

# Fetch, JSON, CRUD & REST

```
index.html
85 }
86 }
87 /*
88 * Fetches post data from my headless cms
89 */
90 function getPersons() {
91   fetch('http://headlesscms.cederdorff.com/wp-json/wp/v2/posts?_embed')
92     .then(function(response) {
93       return response.json();
94     })
95     .then(function(persons) {
96       appendPersons(persons);
97     });
98 }
99 /*
100 * Appends json data to the DOM
101 */
102 function appendPersons(persons) {
103   let htmlTemplate = '';
104   for (let person of persons) {
105     console.log(person);
106     htmlTemplate += `

107       
108       <h4>${person.title.rendered}
109       <p>${person.acf.age} years old
110       <p>Hair color: ${person.acf.hairColor}
111       <p>Relation: ${person.acf.relation}
112     </article>`;
113   }
114   document.querySelector("#family-members").innerHTML += htmlTemplate;
115 }
116


```

# Content

- Client-Server Model og HTTP
- Web Development
- Fetch
- Async JS & await/sync
- What's a Data Source?
- API
- JSON
- CRUD
- REST
- HTTP Request Methods & Verbs
- CRUD vs REST & HTTP Verbs
- What's Firebase?

# Let's turn on the Database!



# Frontend (client)

CRUD App

Lars Bogetoft  
Head of Education  
[larb@kea.dk](mailto:larb@kea.dk)  
**UPDATE**   **DELETE**

Peter Lind  
Senior Lecturer  
[petl@kea.dk](mailto:petl@kea.dk)  
**UPDATE**   **DELETE**

Magdalena "Lenka" Otap  
Lecturer  
[mago@kea.dk](mailto:mago@kea.dk)  
**UPDATE**   **DELETE**

Rasmus Cederdorff  
Senior Lecturer  
[race@kea.dk](mailto:race@kea.dk)  
**UPDATE**   **DELETE**

Create a new User

Magdalena "Lenka" Otap  
Lecturer  
mago@kea.dk  
<https://share.cederdorff.com/images/mago.jpg>

**CREATE USER**

Update User

<https://cederdorff.github.io/dat-js-crud-intro/>

# Backend (Server)

```
Raw Parsed
```

```
{  
  "-NNBQFn28wOOLJTxvma": {  
    "image": "https://kea.dk/slir/w360-c1x1/images/user-profile/chefer/larb.jpg",  
    "mail": "larb@kea.dk",  
    "name": "Lars Bogetoft",  
    "title": "Head of Education"  
  },  
  "-NNBQSO5gFb-4xB7liug": {  
    "image": "https://share.cederdorff.com/images/petl.jpg",  
    "mail": "petl@kea.dk",  
    "name": "Peter Lind",  
    "title": "Senior Lecturer"  
  },  
  "-NNBQSO5gFb-4xB7liug": {  
    "image": "https://share.cederdorff.com/images/mago.jpg",  
    "mail": "mago@kea.dk",  
    "name": "Magdalena \"Lenka\" Otap",  
    "title": "Lecturer"  
  },  
  "fTs84KRoYw5pRZEWcq2Z": {  
    "image": "https://share.cederdorff.com/images/race.jpg",  
    "mail": "race@kea.dk",  
    "name": "Rasmus Cederdorff",  
    "title": "Senior Lecturer"  
  }  
}
```

<https://race-dat-v1-default-rtdb.firebaseio.com/users.json>

Firebase CRUD, REST & MVC

User	Email	Action	Action
Morten Algy Bonderup	moab@eaaa.dk	DELETE	UPDATE
Martin Nøhr	mnor@eaaa.dk	DELETE	UPDATE
Anne Kirketerp	anki@eaaa.dk	DELETE	UPDATE
Jeffrey David Serio	jds@eaaa.dk	DELETE	UPDATE
Birgitte Kirk Iversen	bki@eaaa.dk	DELETE	UPDATE
Rasmus Cederdorff	race@eaaa.dk	DELETE	UPDATE

Firebase CRUD, REST & MVC

Update User

Name	Email	Profile Picture
Morten Algy Bonderup	moab@eaaa.dk	
Birgitte Kirk Iversen	bki@eaaa.dk	
Jes Arbov	jeab@eaaa.dk	
Maria Louise Bendixen	mlbe@mail.dk	

CANCEL

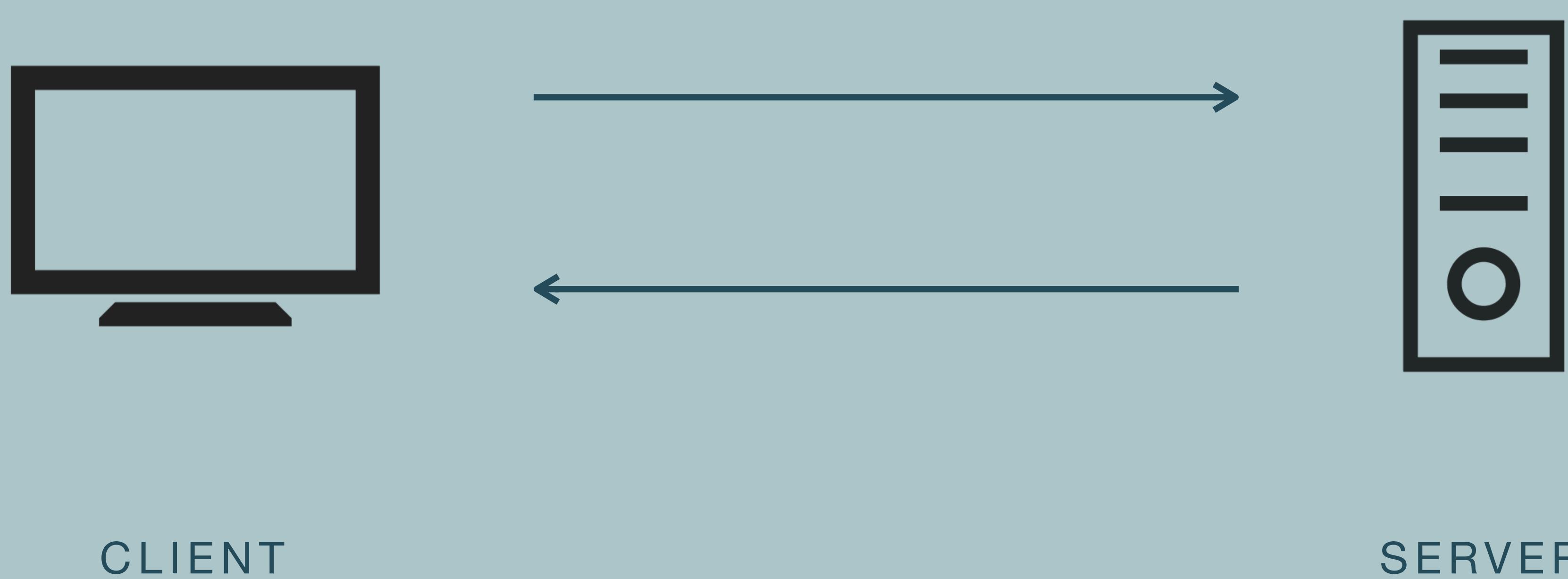
UPDATE USER

[cederdorff.github.io/mdu-frontend/firebase-crud-rest-mvc](https://cederdorff.github.io/mdu-frontend/firebase-crud-rest-mvc)

<https://cederdorff.github.io/dat-js-crud-intro/>

# Client-Server Model

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



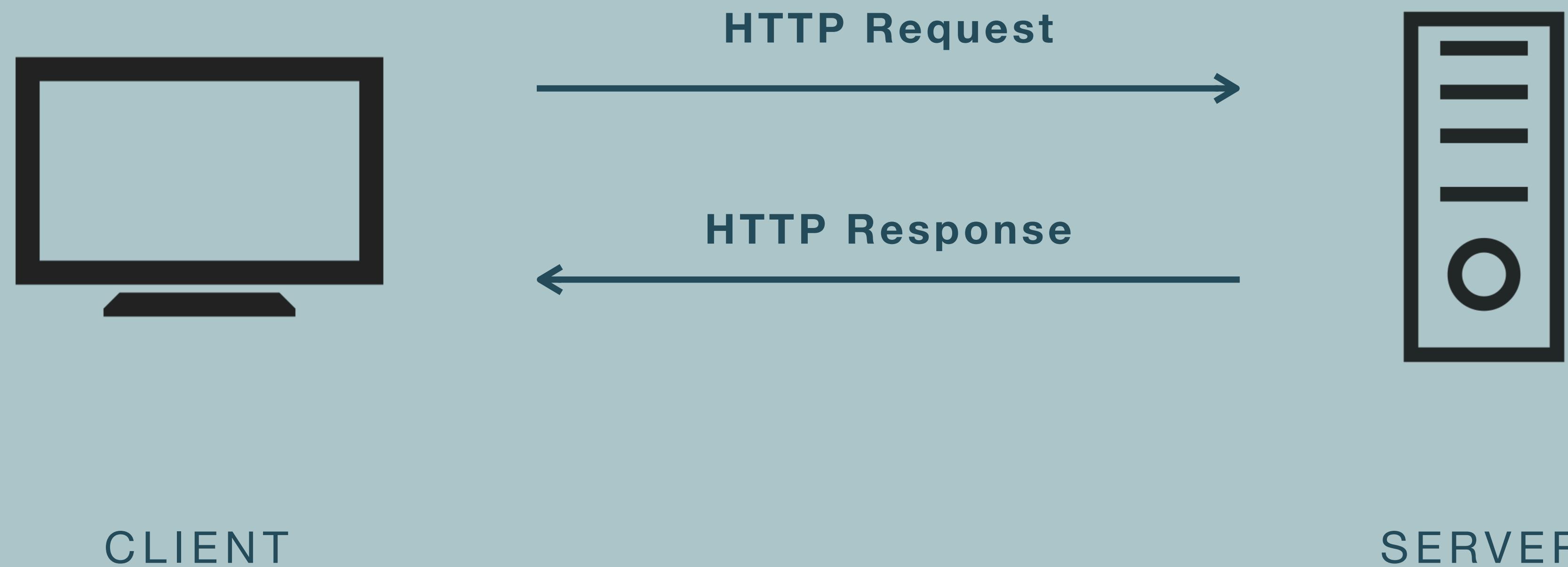
# Client-Server Model

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

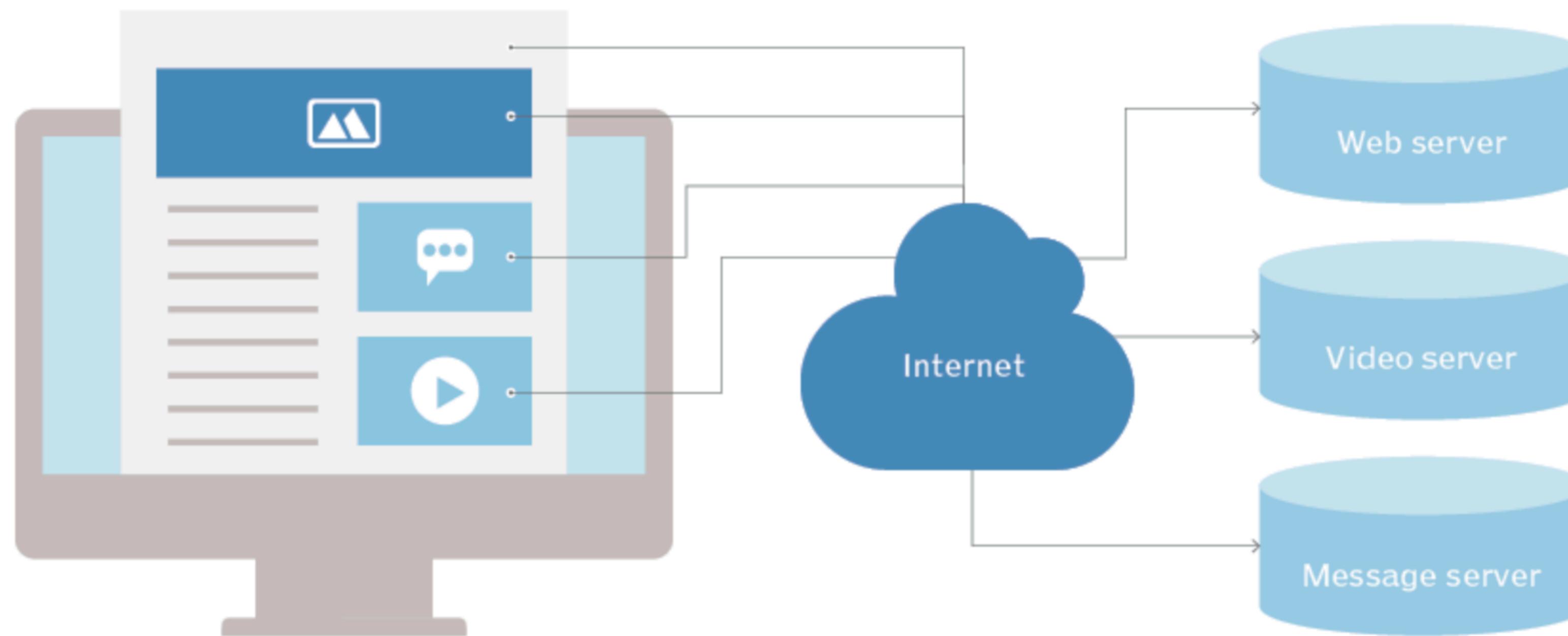
---

## 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 displays a web browser window with the URL [kompetence.kea.dk/kurser-fag/webudvikling-frontend](https://kompetence.kea.dk/kurser-fag/webudvikling-frontend). The main content area shows the title "WEBUDVIKLING FRONTEND" and a "SE HOLDSTART OG TILMELD DIG →" button. On the left, there's a sidebar with links like "FORSIDE / KURSER OG FAGMODULER / WEBUDVIKLING FRONTEND" and a "LAV DIN EGEN PDF-BROCHURE" section with a "FØJ TIL PDF" button. A red callout box highlights the "NY UDDANNELSE: DIPLOM I WEBUDVIKLING" section, which describes the diploma and its focus on modern web development technologies like HTML, CSS, and React.

The Network tab in the developer tools shows the following resource list:

Name	Status	Type	Initiator	Size	Time	Waterfall
webudvikling-fro...	200	do...	Other	27....	90...	
quixtrap.css	200	styl...	webudvi...	(dis...)	2 ms	
quix.css	200	styl...	webudvi...	(dis...)	1 ms	
jquery.min.js?77...	200	script	webudvi...	(m...)	0 ms	
style.css?77958...	200	styl...	webudvi...	(dis...)	2 ms	
jquery-noconflic...	200	script	webudvi...	(m...)	0 ms	
jquery-migrate.m...	200	script	webudvi...	(m...)	0 ms	
front.css?77958...	200	styl...	webudvi...	(dis...)	2 ms	
script.js?779581...	200	script	webudvi...	(m...)	0 ms	
core.js?7795818...	200	script	webudvi...	(m...)	0 ms	
keepalive.js?779...	200	script	webudvi...	(m...)	0 ms	
content.css?779...	200	styl...	webudvi...	(dis...)	2 ms	
bootstrap.min.js	200	script	webudvi...	(m...)	0 ms	
font-awesome.m...	200	styl...	webudvi...	(dis...)	2 ms	
kea-min.js?1.0.28	200	script	webudvi...	(m...)	0 ms	
logo-main-black...	200	png	webudvi...	(m...)	0 ms	
css2?family=Po...	200	styl...	webudvi...	(dis...)	2 ms	
logo-small.png	200	png	webudvi...	(m...)	0 ms	
template.css?1.0...	200	styl...	webudvi...	(dis...)	2 ms	
finder.css?77958...	200	styl...	webudvi...	(dis...)	2 ms	
.	200	.	.	.	.	

