22",  
   "haircolor": "Brown",  
   "countryName": "Denmark",  
   "gender": "Male",  
   "lookingFor": "Female"  
 },  
 {  
   "id": "3",  
   "firstname": " Sarah",  
   "lastname": "Dybvad ",  
   "age": "34",  
   "haircolor": "Blonde",  
   "countryName": "Denmark",  
   "gender": "Female",  
   "lookingFor": "Male"  
 },  
 {  
   "id": "4",  
   "firstname": "Alex",  
   "lastname": "Hansen",  
   "age": "21",  
   "haircolor": "Blonde",  
   "countryName": "Denmark",  
   "gender": "Male",  
   "lookingFor": "Female"  
 }]
```

JSON METHODS

```
const user = {  
    name: "John",  
    age: 30,  
    gender: "male",  
    lookingFor: "female"  
};  
  
// === JSON.stringify === //  
const jsonUser = JSON.stringify(user);  
console.log(jsonUser); // {"name":"John","age":30,"gender":"male","lookingFor":"female"}  
  
// === JSON.parse === //  
const jsonString = '{"name":"John","age":30,"gender":"male","lookingFor":"female"}';  
const userObject = JSON.parse(jsonString);  
console.log(userObject); // logging userObject
```



Test endpoints



<https://jsonplaceholder.typicode.com/>

- Install [JSON Formatter](#)
- Test the following endpoints:
 - <https://jsonplaceholder.typicode.com/posts/>
 - <https://jsonplaceholder.typicode.com/posts/5>
 - <https://jsonplaceholder.typicode.com/users/>
 - <https://jsonplaceholder.typicode.com/users/1>
 - <https://jsonplaceholder.typicode.com/todos>
 - <https://jsonplaceholder.typicode.com/todos/5>

```
{  
  "name": "Rasmus Cederdorff",  
  "birthday": "1990-03-12",  
  "title": "Experienced JavaScript Developer",  
  "experienceYears": 10,  
  "education": {  
    "degree": "Master of Science in IT - Web Communication",  
    "specialization": "Web Architecture"  
  },  
  "skills": ["JavaScript", "React", "Node.js", "UI/UX Design"],  
  "currentPosition": "Senior Lecturer at EAAA",  
  "teachingSubjects": ["Web Development", "JavaScript", "React", "BaaS & Node.js"],  
  "lovesJavaScript": true  
}
```

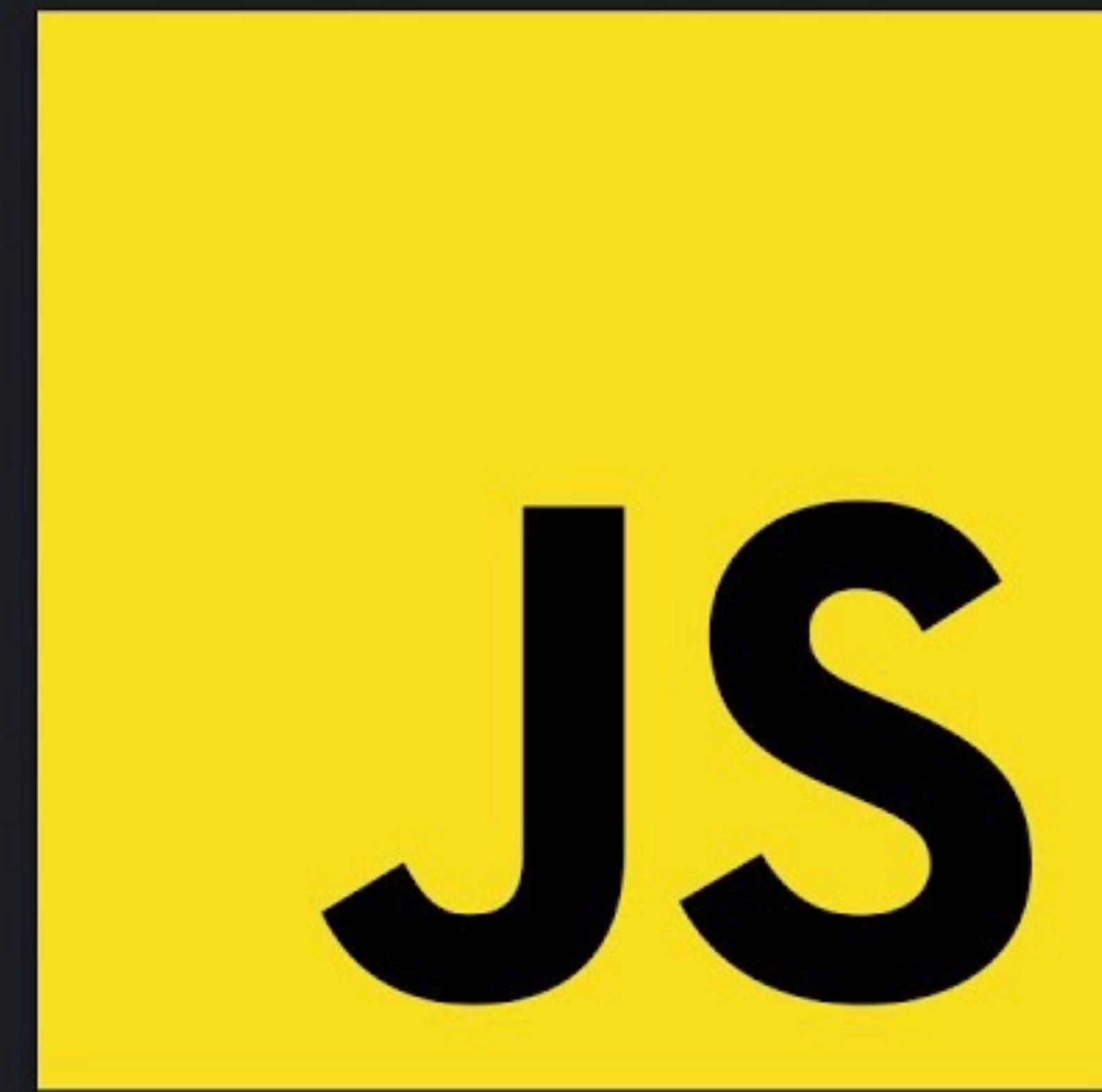
Define yourself with JSON

- <https://race.notion.site/Define-yourself-with-JSON-391944b6b1a041bfa341a0d46f5c0e4a?pvs=4>

DOM Manipulation

Application Programming Interface

100 *SECONDS OF*



<https://www.youtube.com/watch?v=DHjqpvDnNGE>

“

The cool thing about JavaScript is
that you can create stuff that will put
a smile on your face, as a developer.

You can create stuff and it will
visually look like something, and it'll
do stuff, and it makes you feel good,
it makes you fall in love with
programming.

”

Lex Fridman
Computer Scientist

<https://www.youtube.com/watch?v=GLhyjVZp0cw&t=252s>

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Page Title</title>
5      <link rel="stylesheet" href="styles.css" />
6    </head>
7    <body>
8      <h1>This is a Heading</h1>
9      <p>This is a paragraph.</p>
10     <button onclick="tryMe()">Try me</button>
11     <script src="app.js"></script>
12   </body>
13 </html>
14
```

What is JavaScript?

.. is the world's most popular programming language.

... is the programming language of the Web.

... is easy to learn.

... can change content of a webpage (HTML content).

... can change styling of HTML.

```
1  function tryMe() {
2    document.body.style.backgroundColor = "red";
3    document.body.style.color = "white";
4  }
5
```

index.html x

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Page Title</title>
5      <link rel="stylesheet" href="styles.css" />
6    </head>
7    <body>
8      <h1>This is a Heading</h1>
9      <p>This is a paragraph.</p>
10     <button onclick="tryMe()">Try me</button>
11     <script src="app.js"></script>
12   </body>
13 </html>
```

With JavaScript we are able to

... build dynamic web pages and web apps.

... fetch content/ data from a backend (web service, data source, etc.) through an API.

app.js x

```
1  function tryMe() {
2    document.body.style.backgroundColor = "red";
3    document.body.style.color = "white";
4  }
5
```

... do DOM-manipulation.

... build and develop anything 

DOM Manipulation

```
// declaring a variable with a value
let message = "Hi Frontenders!"

//accessing the variable and logging it to the console
console.log(message);

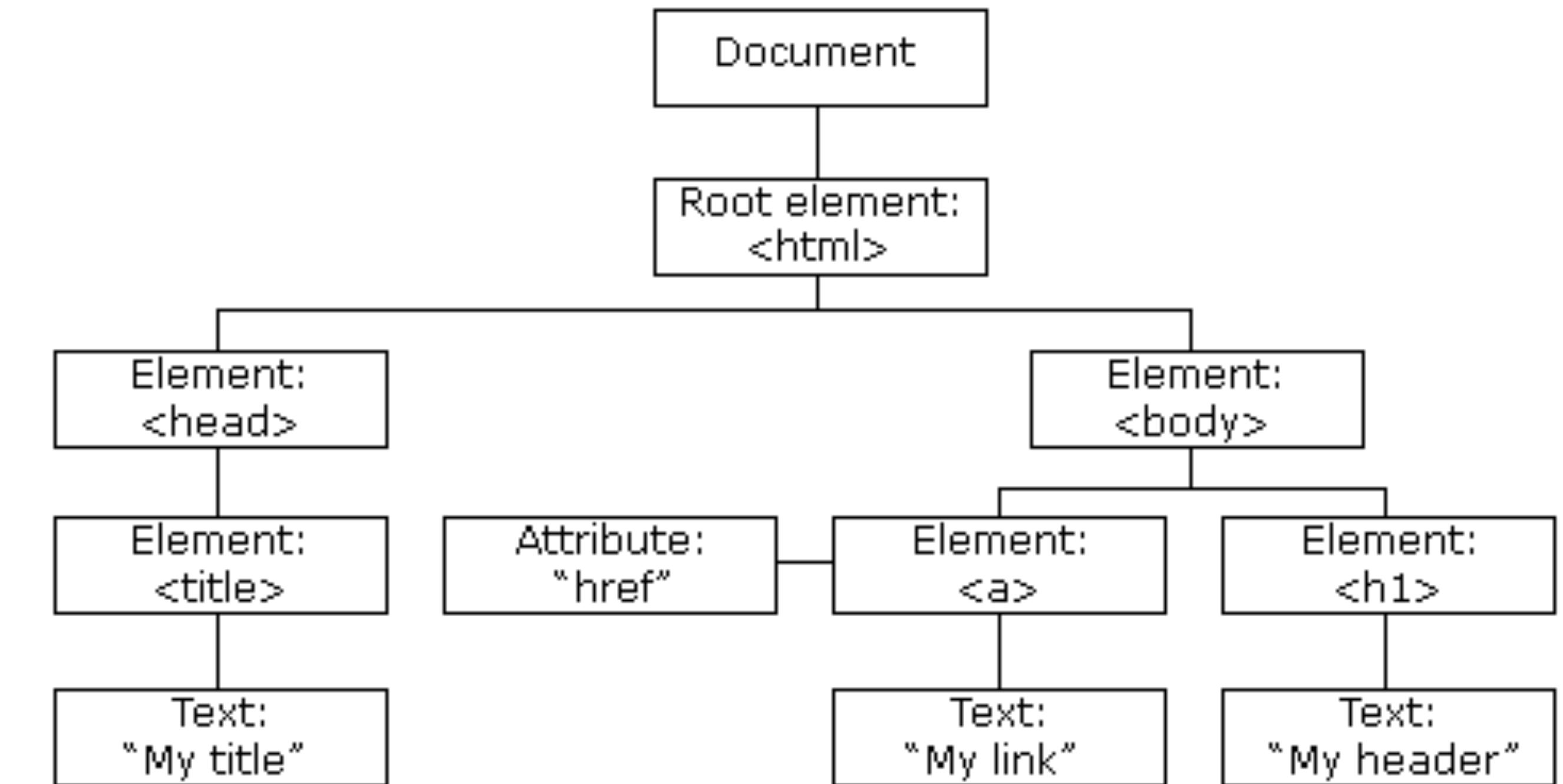
// appending the variable (the string) to the DOM element #content
document.querySelector("#content").innerHTML = message;
```