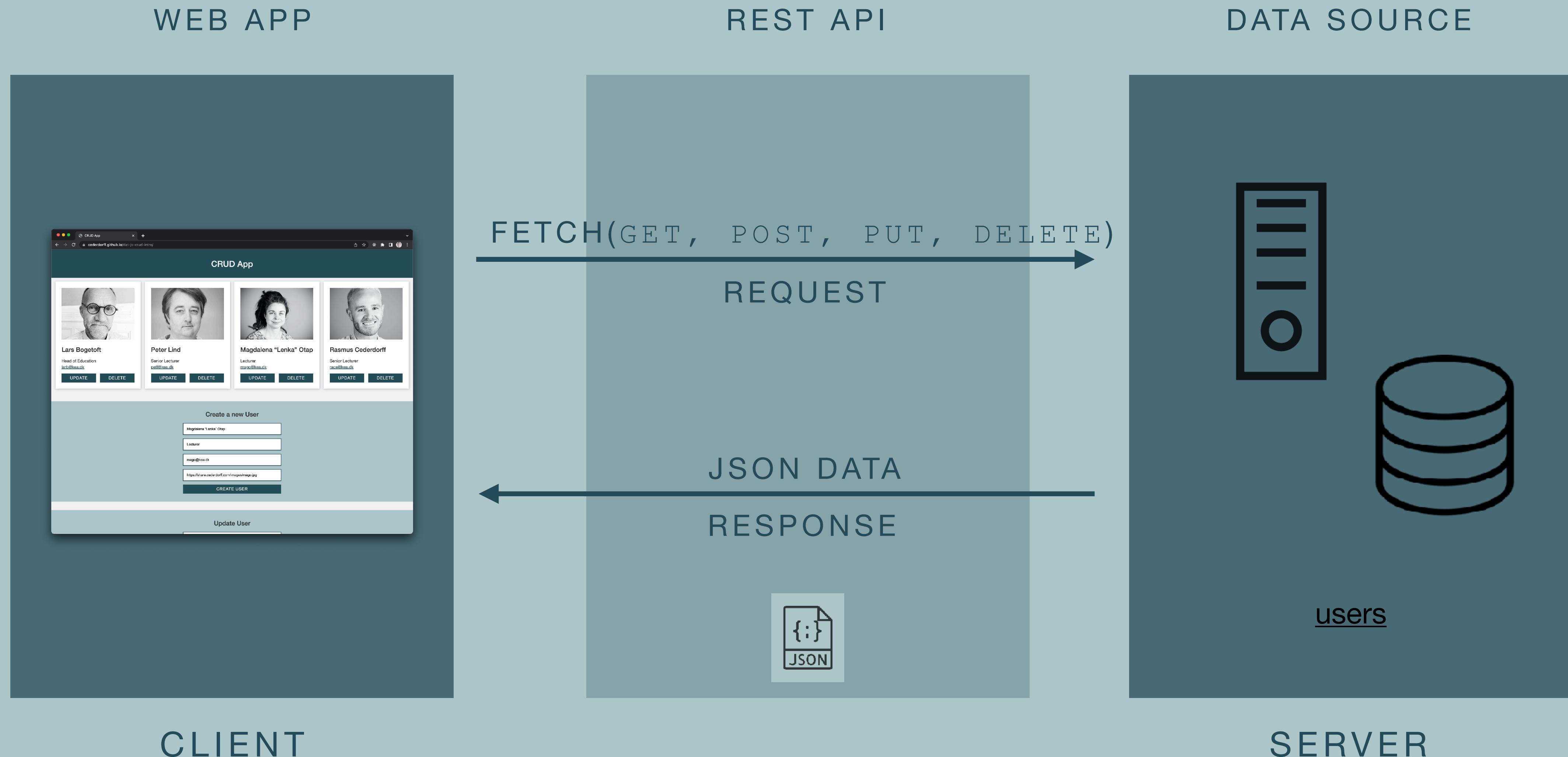
At the bottom of the Network tab, it says "46 requests | 28.7 kB transferred | 1.9 MB resources | Finish: 1.59 s | DOMContentLoad".

- Brug Network-tabben til at undersøge et website (fx [kea.dk](http://kea.dk), [dr.dk](http://dr.dk), [google.dk](http://google.dk), [react-rest-and-auth.web.app](http://react-rest-and-auth.web.app) eller et helt andet).
- Åben websitet i Chrome (eller en anden browser).
- Åben Developer Tool (se: [How to open the dev tool in your browser](#)) og gå til Network/Netværk.
- Genindlæs siden imens du står i Network-tabben og undersøg hvilke ressourcer, der bliver hentet.
- Hvilke(n) type ressourcer er der tale om?
- Overvej hvordan ressourcerne hentes.

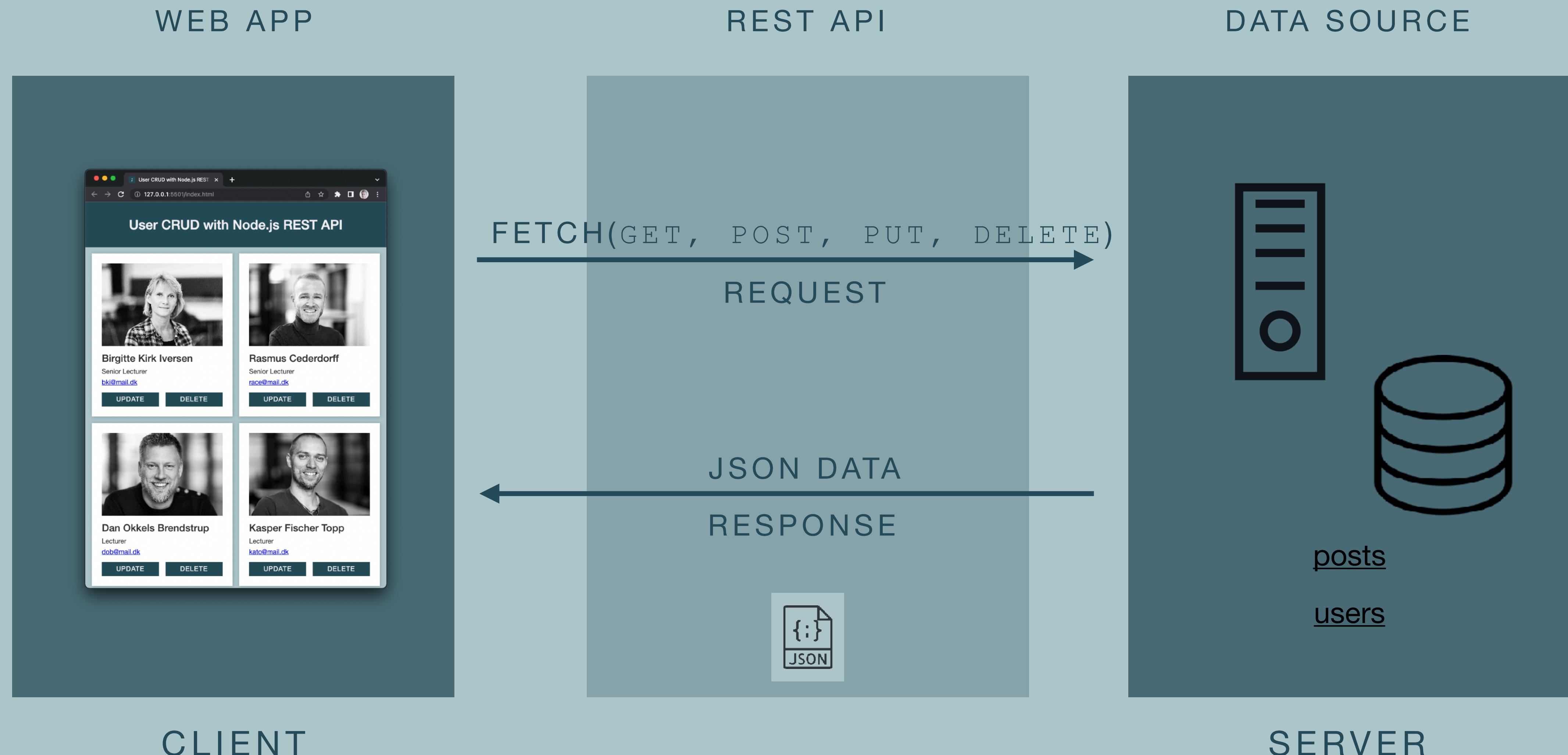


## Network-tabben

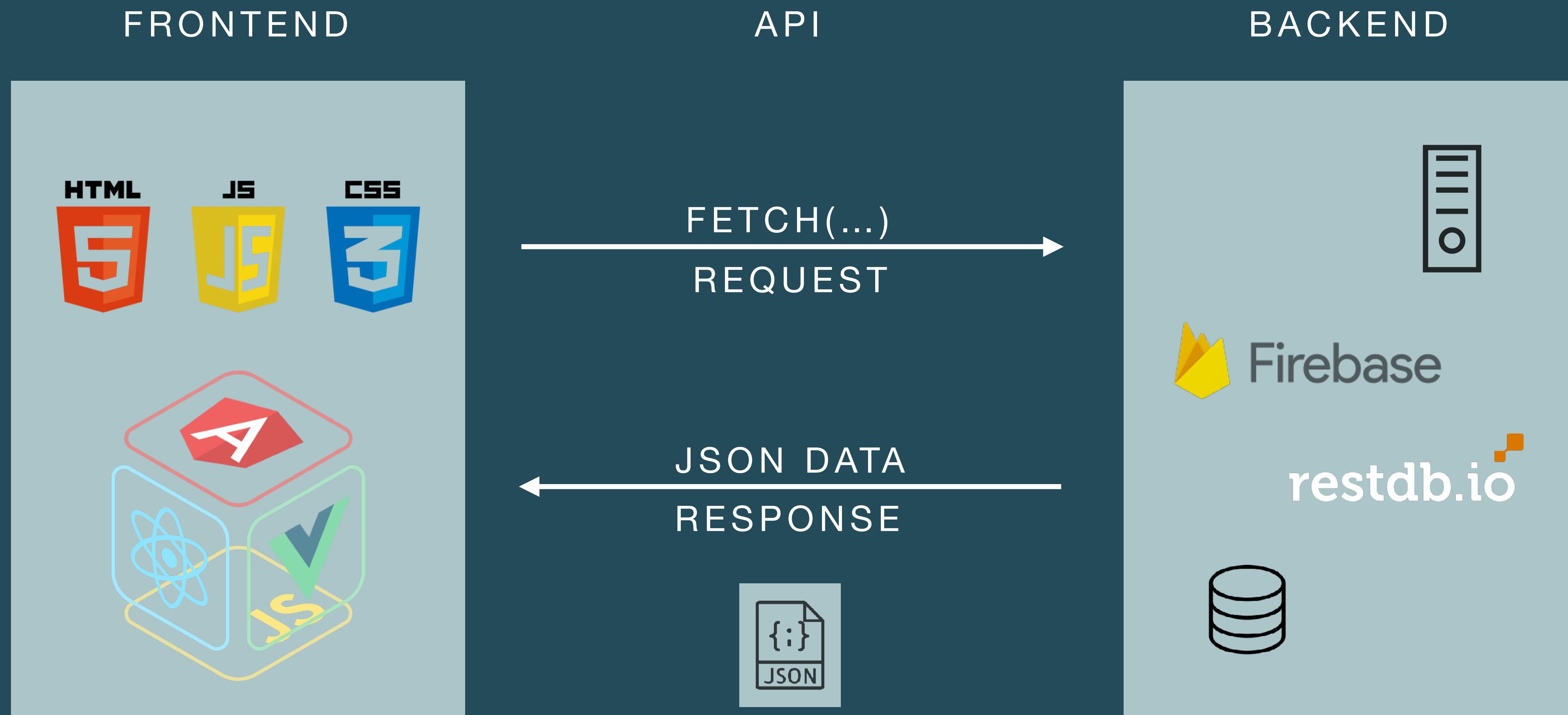
# Web Development



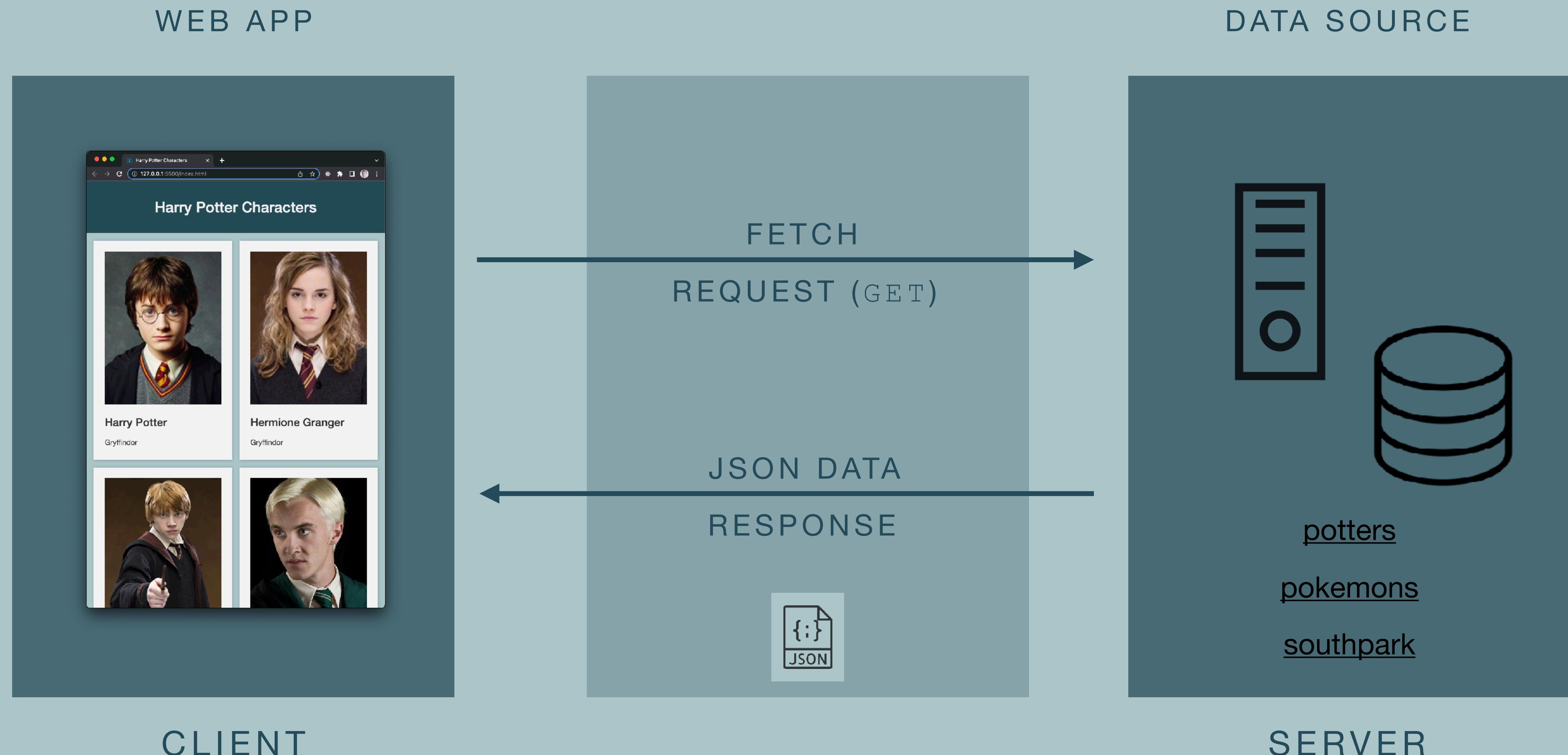
# Fetch, HTTP Request & Response



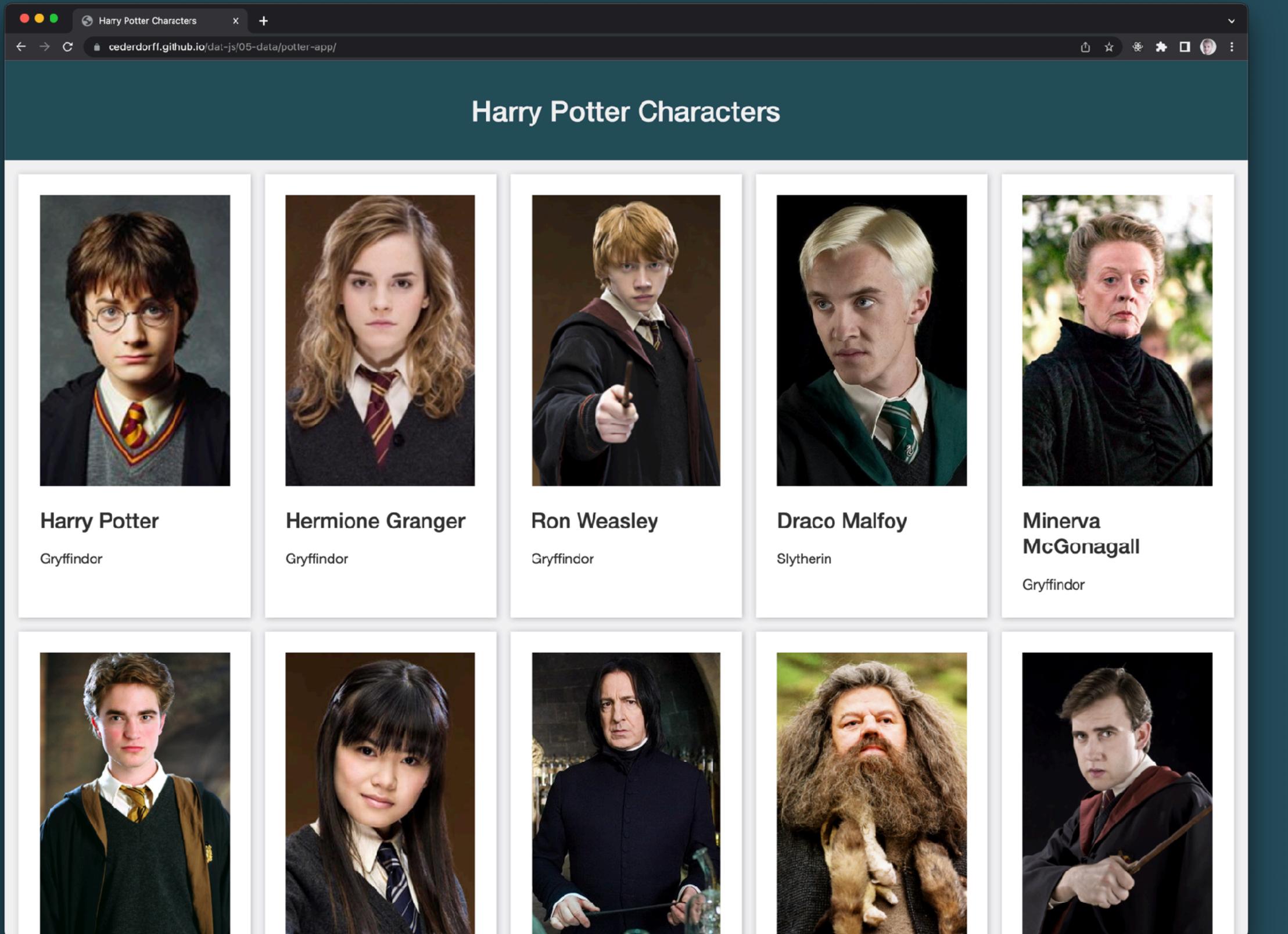
# Web Development



# Fetch, HTTP Request & Response



# Frontend (client)



# JSON Data Source (Server)

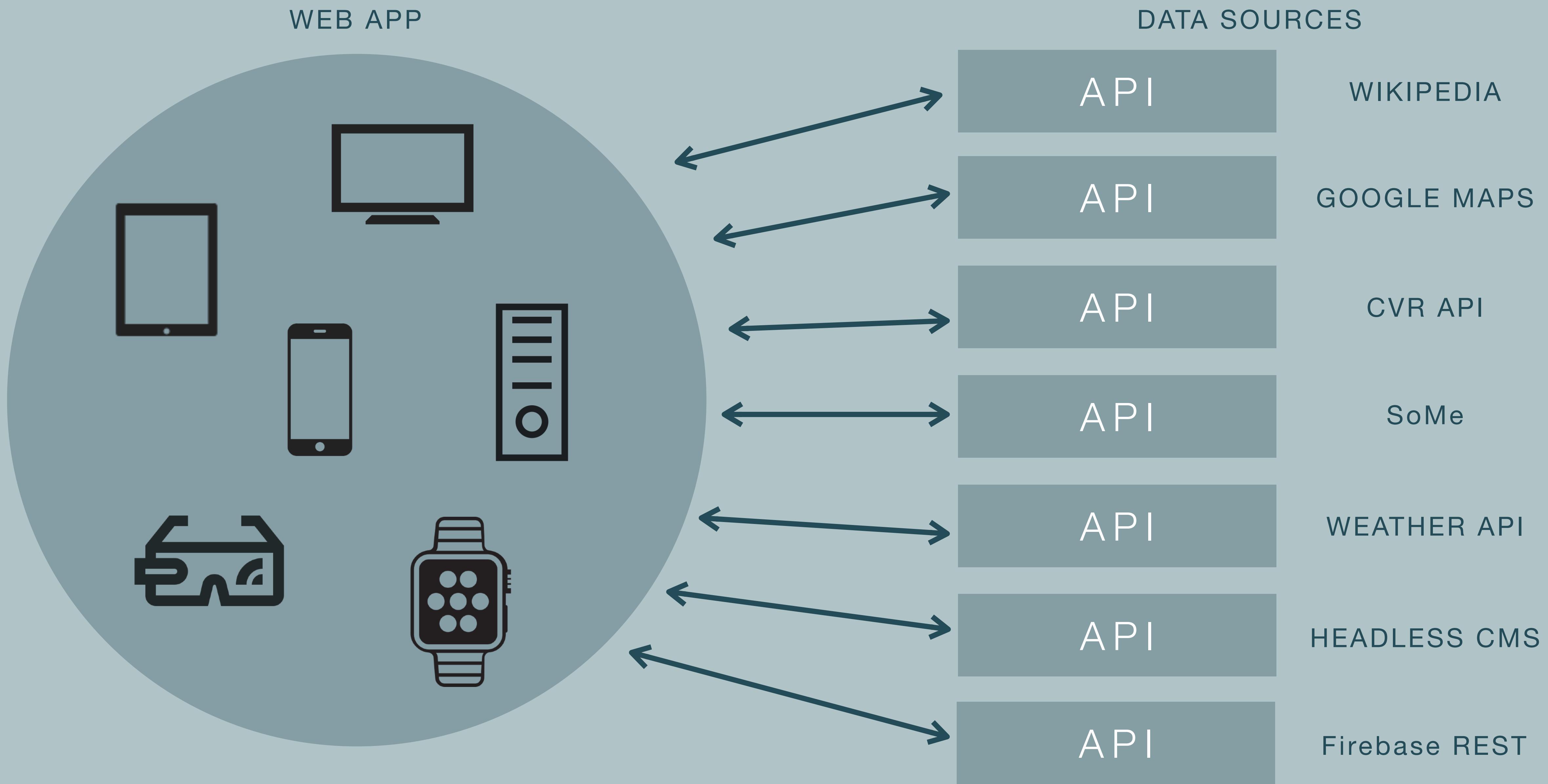
The screenshot shows a browser window displaying a JSON file. The JSON structure represents the Harry Potter characters. Each character is defined by a key-value pair where the key is the character's name and the value is a JSON object containing details like species, gender, house, date of birth, ancestry, eye and hair color, and wand information. The "Raw" tab is selected, showing the raw JSON code, while the "Parsed" tab shows the JSON tree structure.

```
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
  }
},
```

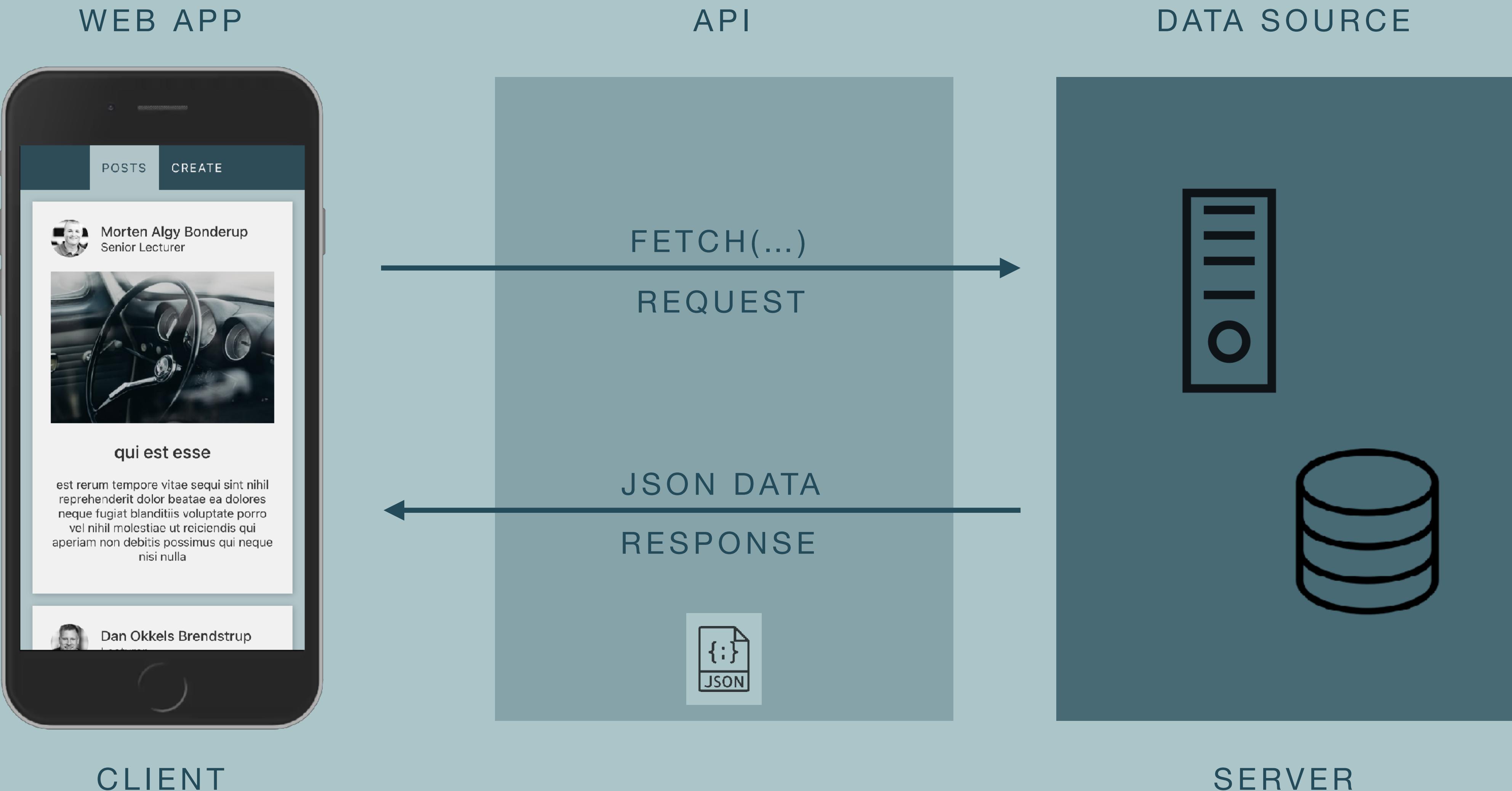
<https://cederdorff.github.io/dat-js/05-data/potter-app/>

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

# API



# Web Development



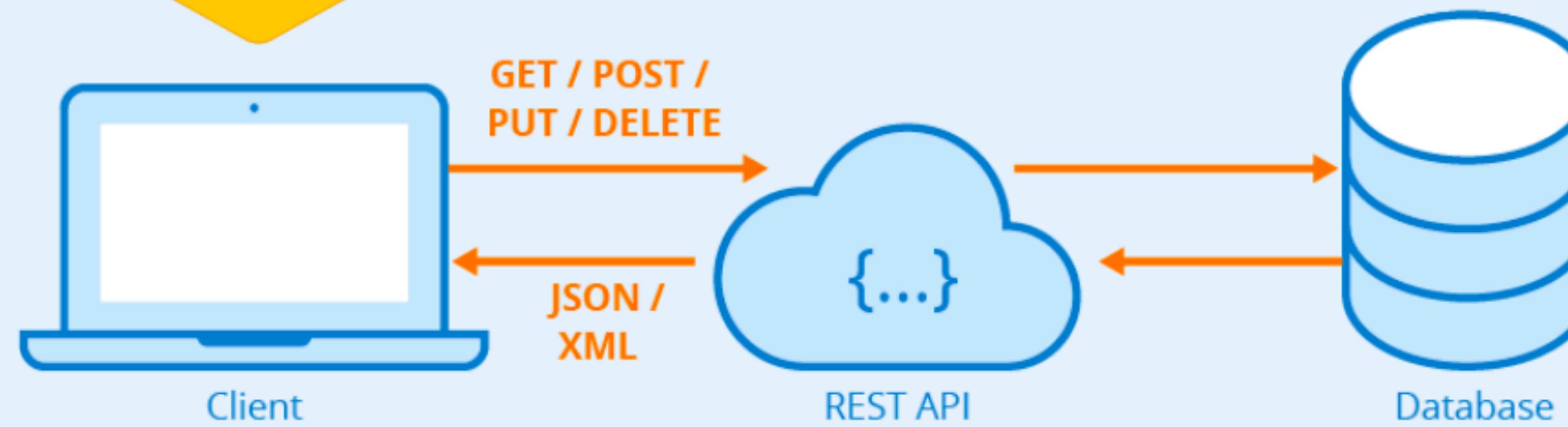
When someone asks you how to get data from a database in a js code



made with mematic

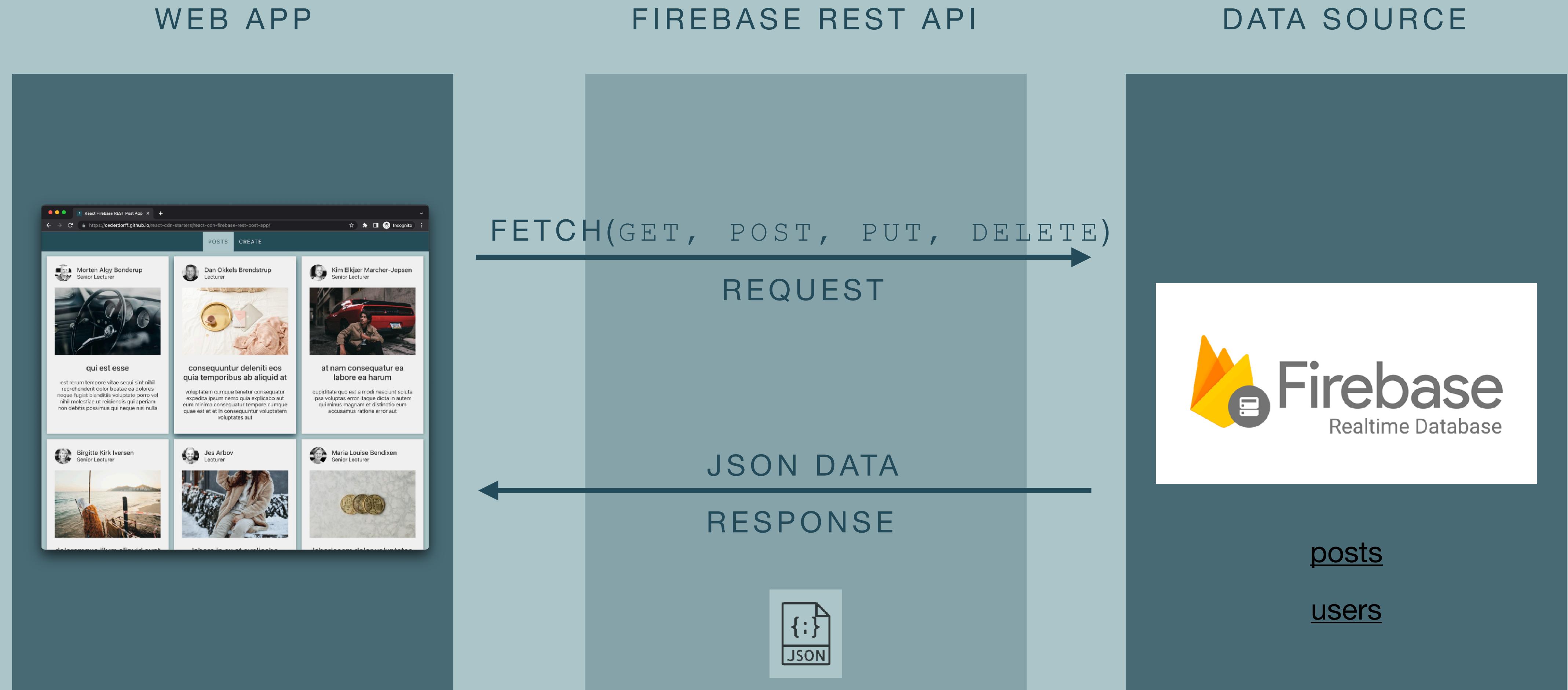


# Firebase





# 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(...)

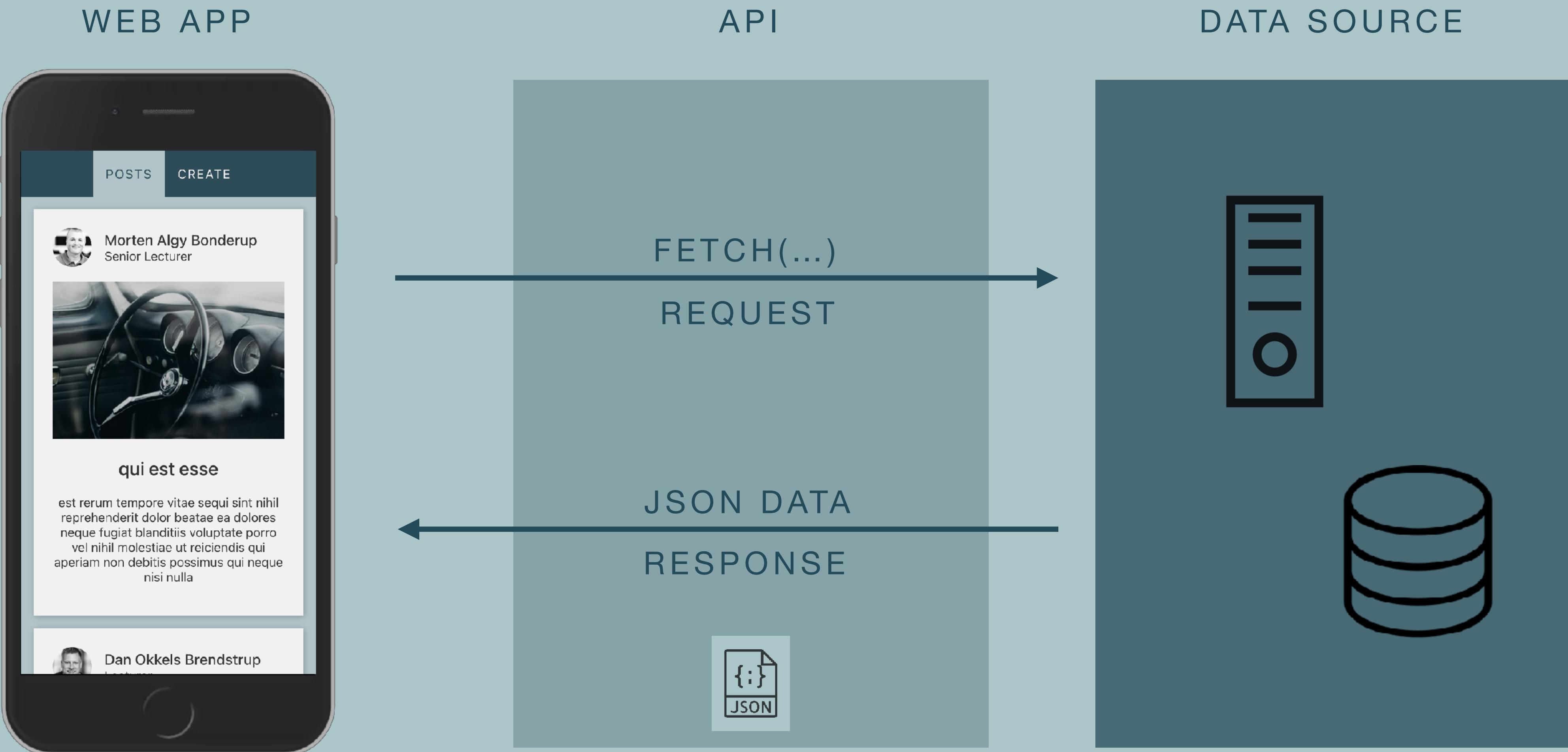
... get & post data from and to a data source  
... can perform network requests to a server