```
<body>
  <header>
    <h1>PROJECT TEMPLATE</h1>
  </header>
  <section id="content"></section>
  <!-- main is file -->
  <script src="js/main.js"></script>
</body>
```



JavaScript HTML DOM

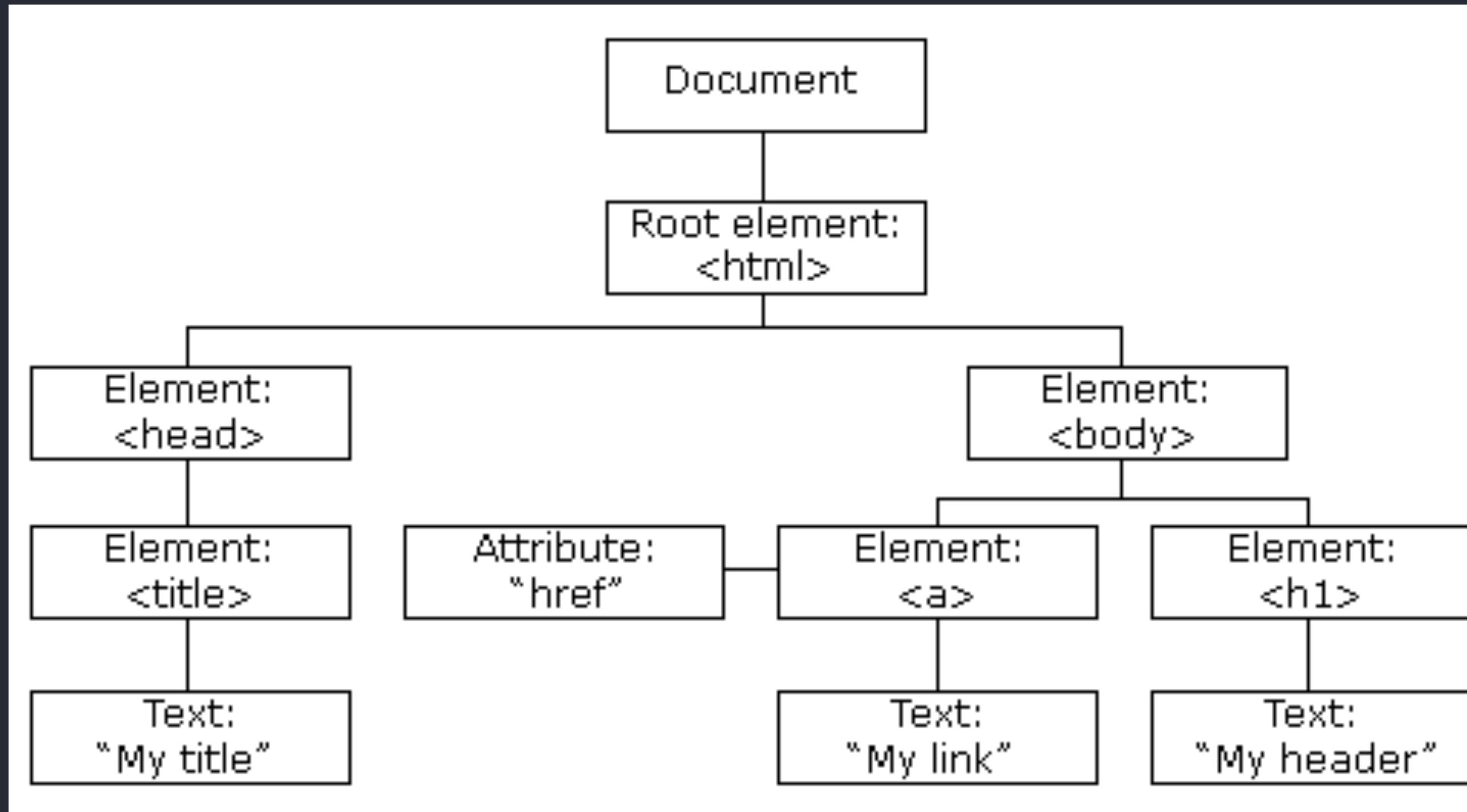
```
index.html *  
1  <!DOCTYPE html>  
2  <html>  
3  | <head>  
4  | | <title>My title</title>  
5  | </head>  
6  |  
7  <body>  
8  | | <h1>My header</h1>  
9  | | <a href="https://cederdorff.com">My link</a>  
10 | </body>  
11 |  
12 </html>
```



https://www.w3schools.com/js/js_htmldom.asp
<https://javascript.info/dom-nodes>
<https://javascript.info/dom-navigation>

The HTML DOM (Document Object Model)

A model as a tree of Objects



- Object Model for HTML:
 - HTML elements as objects
 - Properties for all HTML elements
 - Methods for all HTML elements
 - Events for all HTML elements

JavaScript HTML DOM

Document Object Model

```
index.html ×  
1  <!DOCTYPE html>  
2  <html>  
3  |   <head>  
4  |   |   <title>My title</title>  
5  |   </head>  
6  
7  <body>  
8  |   <h1>My header</h1>  
9  |   <a href="https://cederdorff.com">My link</a>  
10 |</body>  
11 </html>  
12
```

The HTML document as an object

Gives us the power to create dynamic HTML and manipulate with the HTML (the DOM).

JavaScript can:

- ... change all the HTML elements in the page*
- ... change all the HTML attributes in the page*
- ... change all the CSS styles in the page*
- ... remove existing HTML elements and attributes*
- ... add new HTML elements and attributes*
- ... react to all existing HTML events in the page*
- ... create new HTML events in the page*

https://www.w3schools.com/js/js_htmldom.asp

<https://javascript.info/dom-nodes>

<https://javascript.info/dom-navigation>

DOM



The screenshot shows the DOM tab of a browser's developer tools. At the top, there are three colored circular icons (red, yellow, green) followed by the word "Elements". Below this, the DOM structure is listed with line numbers:

```
1 <html>
2   <head></head>
3   <body>
4     <div id="app">
5       <h1>Develop. Preview.
6     Ship. 🚀</h1>
7     </div>
8     <script type="text/
9     javascript">...</script>
10    </body>
11  </html>
```

SOURCE CODE (HTML)



The screenshot shows the Source tab of a browser's developer tools. At the top, there are three colored circular icons (red, yellow, green) followed by the file name "index.html". Below this, the raw HTML code is listed with line numbers:

```
1 <html>
2   <head></head>
3   <body>
4     <div id="app"></div>
5     <script type="text/
6     javascript">...</script>
7   </body>
8 </html>
9
10
11
```

Searching the DOM: getElement* & querySelector*

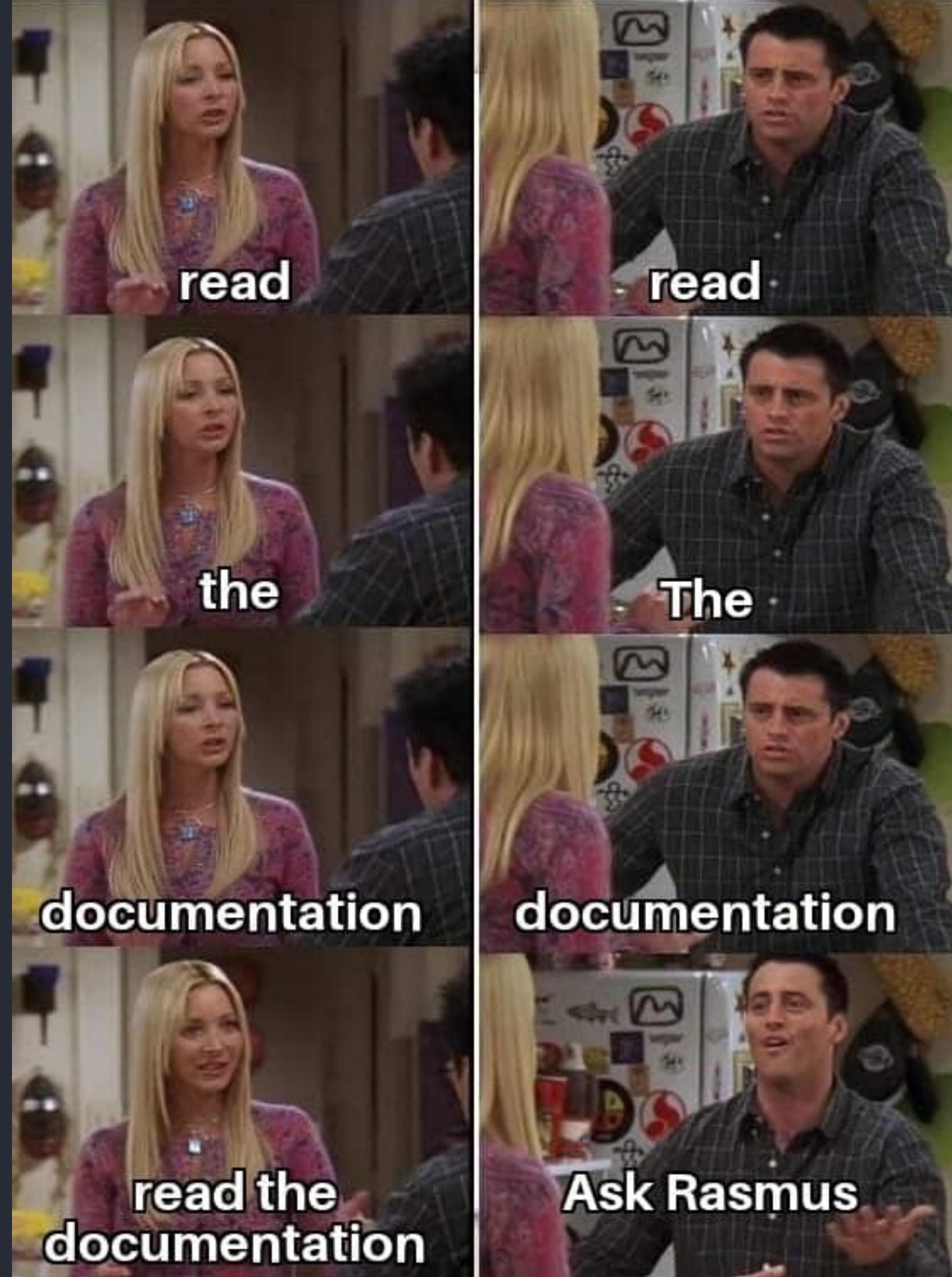
```
<section id="elem">
  <article id="elem-content">Element</article>
</section>