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

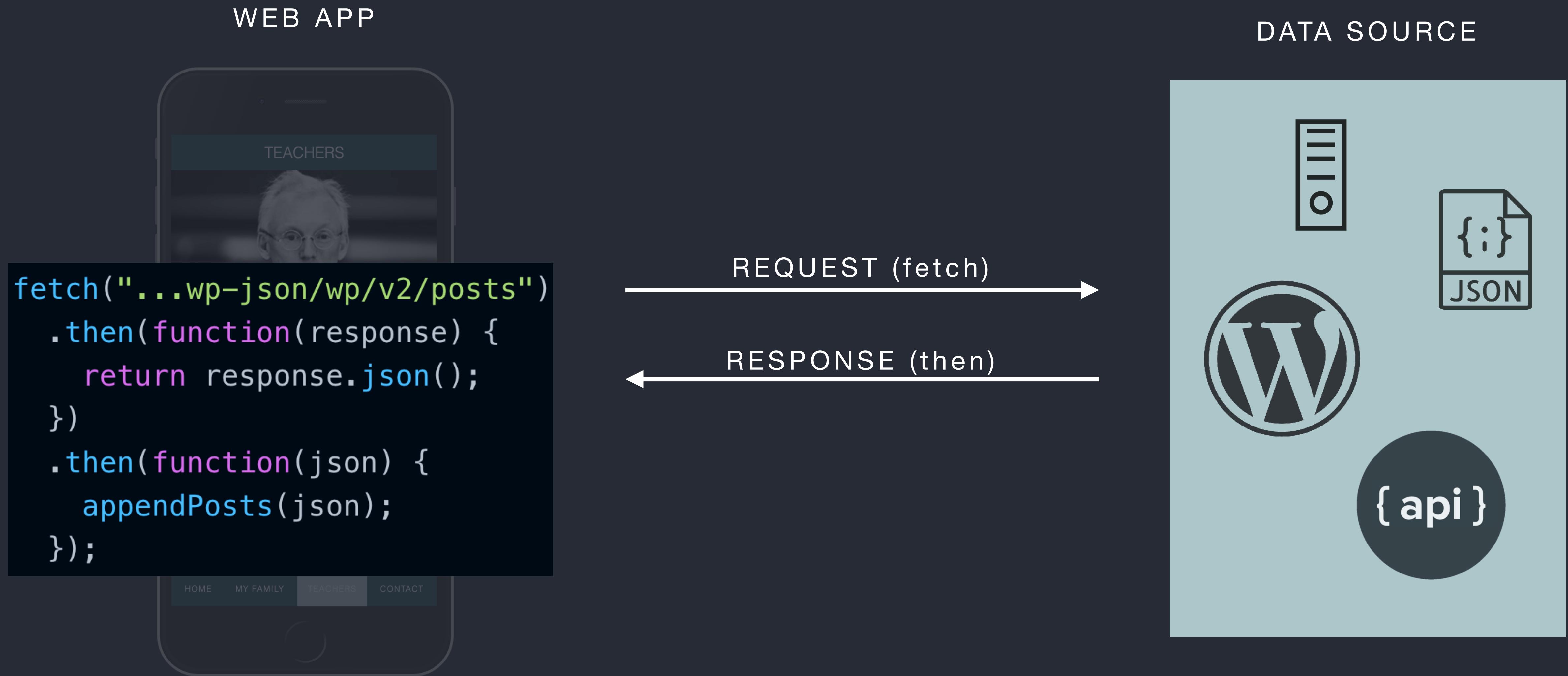
- `url` – the URL to access.
- `options` – optional parameters: method, headers etc.

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;
}

```



```
[
  {
    "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)

app.js — web-diplom-frontend

```
JS app.js x
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

The diagram illustrates the flow of data. A curved arrow originates from the URL in the fetch statement at line 5 of the app.js code and points to the JSON response shown in the browser window. Another curved arrow originates from the JSON response in the browser and points back to the app.js code, specifically to the 'persons' variable assignment at line 7.

https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/\_data/persons.json

Raw Parsed

```
[{"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": ""}]
```

fetch-persons-grid

## request (fetch)

The diagram illustrates the flow of data from the `app.js` code to the final JSON response. A curved arrow labeled "request (fetch)" points from the `fetch` call in the `getPerson` function to the browser window. Another curved arrow labeled "response (JSON)" points from the `response.json` call to the JSON data displayed 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 fetch request. The URL is `https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json`. The response is a JSON array containing 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/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": ""}]
```

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.

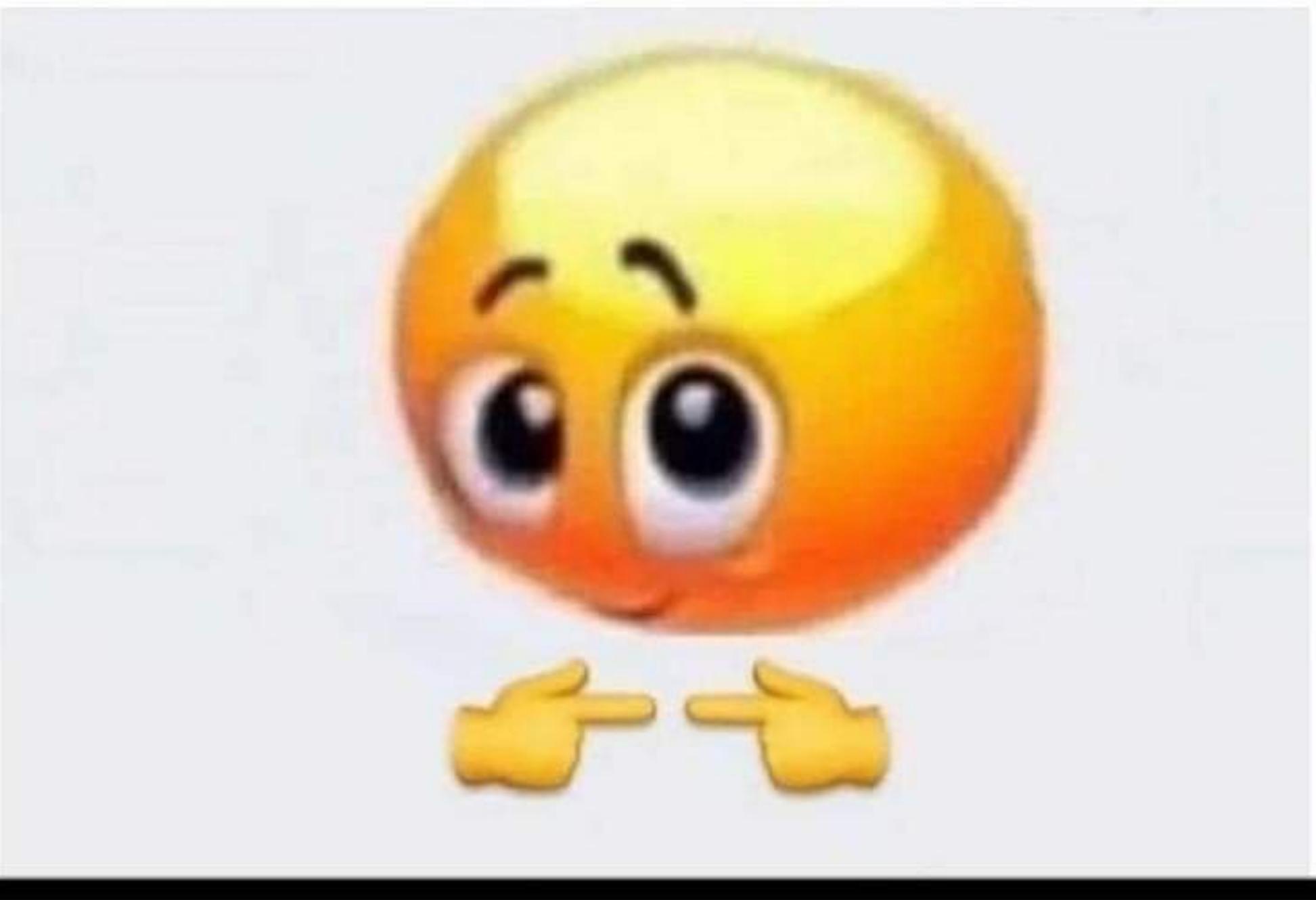
```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json")
  .then(function (response) {
    return response.json();
  })
  .then(function (data) {
    console.log(data);
  });

```

# 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](https://www.w3schools.com/js/js_async.asp)

# Fetch returns a promise

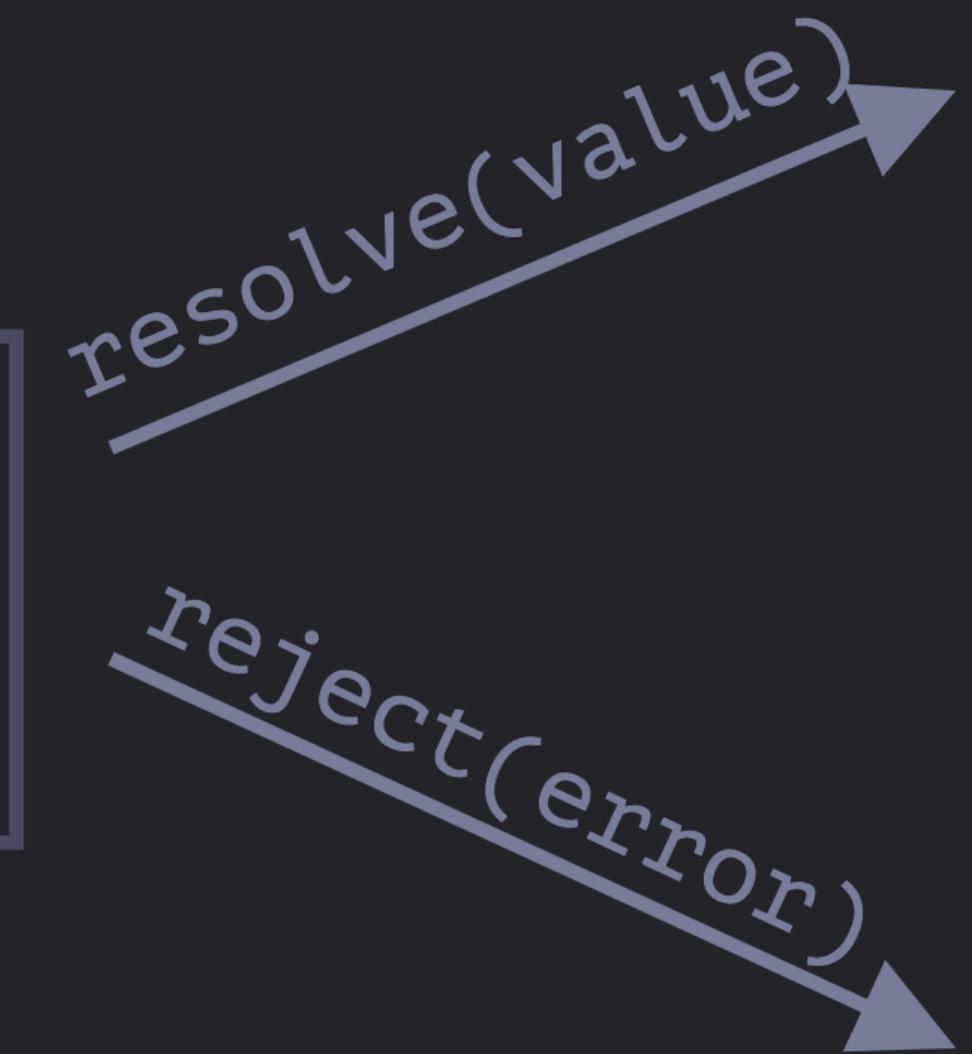
We can use `await` to wait for `fetch` to finish



[https://www.w3schools.com/js/js\\_async.asp](https://www.w3schools.com/js/js_async.asp)

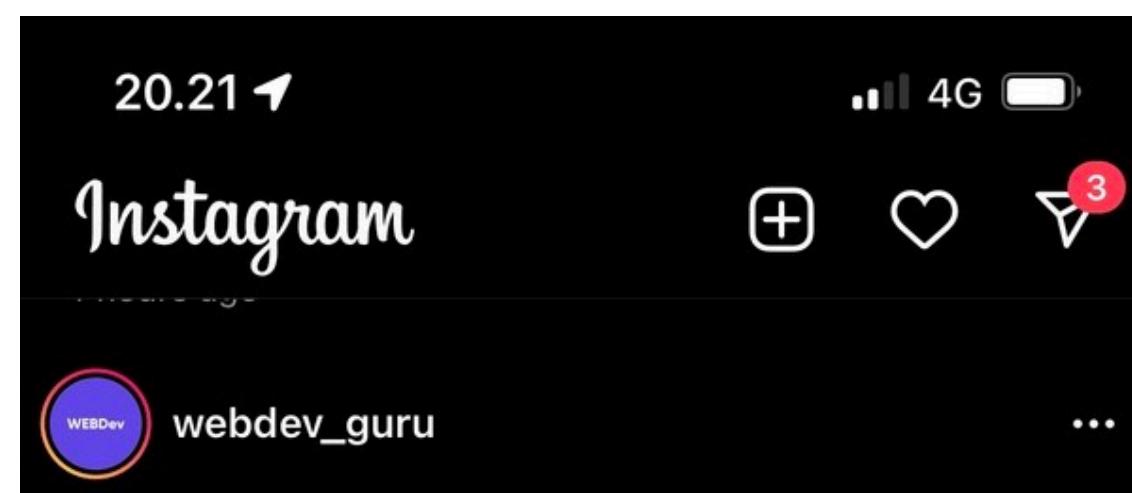
```
new Promise(executor)
```

state: "pending"  
result: undefined



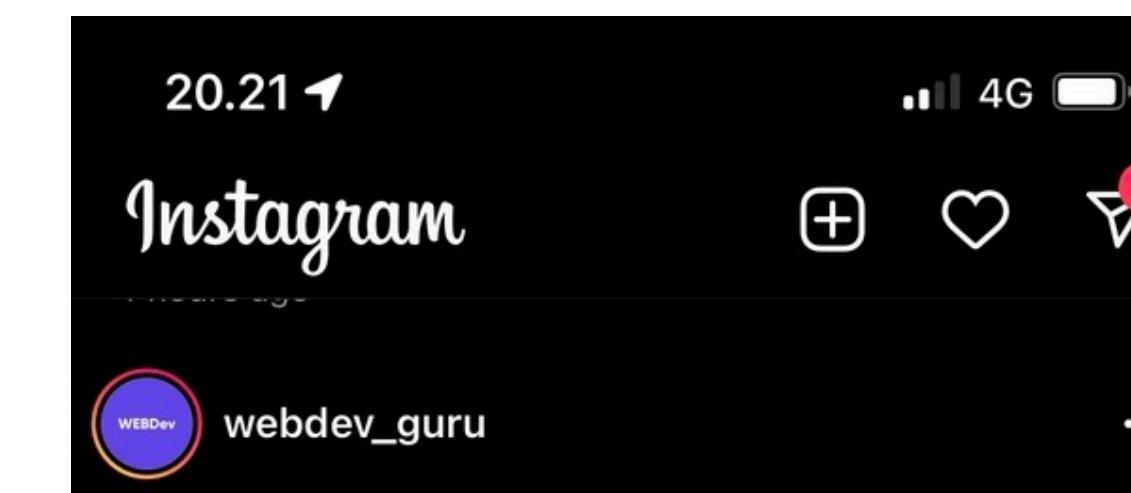
state: "fulfilled"  
result: value

state: "rejected"  
result: error



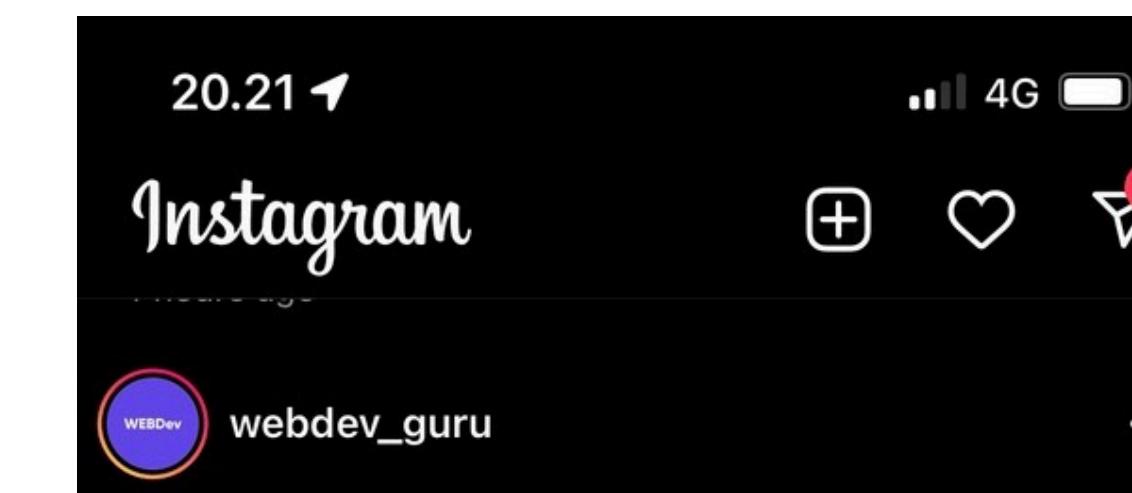
Developer Tips

02



Developer Tips

03



Developer Tips

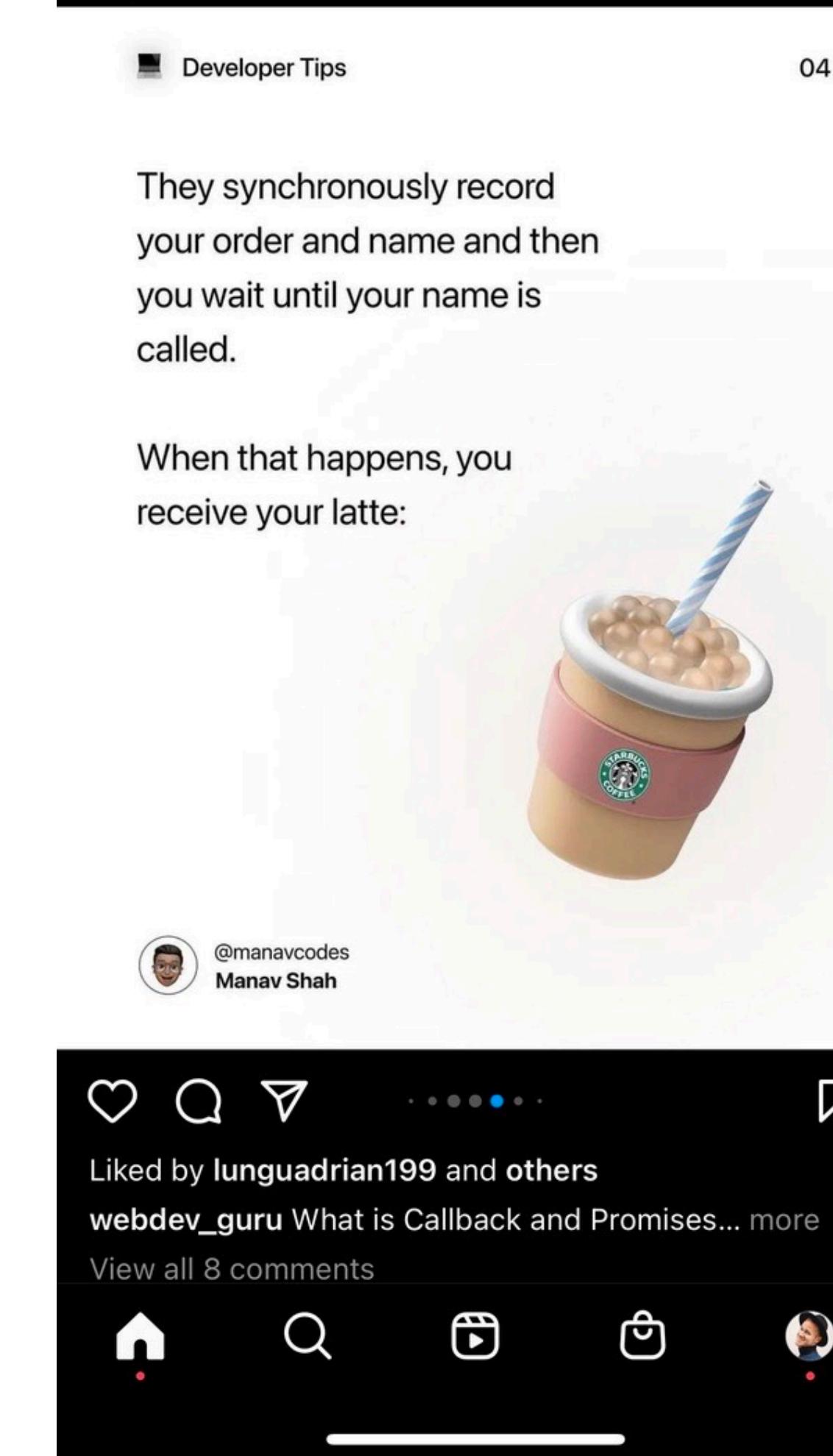
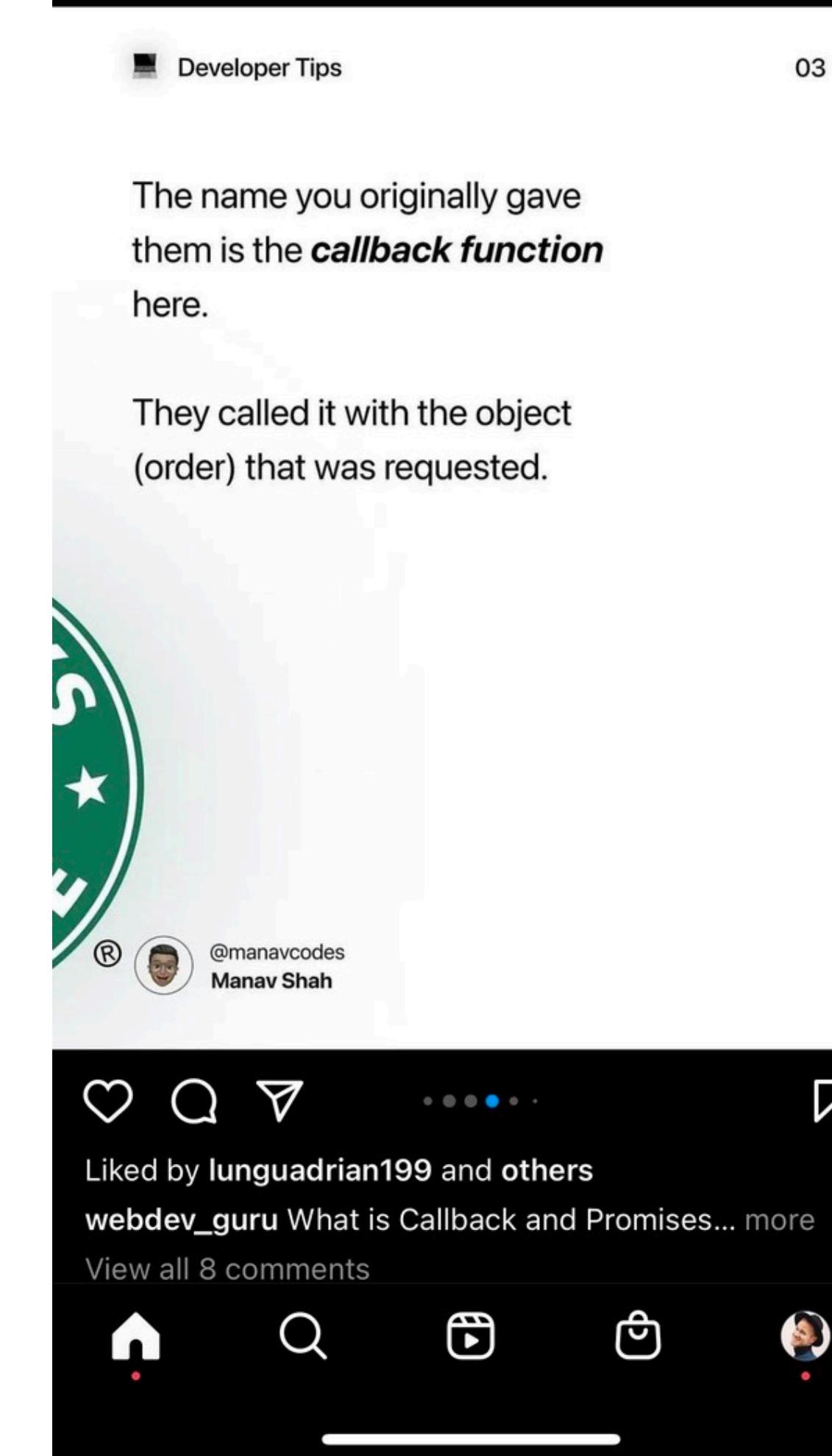
04

The name you originally gave them is the **callback function** here.

They called it with the object (order) that was requested.

They synchronously record your order and name and then you wait until your name is called.

When that happens, you receive your latte:



```
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
    }
);
```

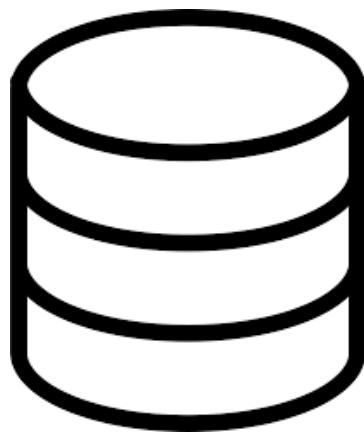
# All you need to know

- 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.

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

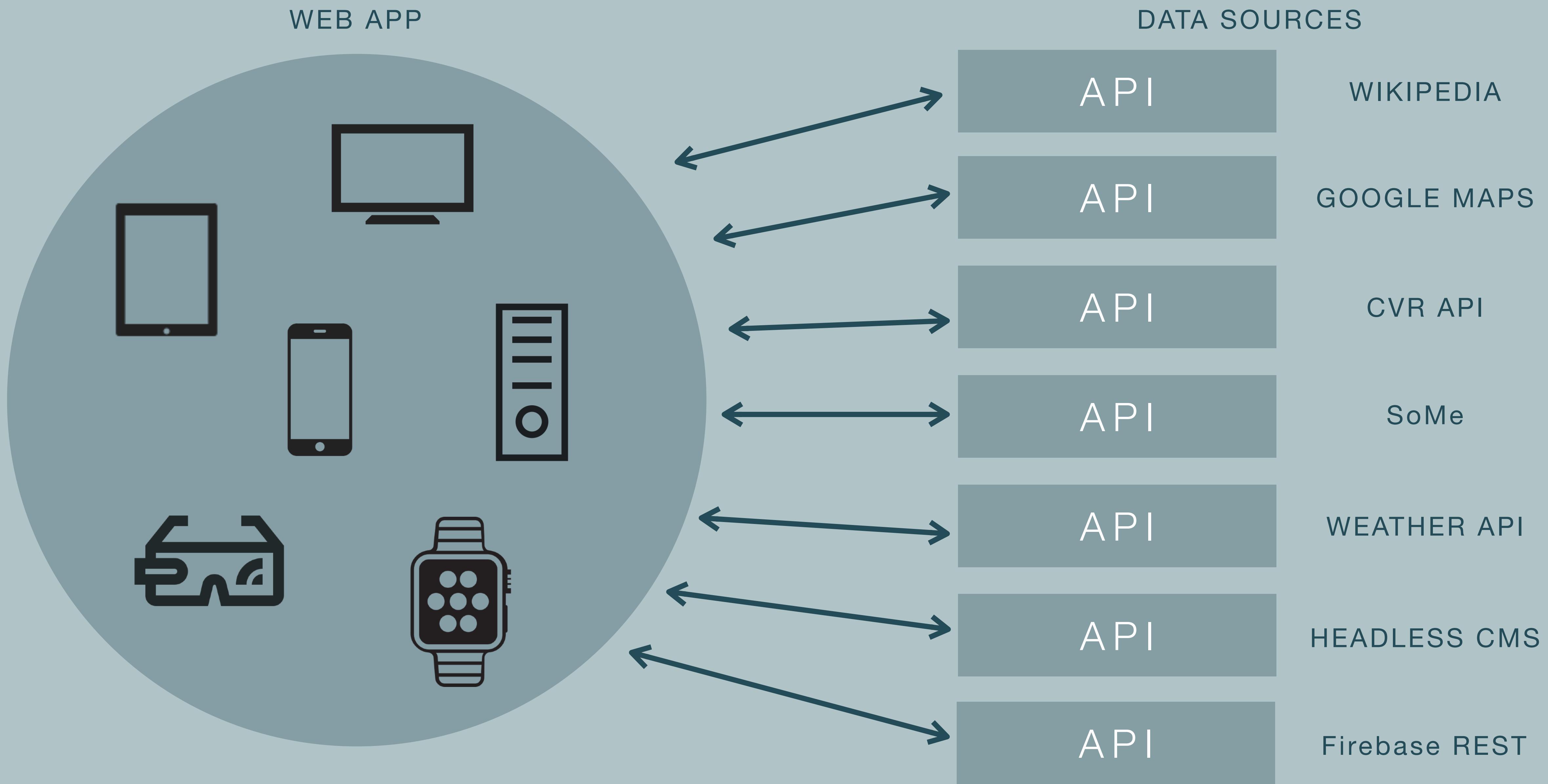
# What's a Data Source?

- Location of data
- Where data is coming from
- Can be any kind of data of any file format
- Database, a file, data sheet, spreadsheet, XML, JSON



{JSON}

# API



# API

## Application Programming Interface

A way for two systems to communicate - get and post data.

# What's an API?

<https://www.youtube.com/watch?v=s7wmiS2mSXY>

# API

## APPLICATION PROGRAMMING INTERFACE

The screenshot shows a web browser window titled "API'er - definition og funktion." with the URL "chat.openai.com". The page content is a conversation with AI, starting with a user question "Hvad er en API?" followed by AI responses explaining what APIs are, their functions, and types.

**Hvad er en API?**

API står for "Application Programming Interface", på dansk "Programmeringsgrænseflade". En API er en samling af software-komponenter, protokoller og værktøjer, der gør det muligt for forskellige programmer eller systemer at kommunikere med hinanden og udveksle data på en standardiseret måde.

API'er kan fungere som en bro mellem forskellige applikationer, platforme eller tjenester, hvilket gør det muligt for udviklere at udnytte funktionalitet eller data fra andre systemer uden at skulle skrive alt koden fra bunden. API'er gør det også muligt at integrere forskellige tjenester og applikationer for at skabe mere komplekse og avancerede systemer.

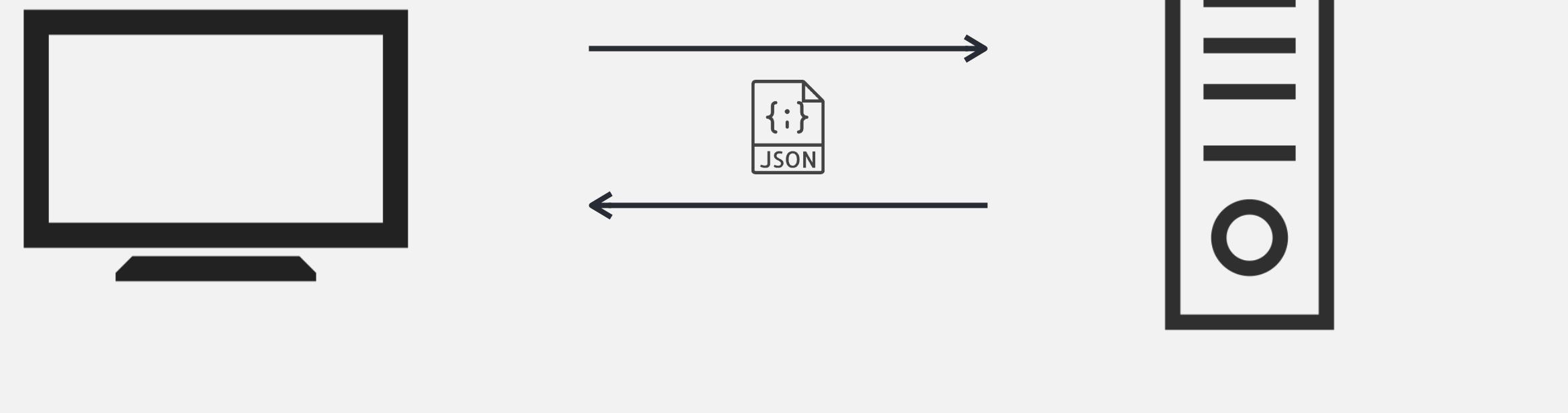
Der findes mange forskellige typer af API'er, inklusiv web-API'er, biblioteks-API'er og operativsystem-API'er. Web-API'er er en af de mest almindelige typer, og disse gør det muligt for applikationer at kommunikere via internettet ved hjælp af standardiserede protokoller såsom HTTP eller REST.