<script>
  // get the element
  const element = document.getElementById('elem');
  // make its background red
  element.style.background = 'red';
  // get the elementContent
  const elementContent = document.querySelector('#elem-content');
  // change inner HTML
  elementContent.innerHTML = "<h2>Hi Web Developers!</h2>"
</script>
```

Searching the DOM: querySelectorAll

```
<section id="elem">
  <article class="elem-content">Element</article>
  <article class="elem-content">Element</article>
  <article class="elem-content">Element</article>
</section>

<script>
  // get all elements matching the selector - returns an array
  const elements = document.querySelectorAll('.elem-content');
  // loop through all elements
  for (const element of elements) {
    element.innerHTML = "<h2>Hi Web Developers!</h2>";
  }
</script>
```



Headless WordPress

Access your content through WP REST API

What's a Headless CMS?



A CMS without a head 🤔

I'm not kidding

It's a CMS without any frontend



Headless CMS

...is a system to control the content that is separated from the display layer or the front-end user experience.

What
Is Headless CMS?



Traditional CMS

FRONTEND

CMS

BACKEND

Headless CMS

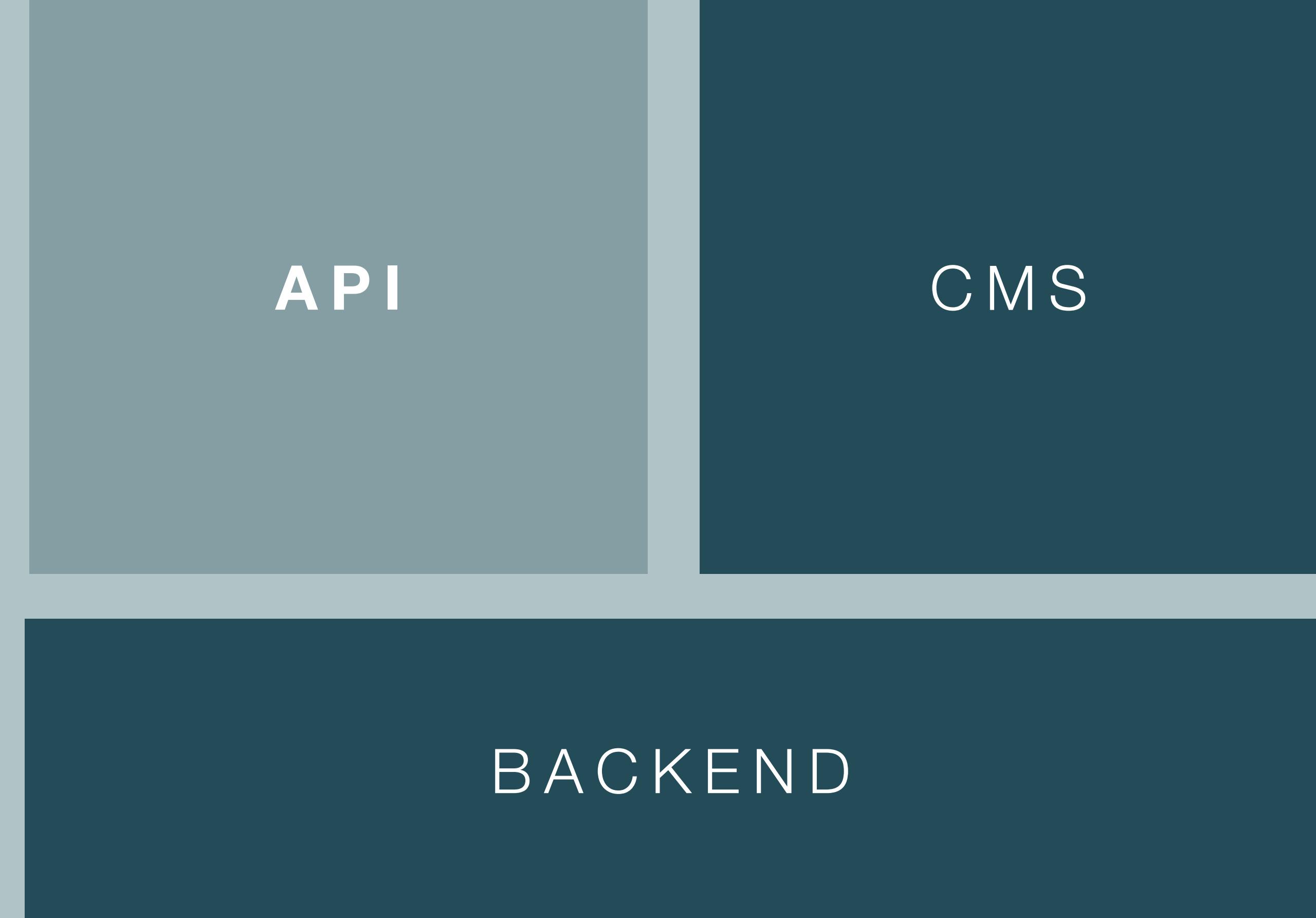
The diagram illustrates the architecture of a Headless CMS. It features three rectangular boxes arranged horizontally. The top-left box is light gray and contains the word "FRONTEND". The top-right box is dark teal and contains the letters "CMS". The bottom box is also dark teal and contains the word "BACKEND".

FRONTEND

CMS

BACKEND

Headless CMS



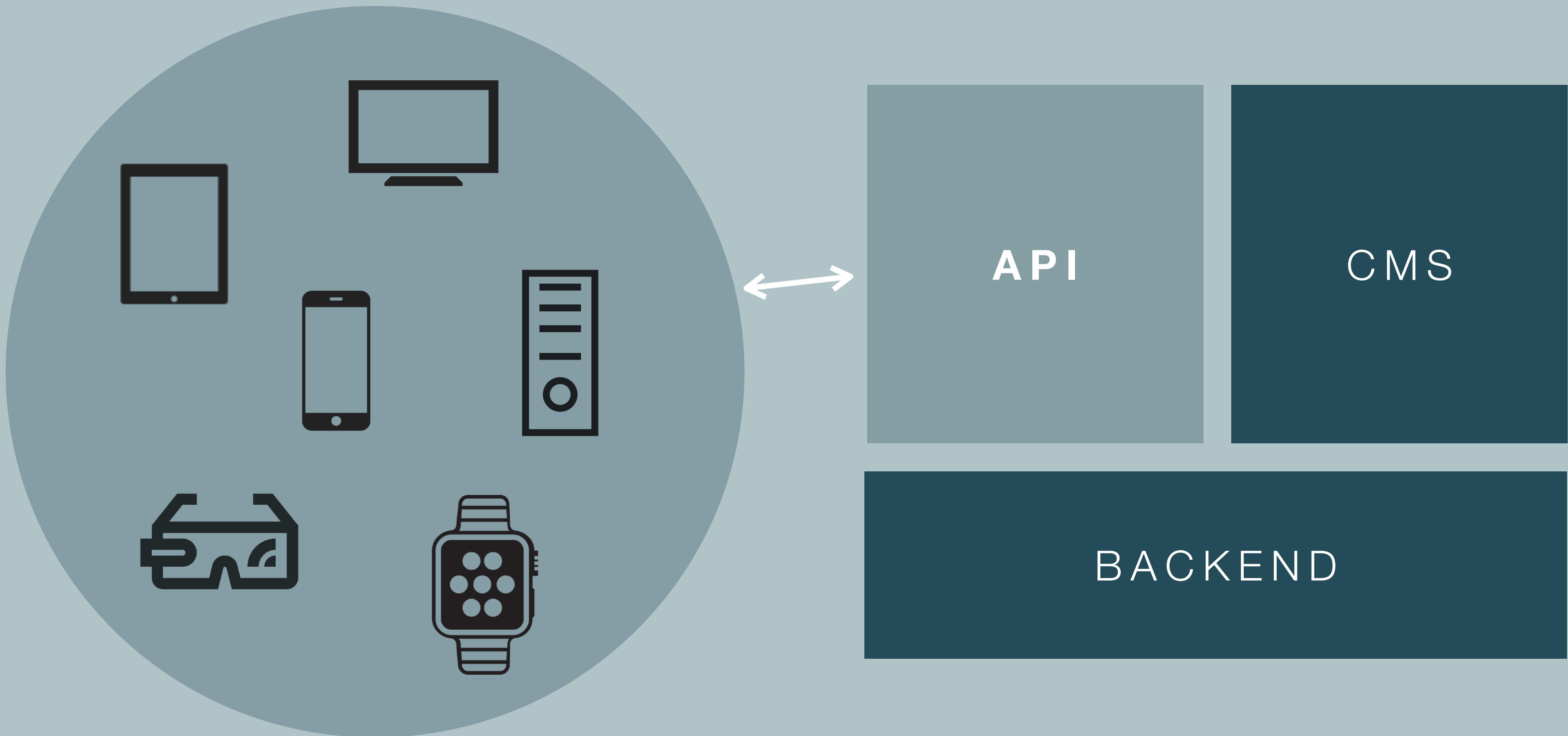
The diagram illustrates the architecture of a Headless CMS. It features three rectangular boxes arranged horizontally. The top-left box is light gray and contains the text "API". The top-right box is dark teal and contains the text "CMS". The bottom box is also dark teal and contains the text "BACKEND".

API

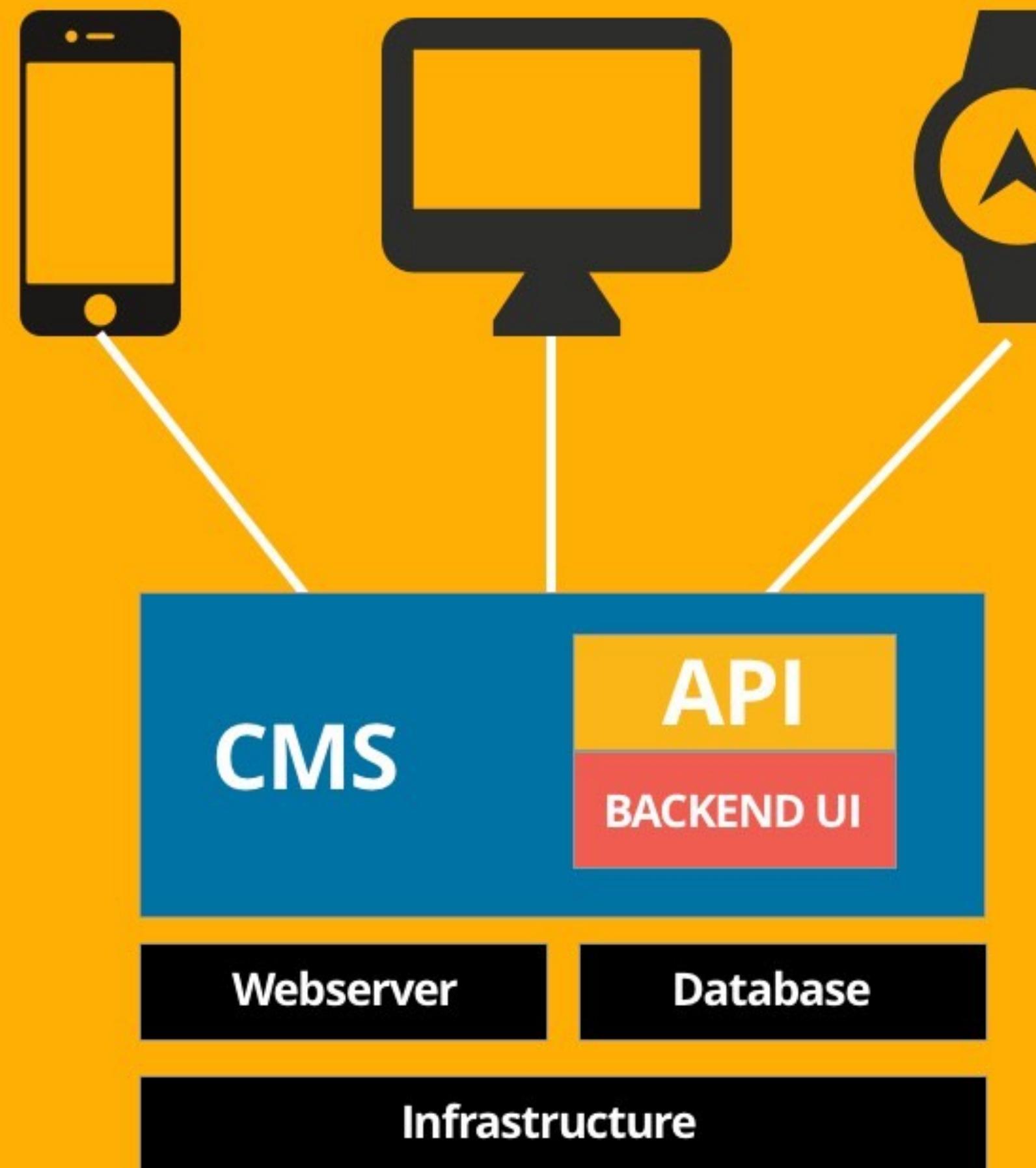
CMS

BACKEND

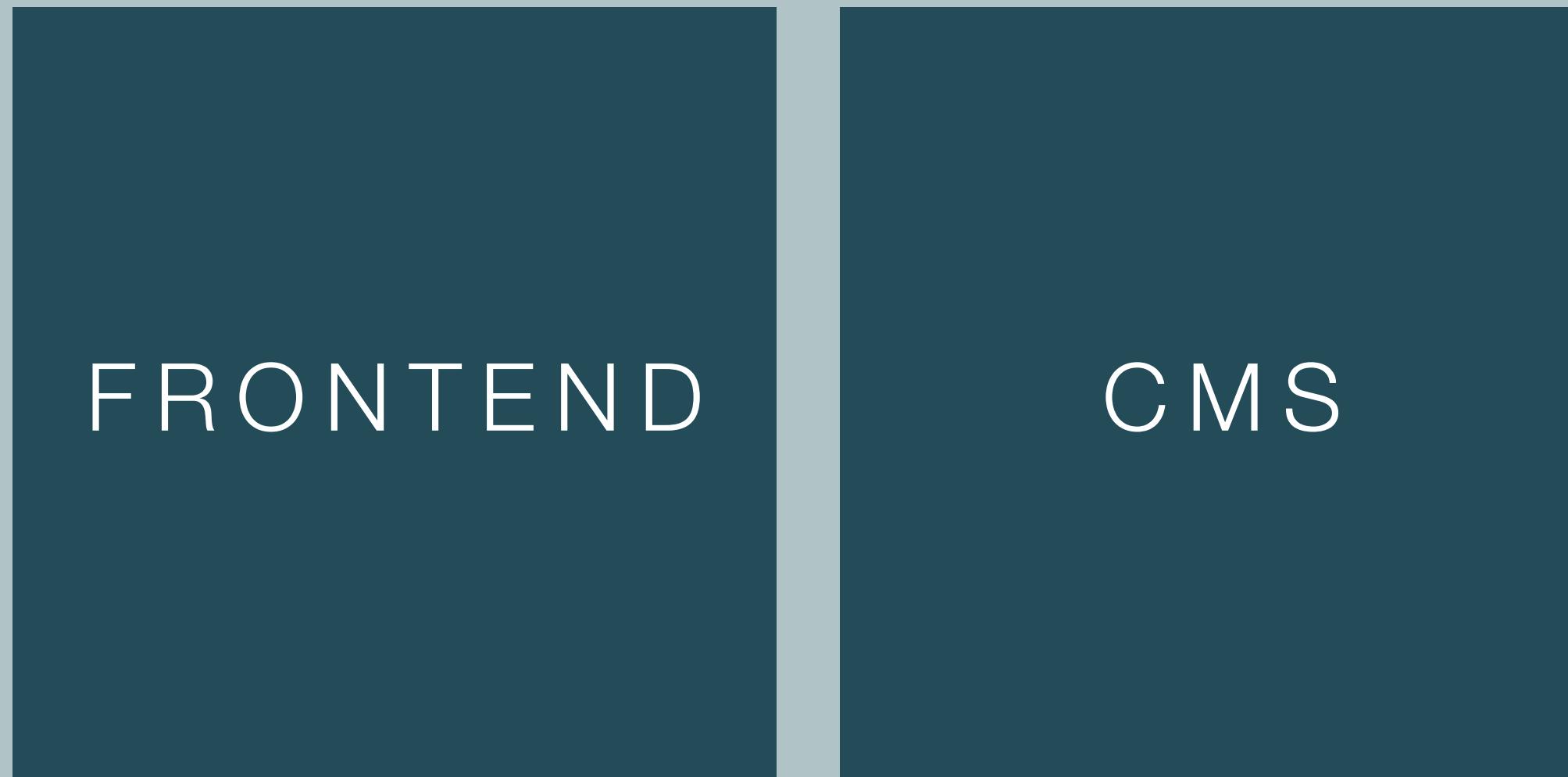
Headless CMS