**Send a message...**

ChatGPT Mar 23 Version. Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts.

# Application Programming Interface

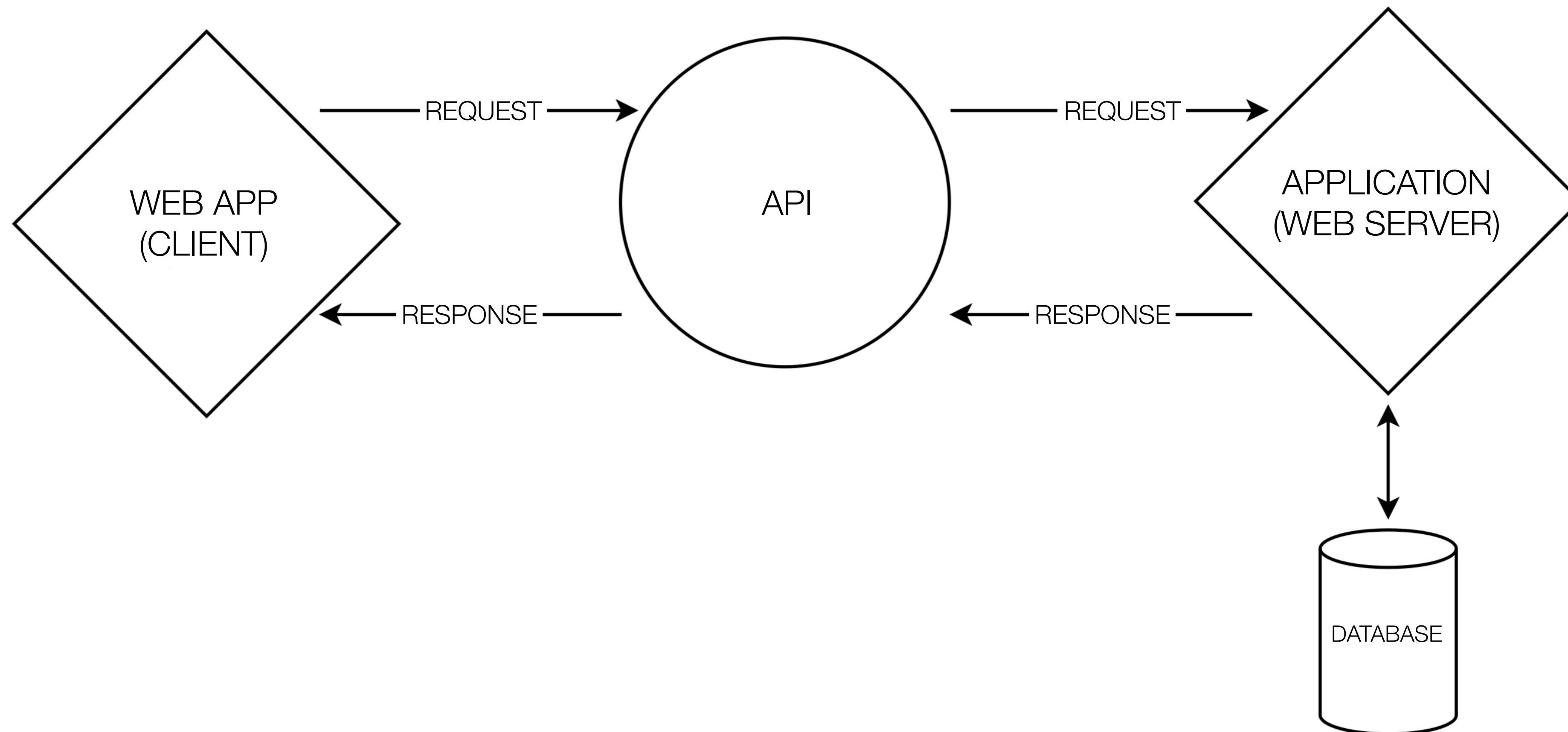
- An interface that makes it possible for two systems to communicate and exchange data in a standardised way.
- Ex communication between a web app and a web server.
- An API is the glue or bridge between different applications, platforms, or services.
- Platform independent: Can be used by different clients, devices and users: websites, web apps, mobile apps, webshops and other clients.
- The client does not need to know anything about the service or program provided by the API and vice versa.
- There are many different types of API's: web APIs, library APIs, operating system APIs, etc.
- Web APIs are one of the most common types, allowing applications to communicate over the web using standardised protocols such as HTTP or REST.



WEB APP  
(CLIENT)

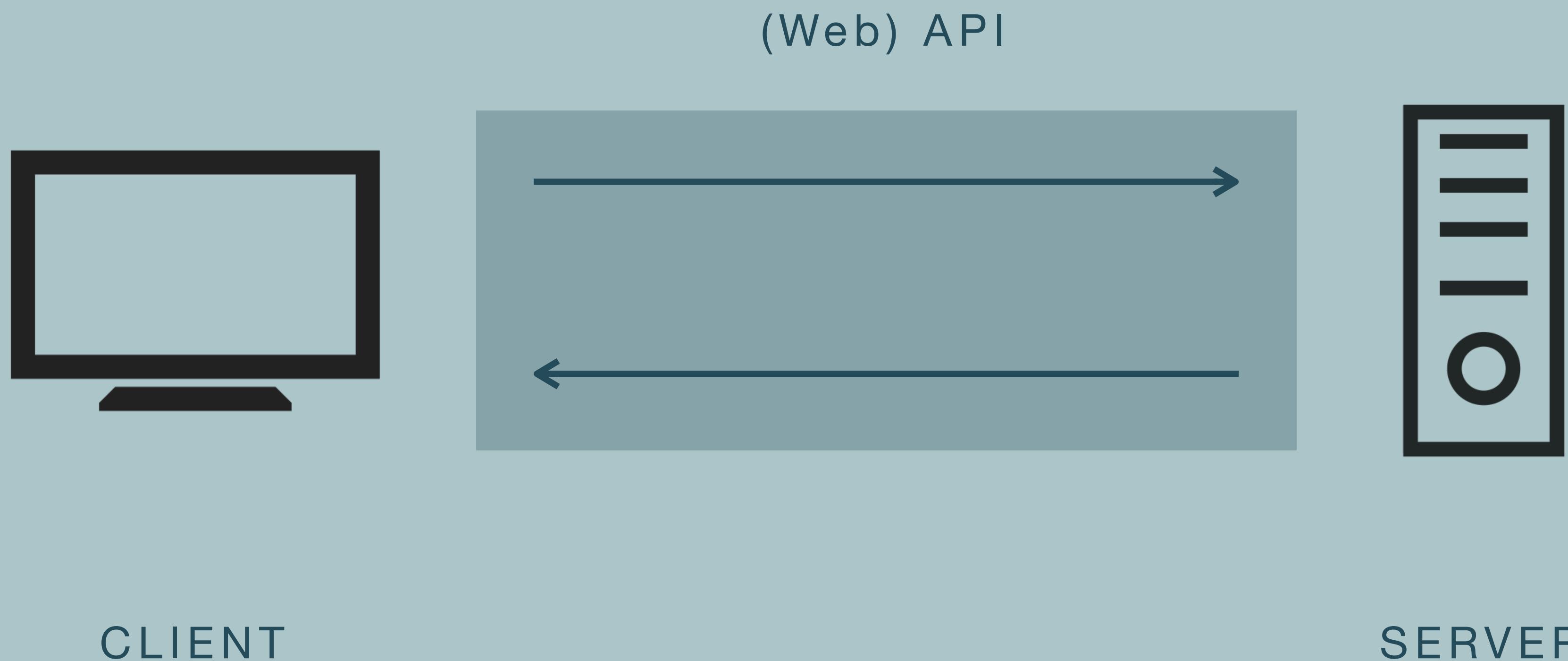
APPLICATION/ SERVICE  
(WEB SERVER)

# Application Programming Interface



# Application Programming Interface

Communication between two systems



# Application Programming Interface

Communication between two systems



Web App

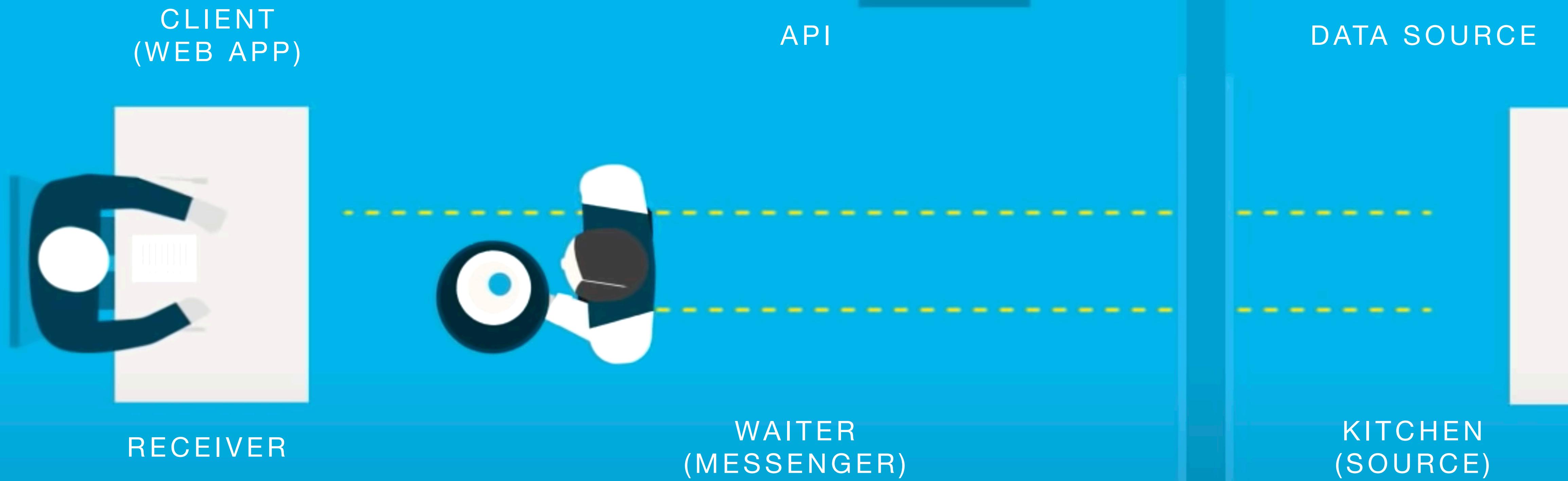
Camera API



Camera



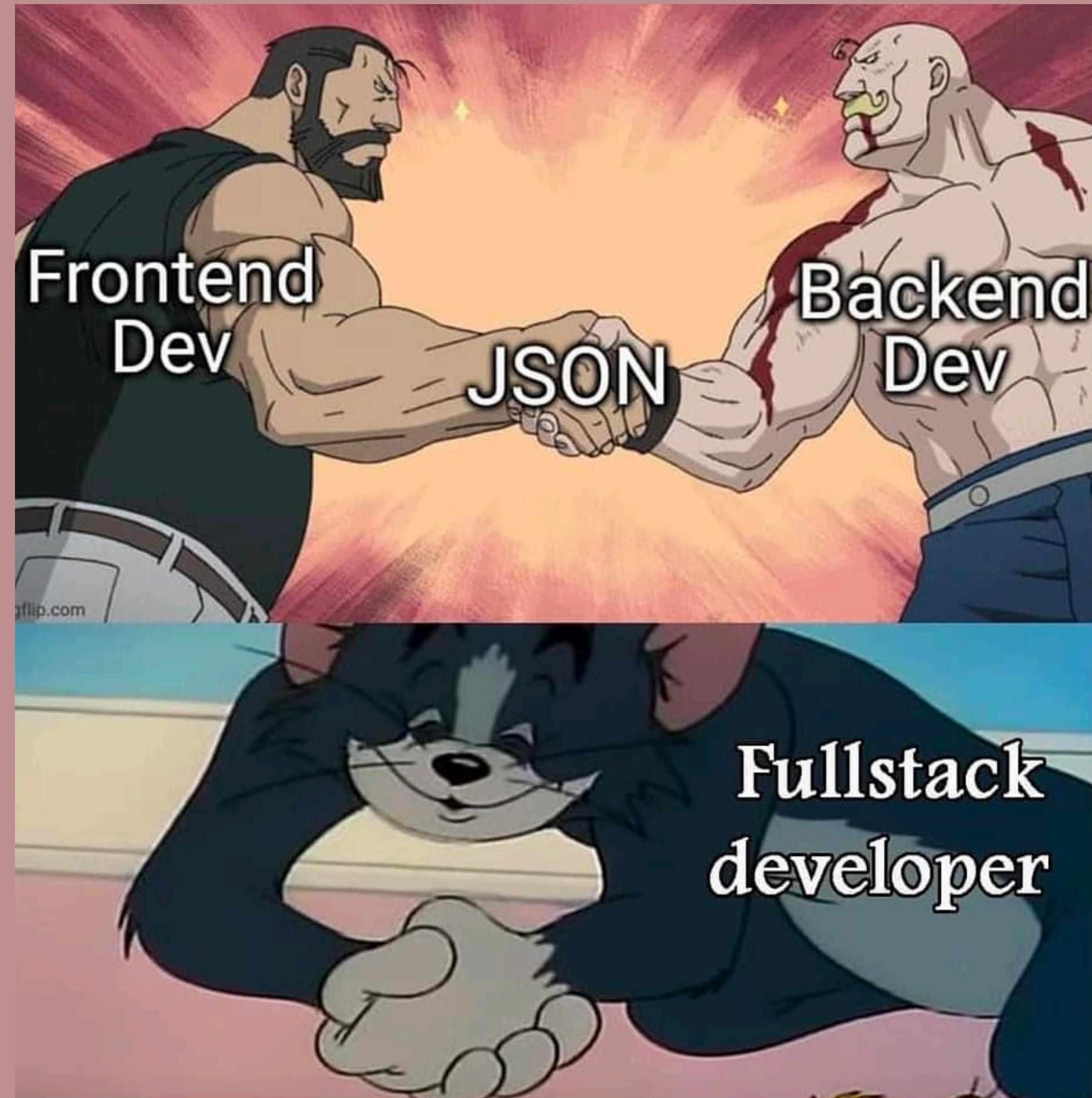
<https://www.youtube.com/watch?v=s7wmiS2mSXY>



# JSON

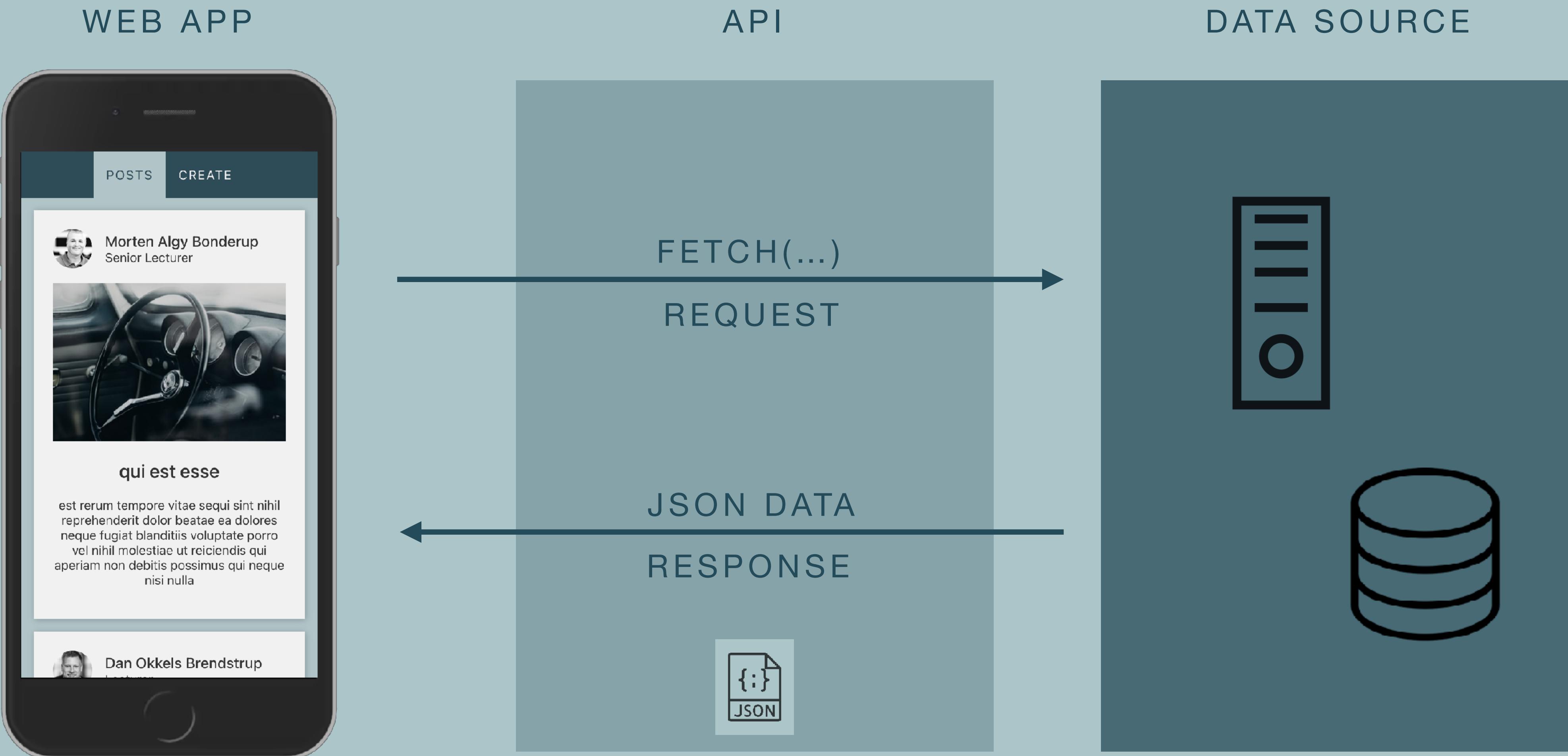
## JavaScript Object Notation

... 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

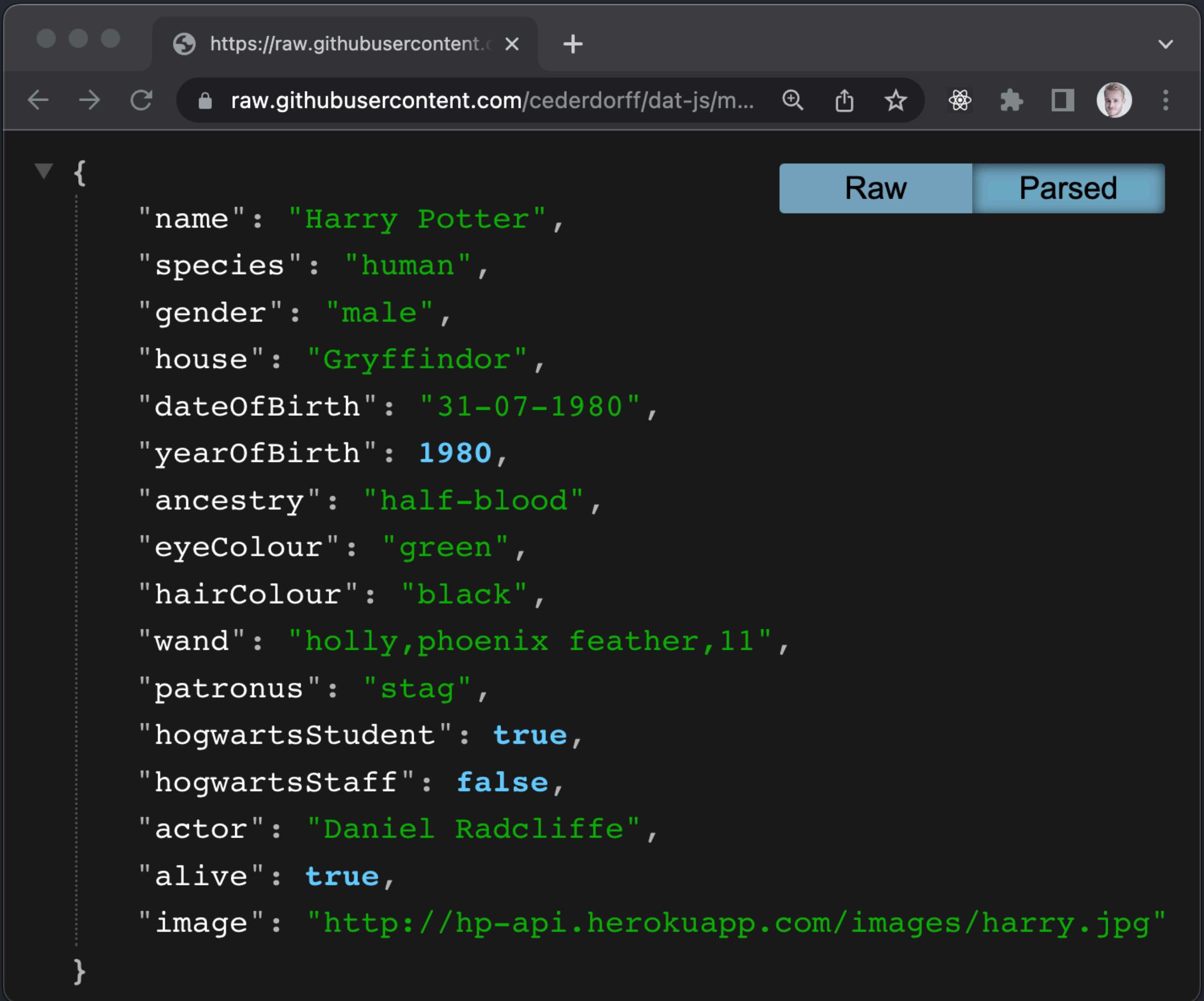
```
{  
  "name": "Alicia",  
  "age": 6  
}
```

JSON OBJECT

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

LIST OF JSON OBJECTS

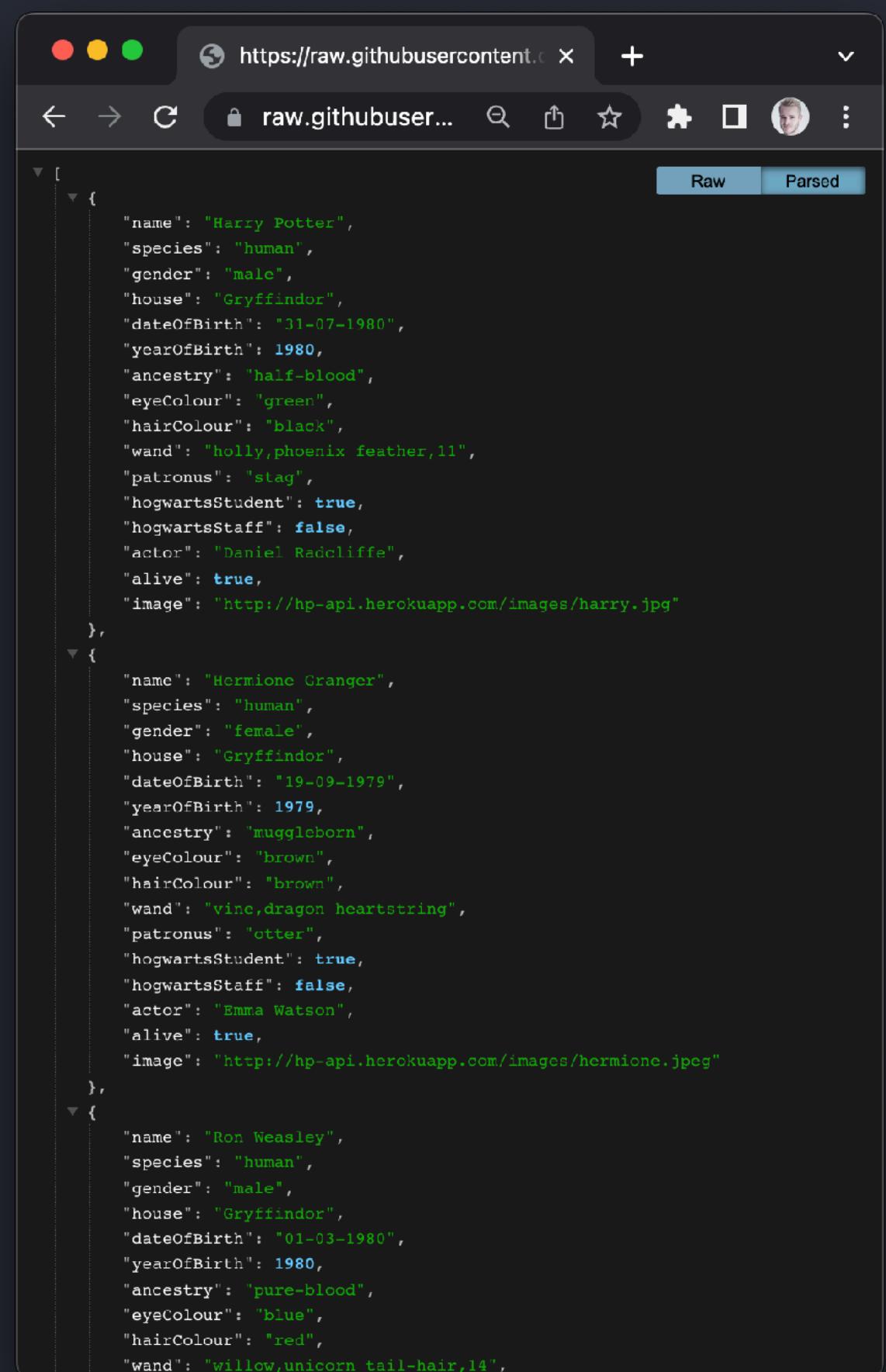
# JSON



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 object represents Harry Potter's character details. The 'Raw' tab is selected, showing the raw JSON code, while the 'Parsed' tab shows the same data as an expandable tree structure.

```
{  
  "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"  
}
```

JSON OBJECT



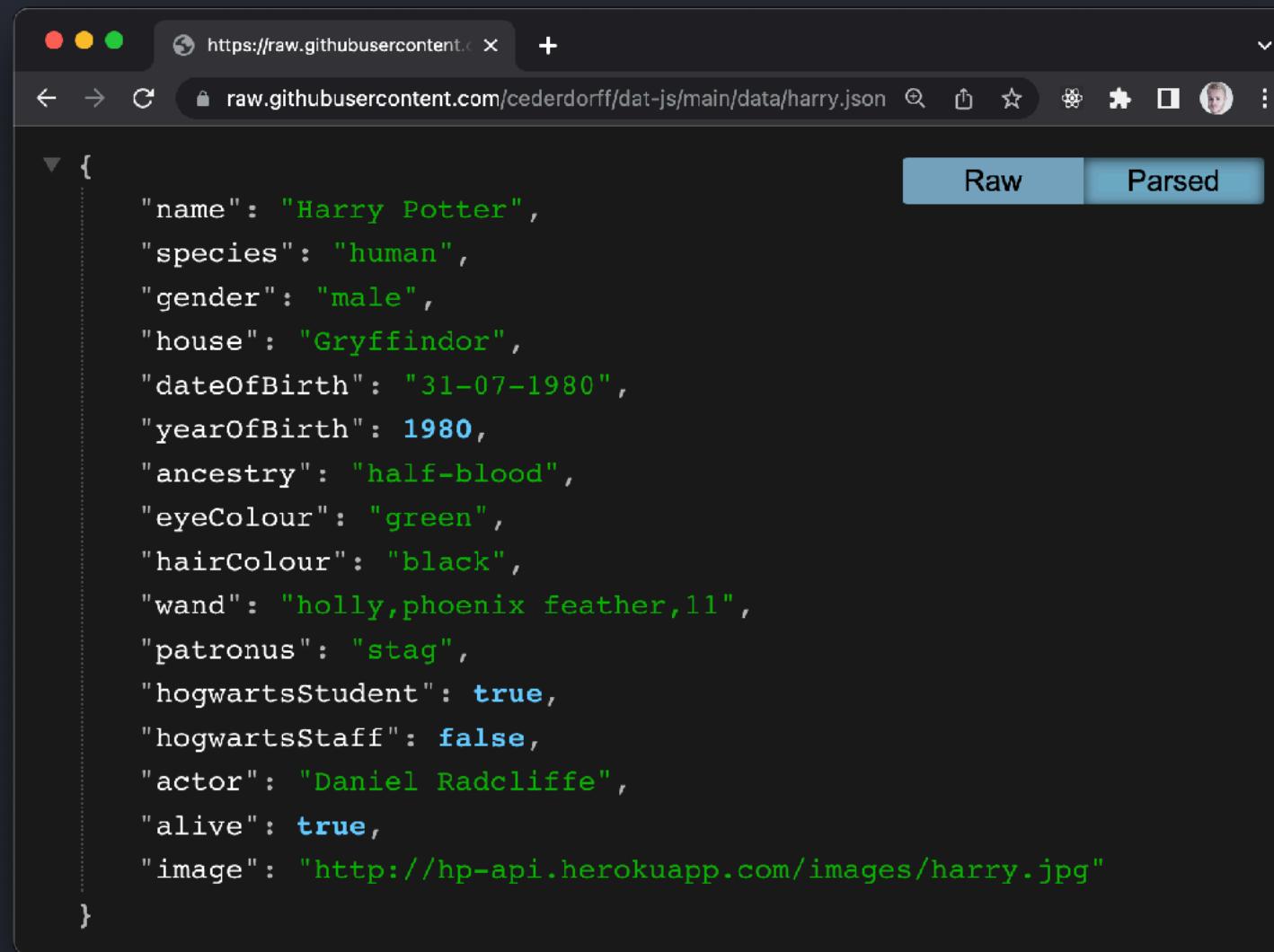
A screenshot of a web browser window displaying a list of three JSON objects. The URL is <https://raw.githubusercontent.com/cederdorff/dat-js/master/data/characters.json>. The JSON array contains three objects: Harry Potter, Hermione Granger, and Ron Weasley. The 'Raw' tab is selected, showing the raw JSON code, while the 'Parsed' tab shows the data as an expandable tree structure.

```
[  
  {  
    "name": "Harry Potter",  
    "species": "human",  
    "gender": "male",  

```

LIST OF JSON OBJECTS

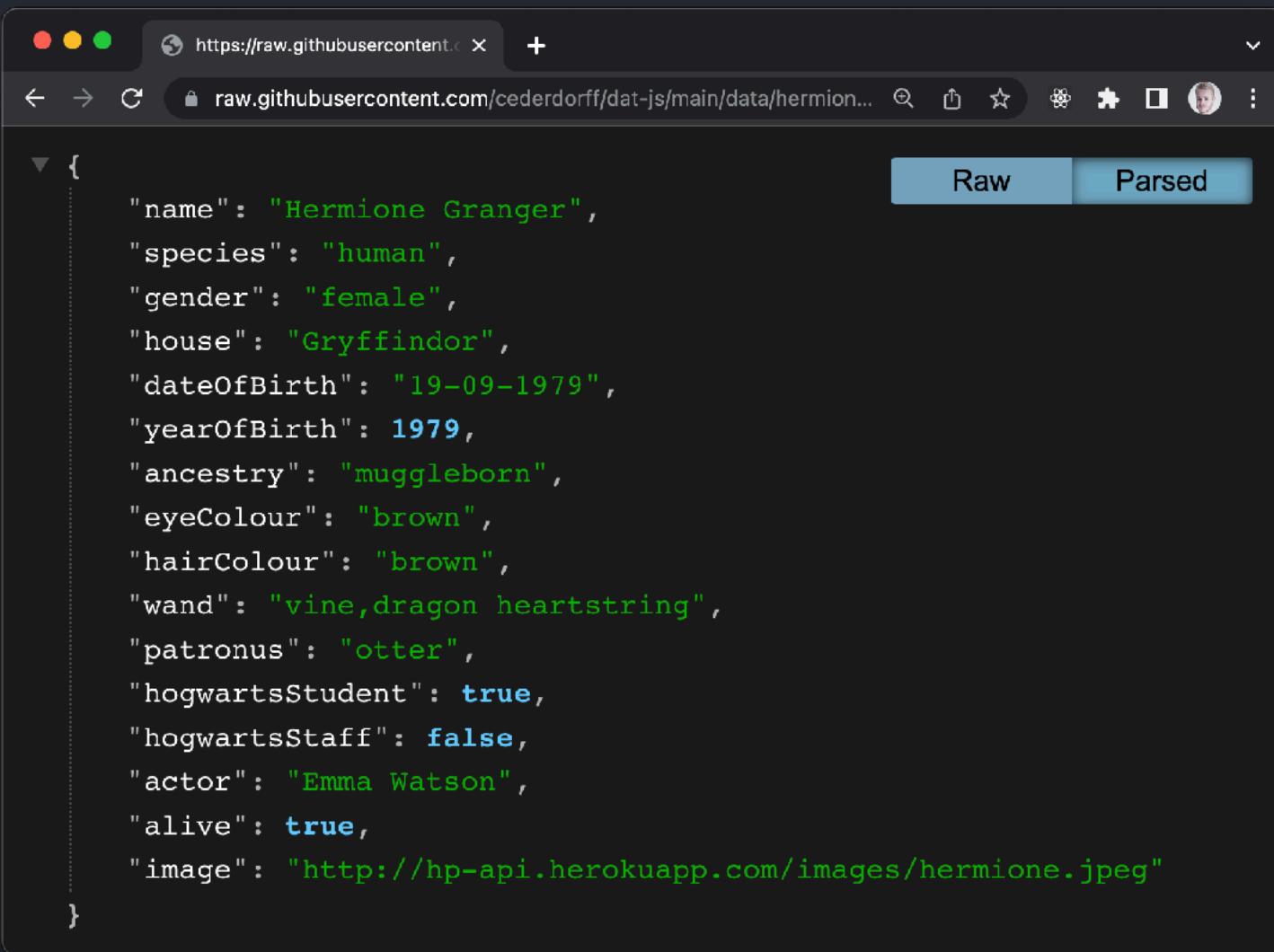
# JSON Objects



A screenshot of a web browser displaying a JSON object for Harry Potter. The JSON is shown in a dark-themed code editor interface with syntax highlighting. The object contains properties such as name, species, gender, house, date of birth, year of birth, ancestry, eye colour, hair colour, wand, patronus, Hogwarts student status, staff status, actor, alive status, and image URL. A 'Raw' and 'Parsed' button is visible at the top right.

```
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"
}
```