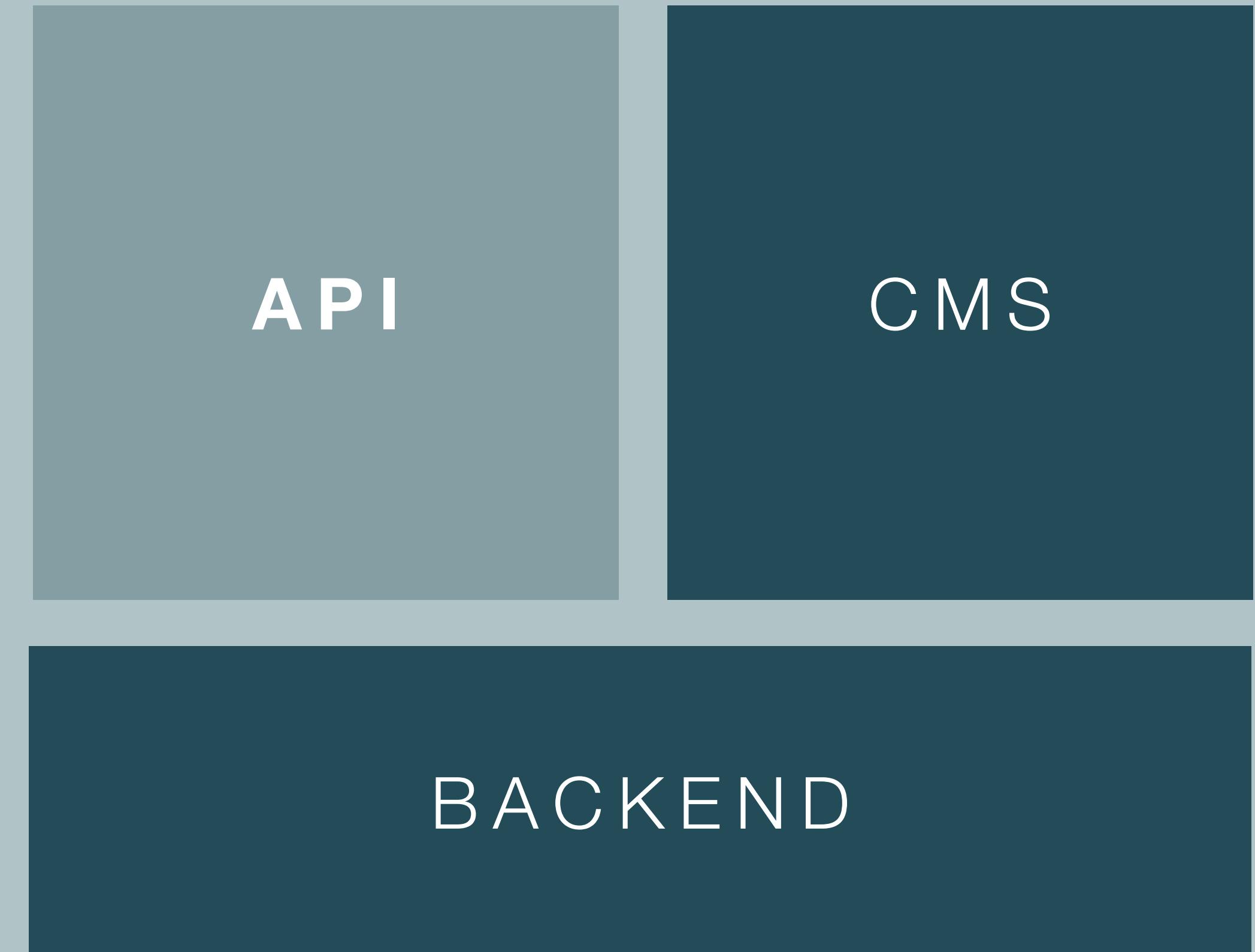
Headless CMS



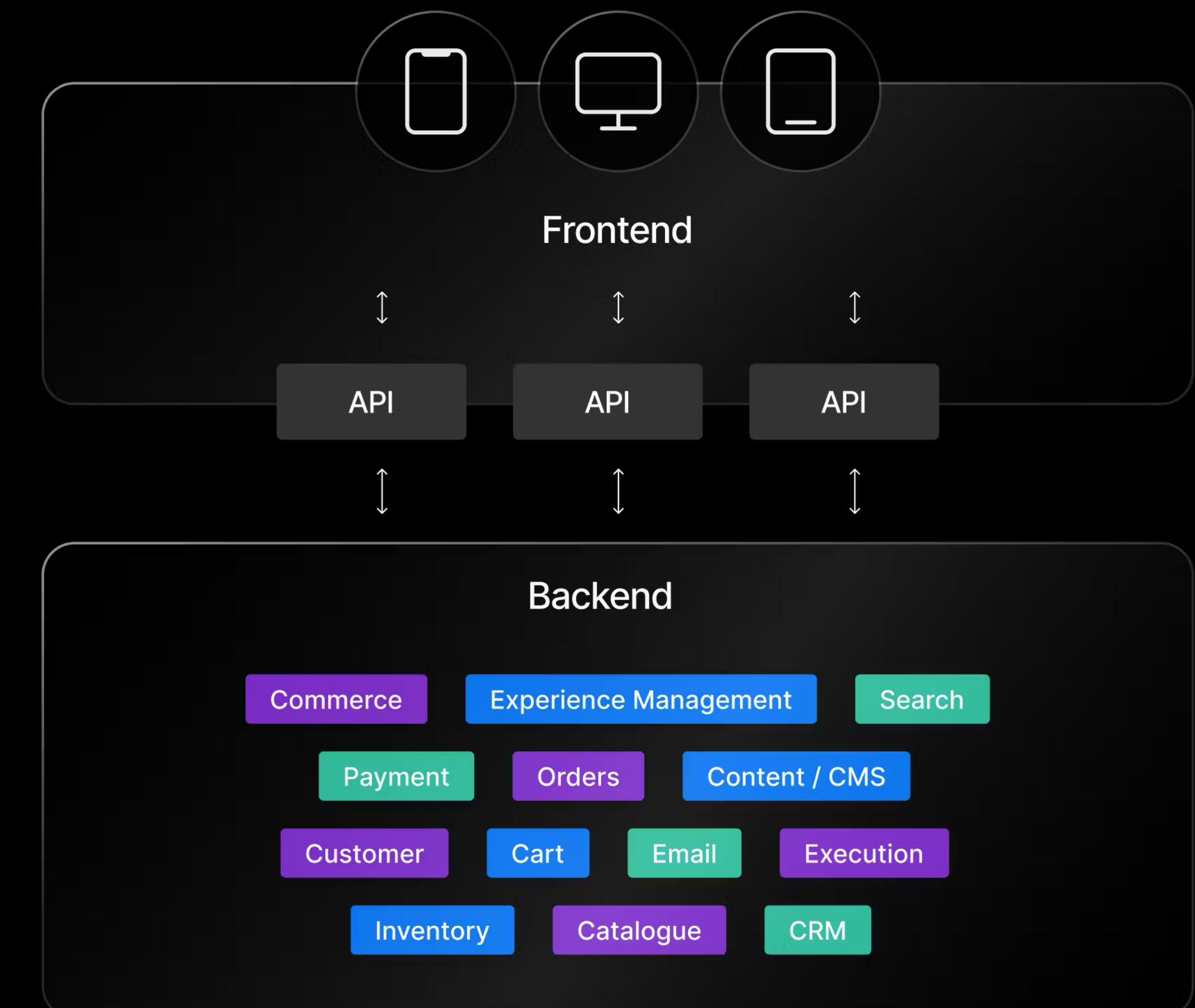
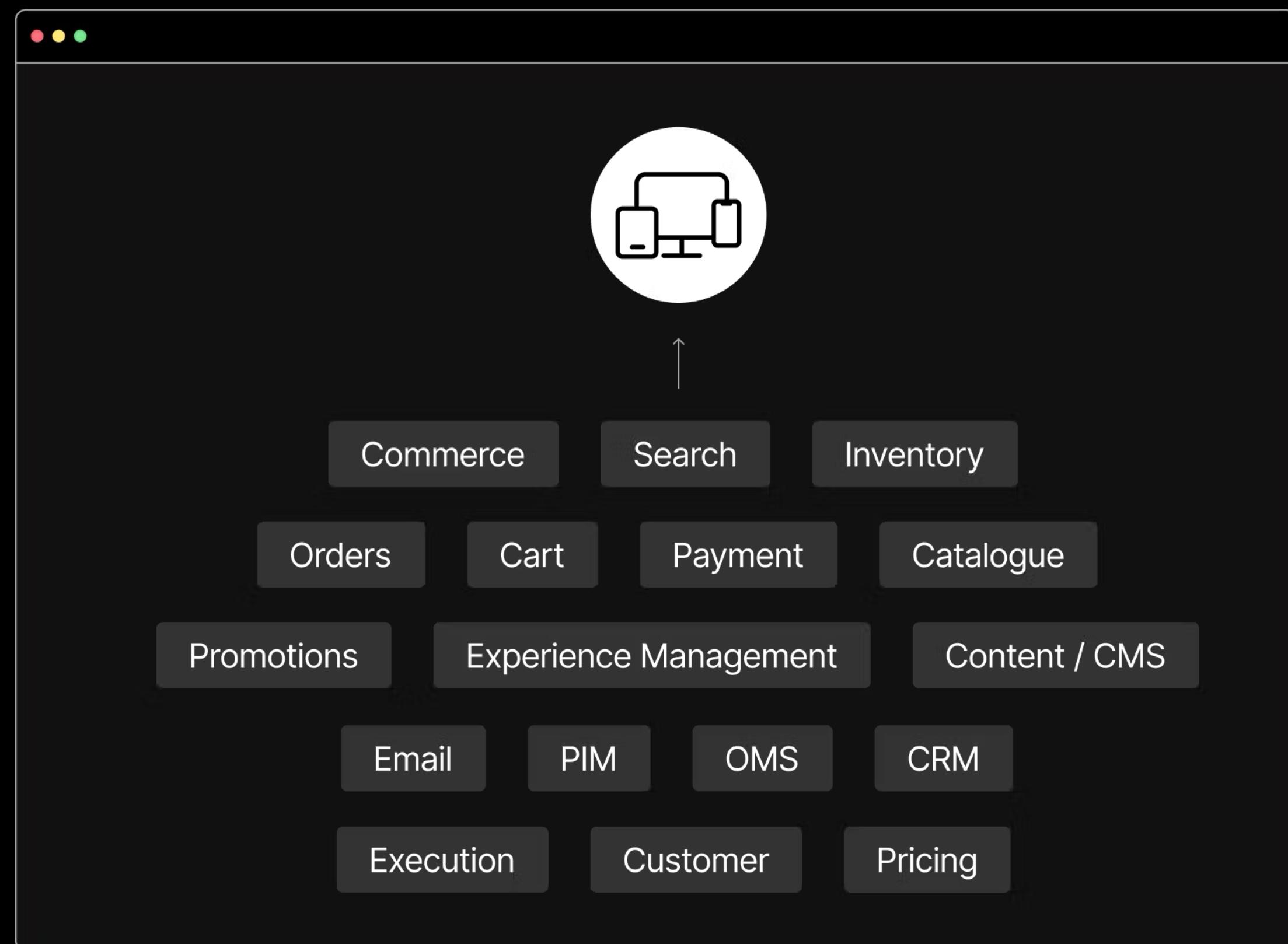
Traditional CMS



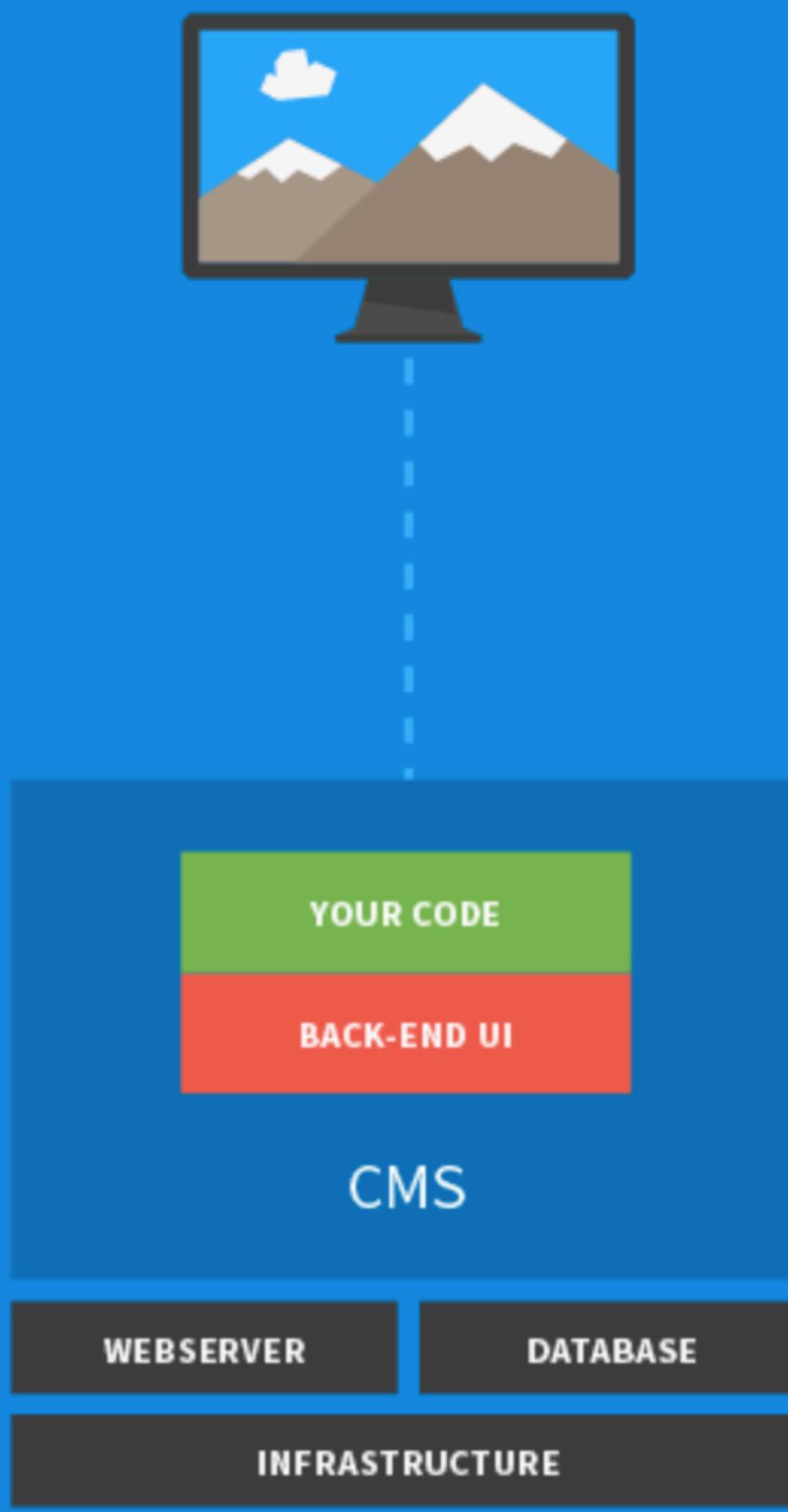
Headless CMS



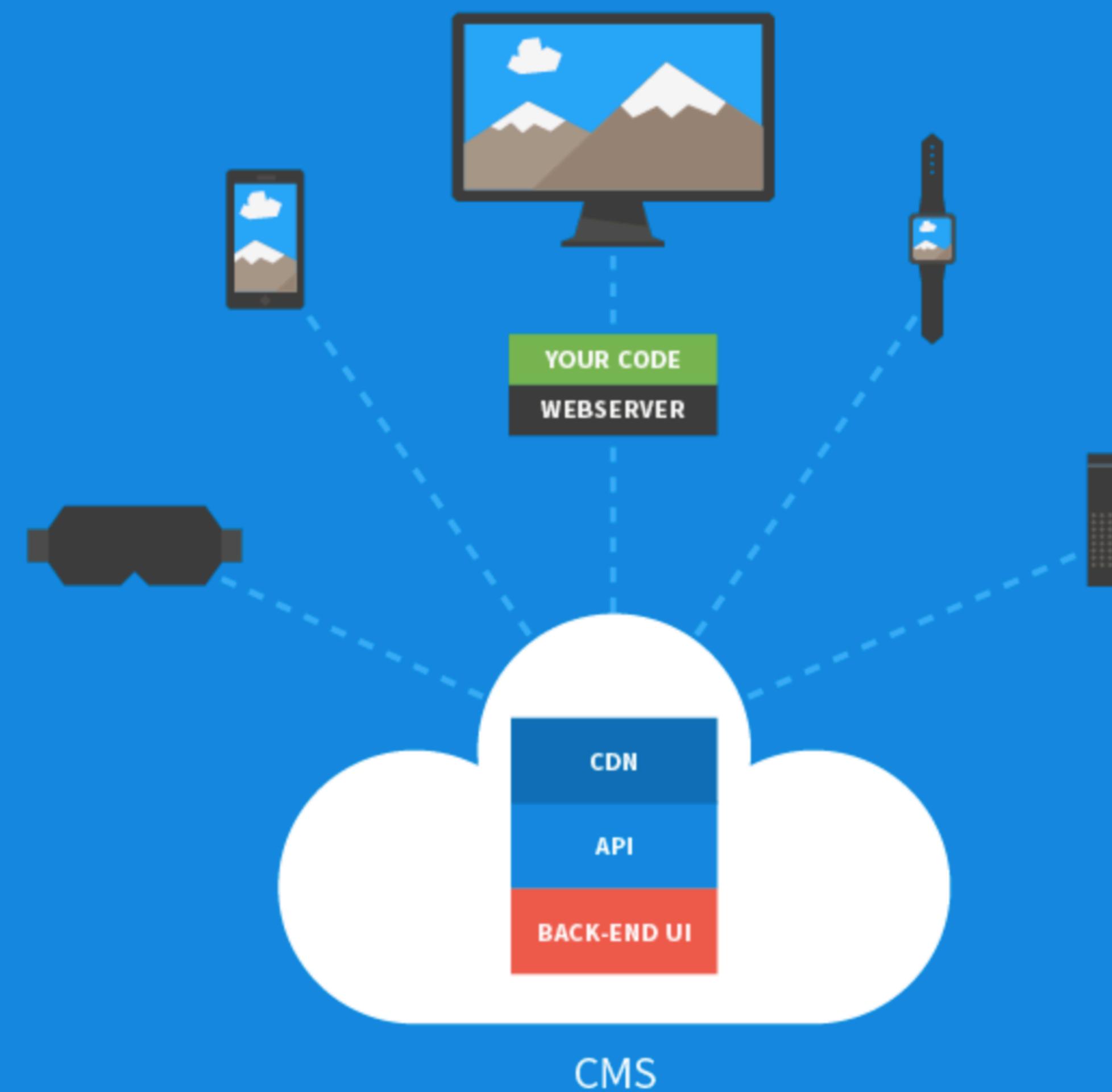
Monolith vs. Headless



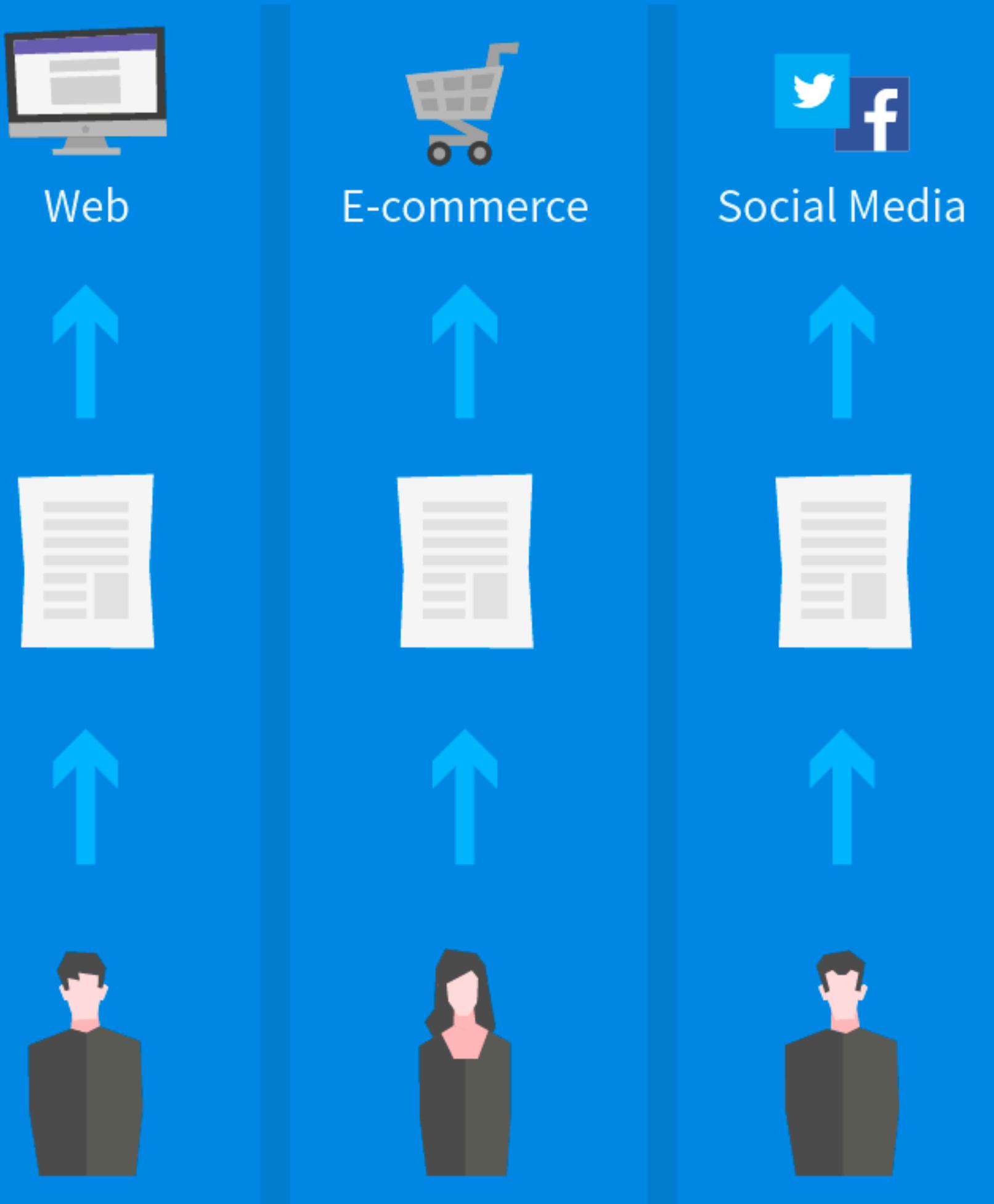
TRADITIONAL CMS



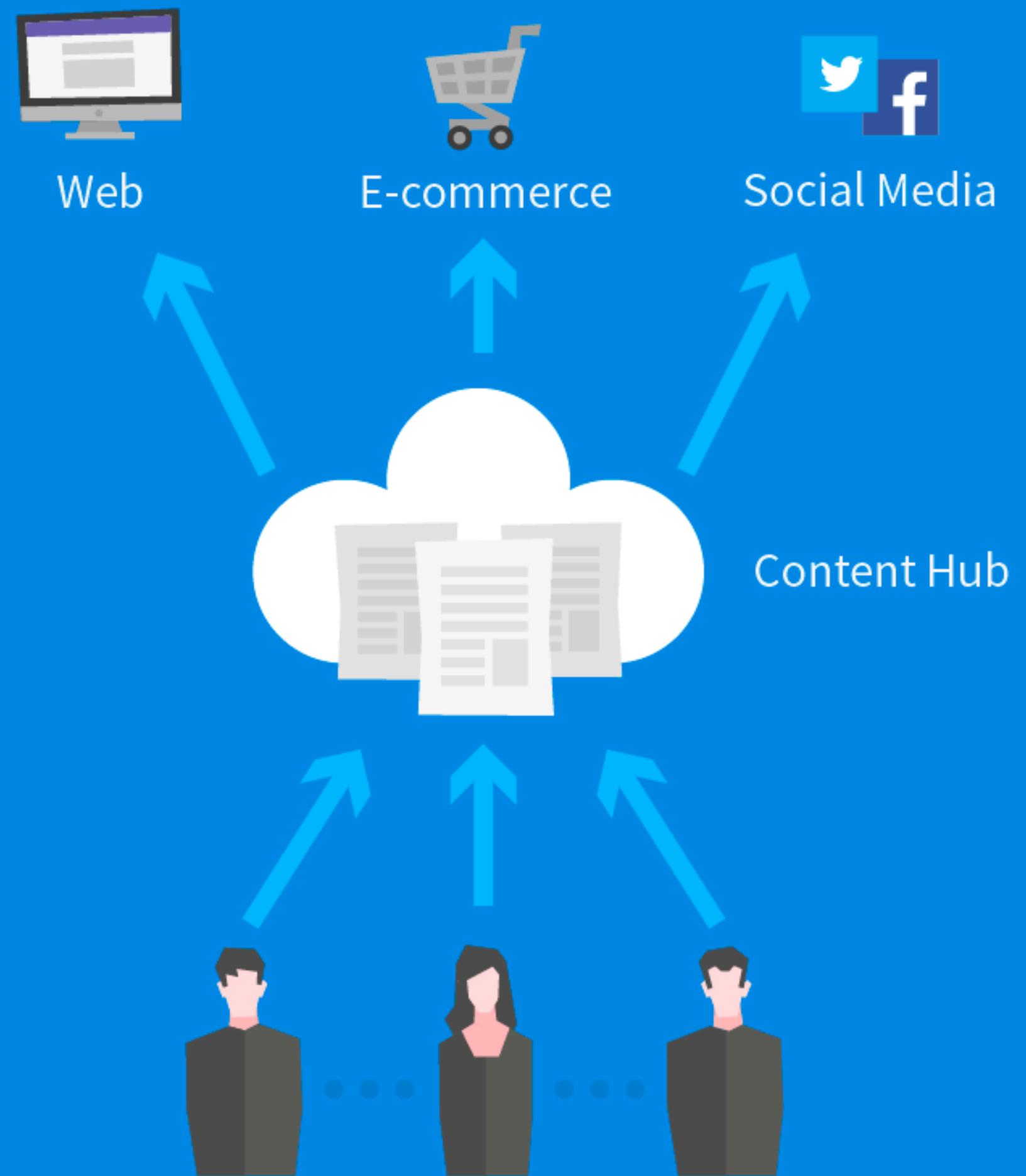
HEADLESS CMS



TRADITIONAL CMS



HEADLESS CMS



TRADITIONAL CMS

A WAY TO STORE DATA

A CRUD UI

A WAY TO DISPLAY THE DATA

CRUD: CREATE, READ, UPDATE & DELETE

HEADLESS CMS

A WAY TO STORE DATA

A CRUD UI

AN API TO ACCESS THE DATA

WordPress

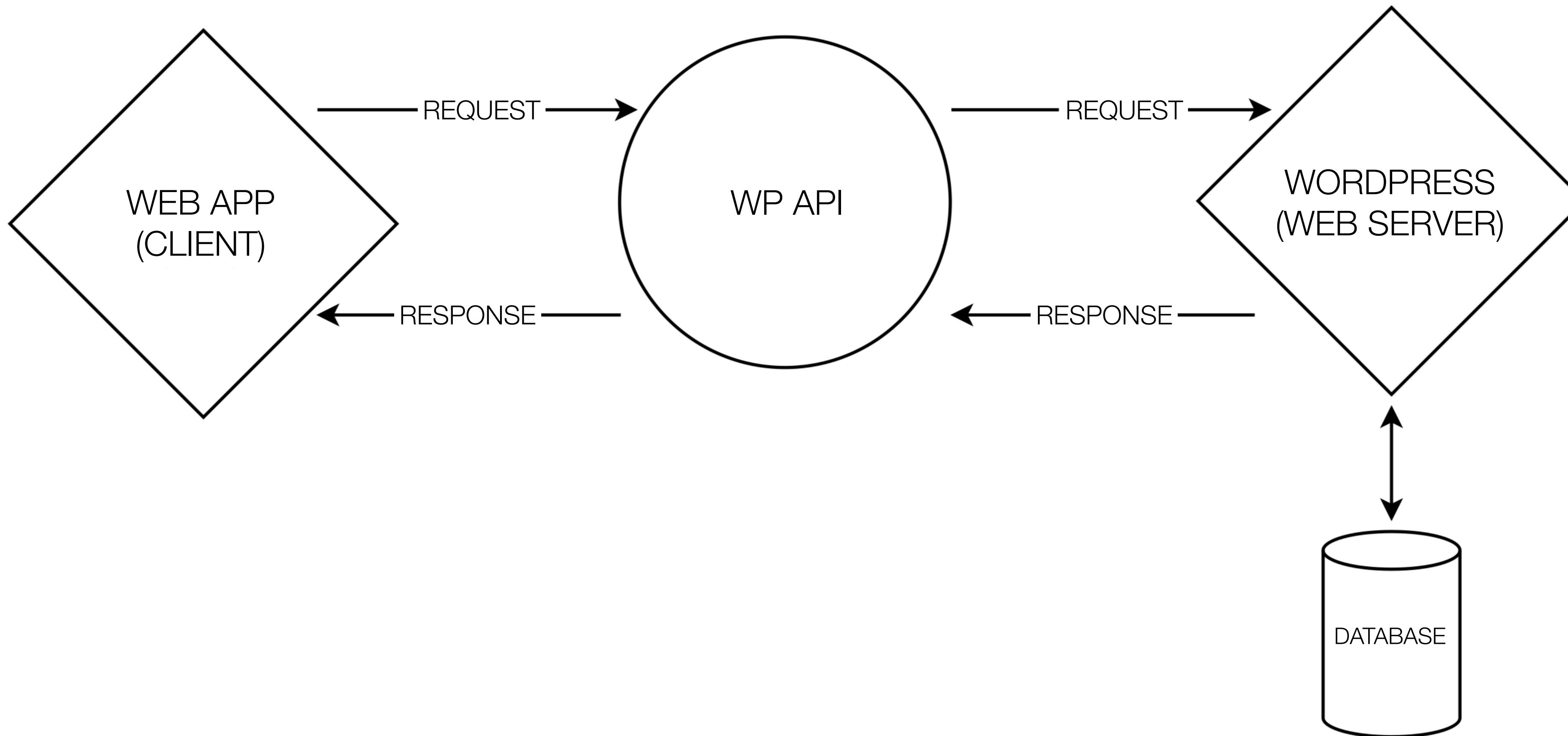
FRONTEND

API

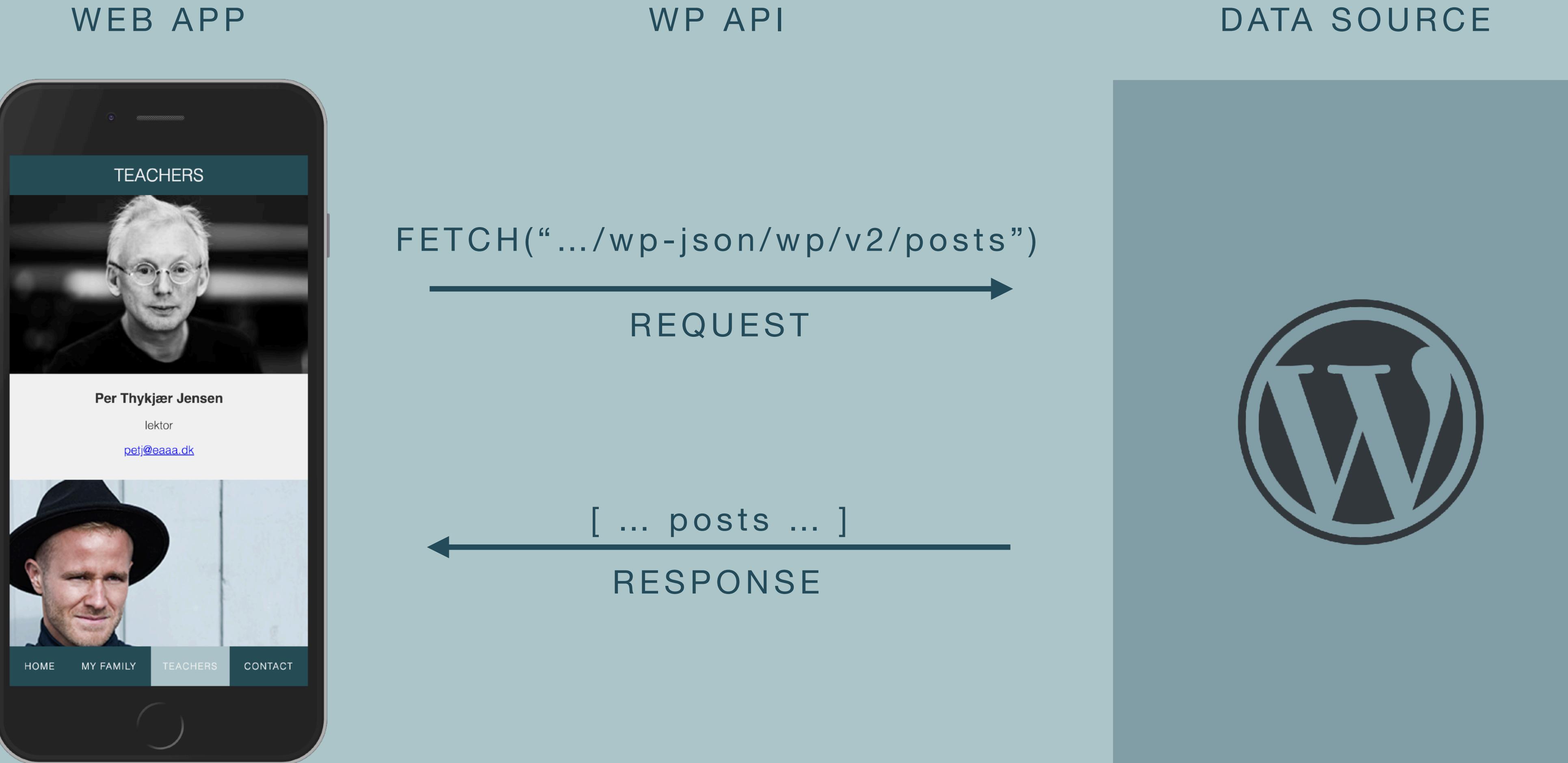
CMS

BACKEND

APPLICATION PROGRAMMING INTERFACE



Fetch & Headless CMS



Posts < MY HEADLESS CMS — +

← → ⌘ 🔒 headlesscms.cederdorff.com/wp-admin/edit.php

MY HEADLESS CMS 27 + New Howdy, mdu-front

Dashboard Posts All Posts Add New Categories Tags

WordPress 5.8 is available! Please notify the site administrator.

Posts Add New

All (8) | Published (8) | Trash (2)

Bulk actions ▾ Apply All dates ▾ All Categories ▾ Filter

8 items

<input type="checkbox"/> Title	Author	Categories	Tags		Date
Martin	admin	Persons	—	 1	Published 2020/08/31 at 6:37 am
Peter Cederdorff	admin	Family Members	—	—	Published 2019/02/07 at 4:14 pm
Rasmus Cederdorff	admin	Family Members	—	 3	Published 2019/02/07 at 4:12 pm
Alicia	admin	Family Members	—	 3	Published 2019/02/07 at 11:34 am
Per Thykjær	admin	Teachers	—	 6	Published 2019/02/06 at 7:40 am
Rasmus Cederdorff	admin	Teachers	—	 4	Published 2019/02/05 at 6:35 pm
Michael Hvidtfeldt	admin	Teachers	—	 2	Published 2019/02/05 at 6:34 pm
Birgitte Kirk Iversen	admin	Teachers	—	 8	Published 2019/02/05 at 6:32 pm
<input type="checkbox"/> Title	Author	Categories	Tags		Date

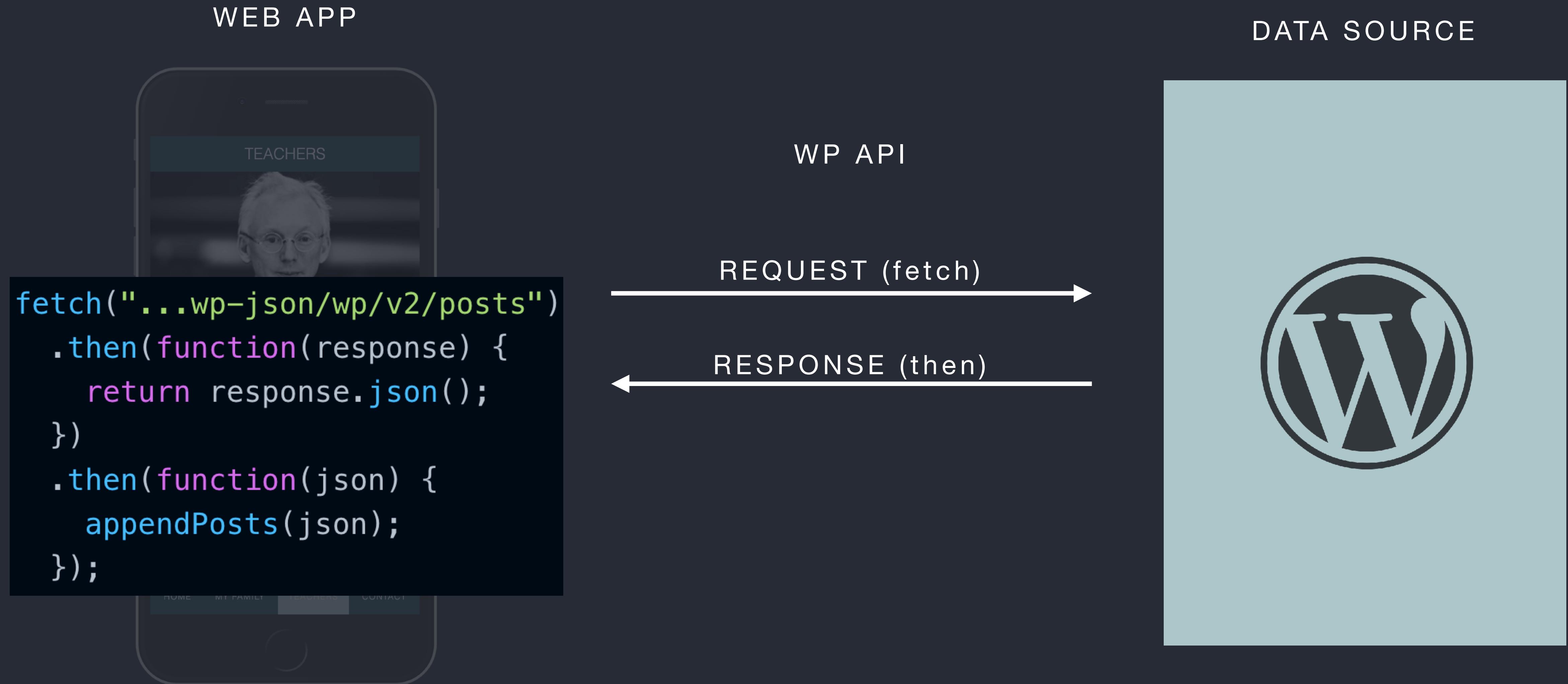
8 items

Bulk actions ▾ Apply

Thank you for creating with [WordPress](#).

Version 5.5.5

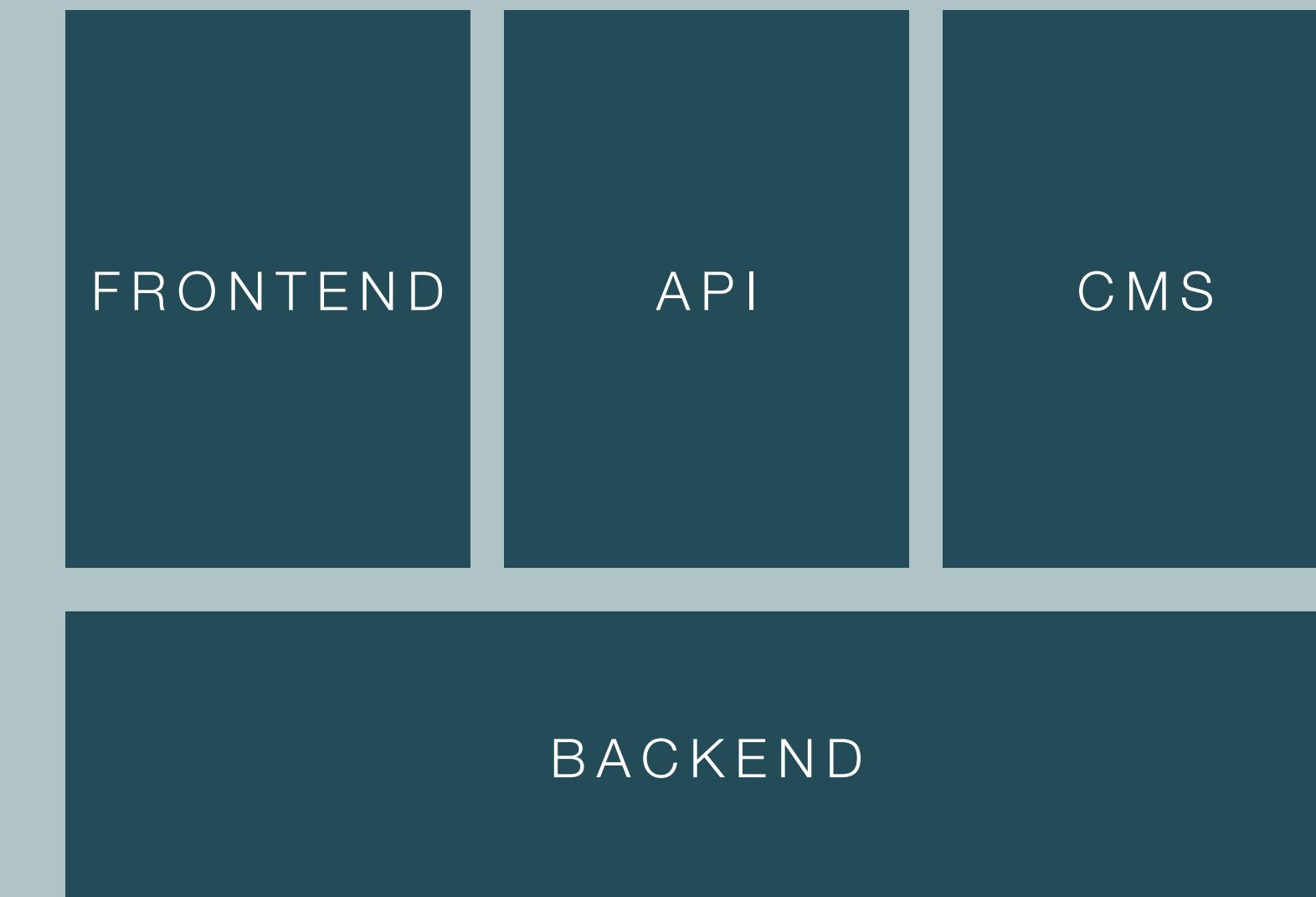
Fetch & Headless CMS



WEB APP



HEADLESS WP



movie-api.cederdorff.com

Fetch posts

Screenshot of the WordPress admin dashboard showing the Posts screen.

The dashboard header includes the WordPress logo, site title "Movie API", a refresh icon with "7", a comment icon with "0", and a "+ New" button.

A notification bar at the top right says "WordPress 5.3.2 is available! [Please update now.](#)"

The left sidebar menu is visible, with "Posts" selected and highlighted in blue. Other menu items include Dashboard, All Posts, Add New, Categories, Tags, Media, Pages, Comments, Appearance, Plugins (with a red "3" badge), Users, Tools, Settings, Custom Fields, and Collapse menu.

The main content area is titled "Posts" with a "Add New" button. It shows a list of published posts:

	Title	Author	Categories	Tags
<input type="checkbox"/>	Frozen	admin	Animation, Comedy	—
<input type="checkbox"/>	Don't Let Go	admin	Drama, Horror	—
<input type="checkbox"/>	Lion King	admin	Animation	—
<input type="checkbox"/>	Title ▲			

Below the table, there are "Bulk Actions" dropdown and "Apply" buttons.

Fetch posts

```
let _movies = [];

// fetch all movies from WP
async function getMovies() {
  let response = await fetch("https://movie-api.cederdorff.com/wp-json/wp/v2/posts?_embed");
  let data = await response.json();
  _movies = data;
  appendMovies(data);
  showLoader(false);
}

getMovies();
```

API Routes

Meta data: <http://your-domain.com/wp-json/>

Posts: <http://your-domain.com/wp-json/wp/v2/posts>

Pages: <http://your-domain.com/wp-json/wp/v2/pages>

Media: <http://your-domain.com/wp-json/wp/v2/media>

Types: <http://your-domain.com/wp-json/wp/v2/types>

Taxonomies: <http://your-domain.com/wp-json/wp/v2/taxonomies>

Categories: <http://your-domain.com/wp-json/wp/v2/categories>

Tags: <http://your-domain.com/wp-json/wp/v2/tags>

Users: <http://your-domain.com/wp-json/wp/v2/users>

Comments: <http://your-domain.com/wp-json/wp/v2/comments>

DOCS: <https://developer.wordpress.org/rest-api/>

Filter, sort & order

[https://api.cederdorff.com/wp-json/wp/v2/posts? embed](https://api.cederdorff.com/wp-json/wp/v2/posts?embed)

[https://api.cederdorff.com/wp-json/wp/v2/posts?
embed&categories=2](https://api.cederdorff.com/wp-json/wp/v2/posts?embed&categories=2)

[https://api.cederdorff.com/wp-json/wp/v2/posts?
orderby=date&order=desc&after=2019-00-10T17:00:00](https://api.cederdorff.com/wp-json/wp/v2/posts?orderby=date&order=desc&after=2019-00-10T17:00:00)

[https://api.cederdorff.com/wp-json/wp/v2/posts?
orderby=date&order=asc](https://api.cederdorff.com/wp-json/wp/v2/posts?orderby=date&order=asc)

Recap on JS

More on Fetch, JSON, Objects, Arrays, Loops,
Functions, backticks and DOM Manipulation

<https://cederdorff.com/race/slides/js-concepts.pdf>



Code
Every
Day