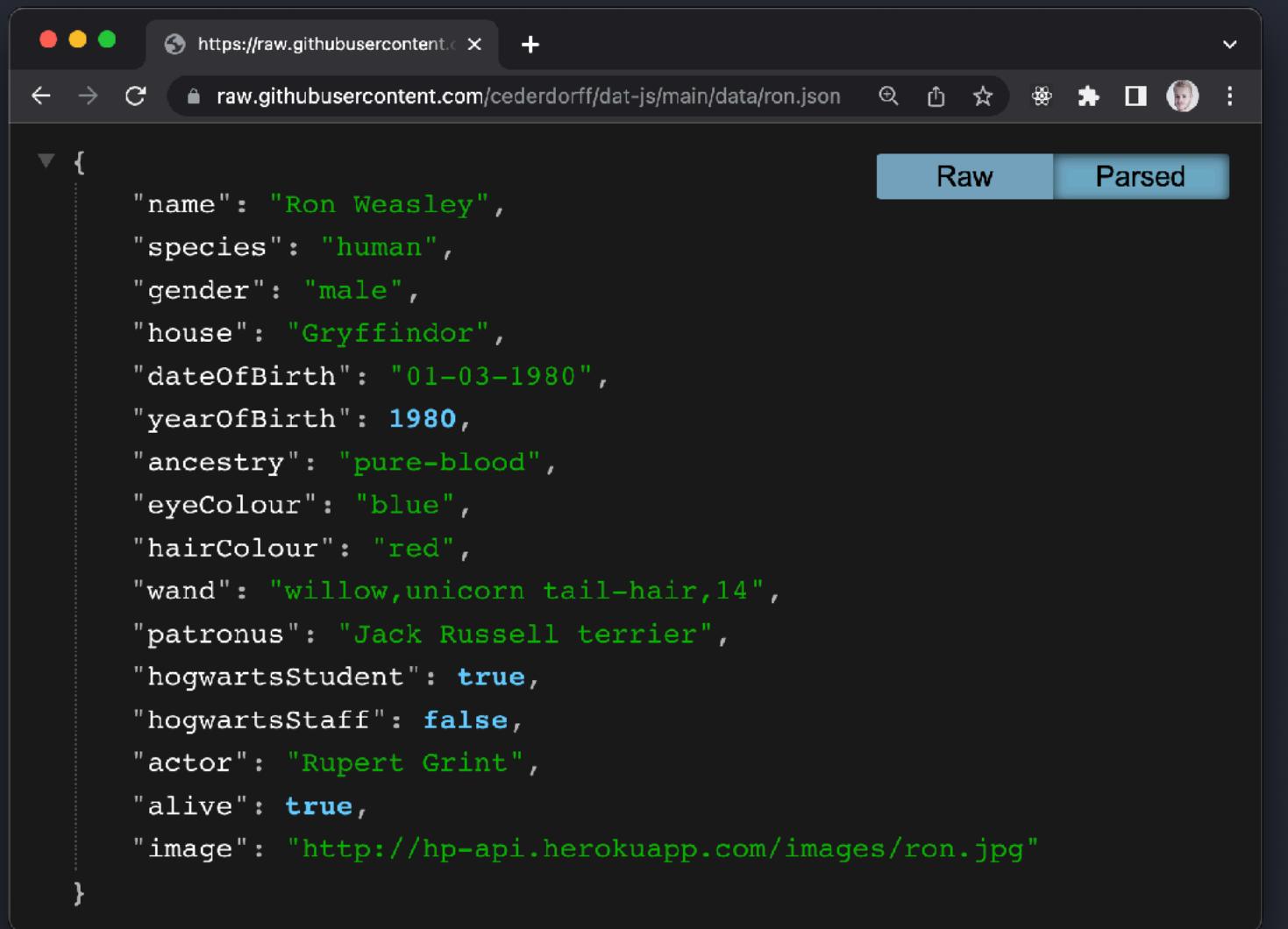
<https://raw.githubusercontent.com/cederdorff/dat-js/main/data/harry.json>



A screenshot of a web browser displaying a JSON object for Hermione Granger. The JSON is shown in a dark-themed code editor interface with syntax highlighting. The object contains properties such as name, species, gender, house, date of birth, year of birth, ancestry, eye colour, hair colour, wand, patronus, Hogwarts student status, staff status, actor, alive status, and image URL. A 'Raw' and 'Parsed' button is visible at the top right.

```
Raw Parsed
{
  "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"
}
```

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



A screenshot of a web browser displaying a JSON object for Ron Weasley. The JSON is shown in a dark-themed code editor interface with syntax highlighting. The object contains properties such as name, species, gender, house, date of birth, year of birth, ancestry, eye colour, hair colour, wand, patronus, Hogwarts student status, staff status, actor, alive status, and image URL. A 'Raw' and 'Parsed' button is visible at the top right.

```
Raw Parsed
{
  "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",
  "patronus": "Jack Russell terrier",
  "hogwartsStudent": true,
  "hogwartsStaff": false,
  "actor": "Rupert Grint",
  "alive": true,
  "image": "http://hp-api.herokuapp.com/images/ron.jpg"
}
```

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

# 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:

- 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 Harry Potter's character data. The JSON is displayed in a collapsible tree view with the "Parsed" tab selected. The data 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 the corresponding JSON properties in the JSON object, illustrating how the application maps the JSON data to its user interface.

<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
```

# JSON.parse()

Converts a string  
(JSON) to a JavaScript  
Object or Array.

```
// User as a text string
const userString = '{"name":"John", "age":30, "city":"New York"}';
// Converts the text into a JavaScript Object
const userObject = JSON.parse(userString);
console.log(userObject);
```

```
// Cars as a text string
const carsString = '[{"Ford", "BMW", "Audi", "Fiat", "VW"}]';
// Converts the text into a JavaScript Array
const carsArray = JSON.parse(carsString);
console.log(carsArray);
```

# JSON.stringify()

Converts a JavaScript Object or Array to a string (JSON).

```
// User as a JavaScript Object
const user = { name: "John", age: 30, city: "New York" };
// Converts user object to JSON string
const userJSON = JSON.stringify(user);
console.log(userJSON);
```

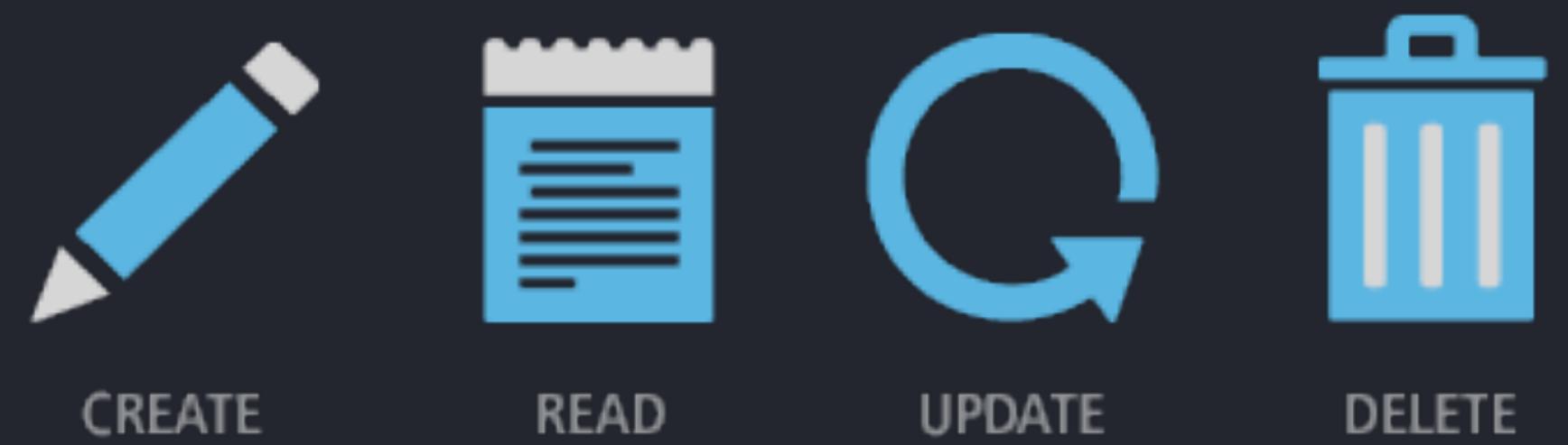
```
// Cars as a JavaScript Array
const cars = ["Ford", "BMW", "Audi", "Fiat", "Vw"];
// Converts cars array to JSON string
const carsJSON = JSON.stringify(cars);
console.log(carsJSON);
```

# Fetch and create post

newPost object is parsed  
to JSON and posted to the  
ressource wrapped in body

```
// new post object
const newPost = {
  title: "My new post",
  body: "Body description of a new post",
  image: "image url or image data string"
};

const url = "https://race-rest-default.firebaseio.com/posts.json";
const response = await fetch(url, {
  method: "POST", // fetch request using POST
  body: JSON.stringify(newPost) //←newPost object to JSON
});
```



C R U D

# What's CRUD?

- CREATE objects like a post, user, movie, product, etc.
- READ objects like an array (or object) of objects (posts, users, movies, products, etc)
- UPDATE an object, often given by a unique id.
- DELETE an object, often given by a unique id.

# What's REST?

GET

POST

PUT

DELETE

- REpresentational State Transfer
- A standard for systems (client & server) to communicate over HTTP in order to retrieve or modify (data) resources.
- Stateless, meaning the two systems doesn't need to know anything about the state.
- The client makes the requests using the 4 basic HTTP verbs to define the operation.

# 100 *SECONDS OF* node



<https://youtu.be/-MTSQjw5DrM>

# Dataforsyningen REST API

... an example of a REST API



<https://dataforsyningen.dk/>

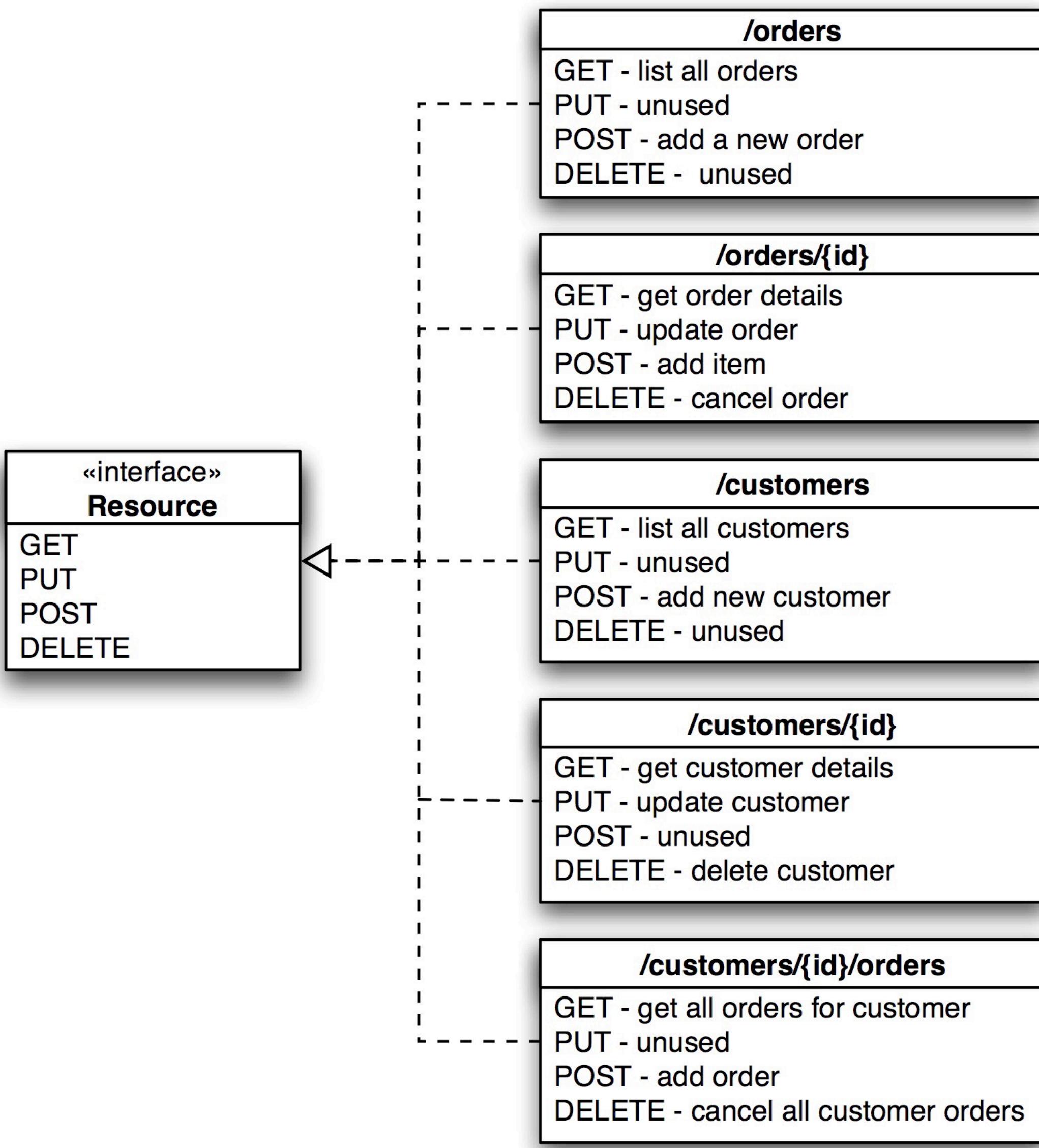
A screenshot of a web browser showing the REST API documentation for Dataforsyningen. The title "Dataforsyningen REST API" is prominently displayed. Below it, links to the API endpoint (https://api.dataforsyningen.dk/), documentation (https://dataforsyningen.dk/), and a data overview (https://dataforsyningen.dk/data). A section titled "Eksempel på kald" (Example of call) shows a link to "Danmarks adresser og vejnavne" (https://dataforsyningen.dk/data/4729) and its corresponding REST API endpoint (https://dawadocs.dataforsyningen.dk/dok/api). Another section titled "Regioner" (Regions) shows how to get all regions (GET https://api.dataforsyningen.dk/regioner) and a specific region (GET https://api.dataforsyningen.dk/regioner/1082).

[Dataforsyningen REST API](#)

# REpresentational State Transfer

- Clean, semantic URL:
  - <http://example.com/products/2/25> instead of:
  - [http://example.com/products?  
category=2&id=25](http://example.com/products?category=2&id=25)
- Basic HTTP requests to perform create, read, update and delete with the HTTP methods GET, POST, PUT and DELETE

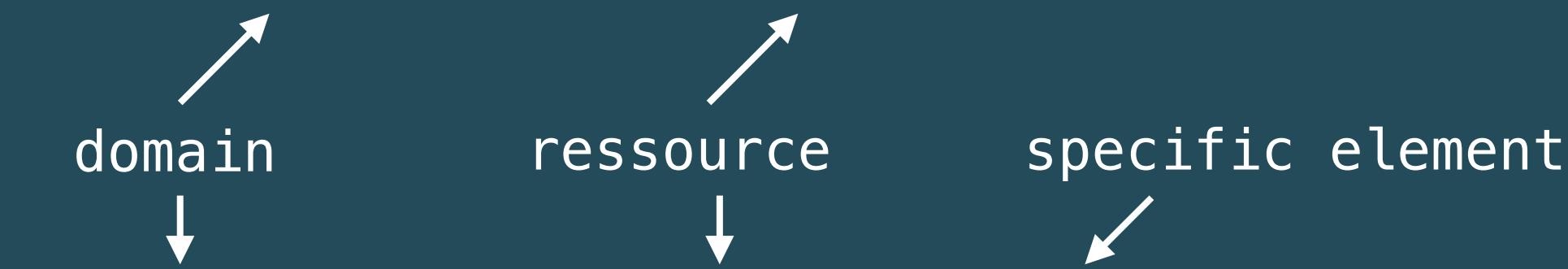
# Principles of REST



- Ressources and endpoints:
  - <http://example.com/products>
  - <http://example.com/users>
  - <http://domain.com/orders>
- Everything must have an id:
  - <http://example.com/products/2/25>
  - <http://example.com/users/7503>
  - <http://domain.com/orders/2014/06/4022>
- Connect data
  - <http://example.com/users/7503/orders/3/item/1>
- Standard HTTP request methods (GET, POST, PUT, DELETE)
- Exchanges data, often JSON (oldschool: XML)
- Stateless Communication

# RESTful API

- Base URL (endpoint): http://some-url.com/products



- http://some-url.com/products/product1

- Data type → JSON

Ressource	GET	POST	PUT	DELETE
Collection: <u>http://some-url.com/products</u>	Returns a list with all products	Creates new product, added to the collections	Replaces a collection with a another	Deletes all products
Element: <u>http://some-url.com/products/product1</u>	Returns a specific product	÷	Replaces product with new (updated) data	Deletes product

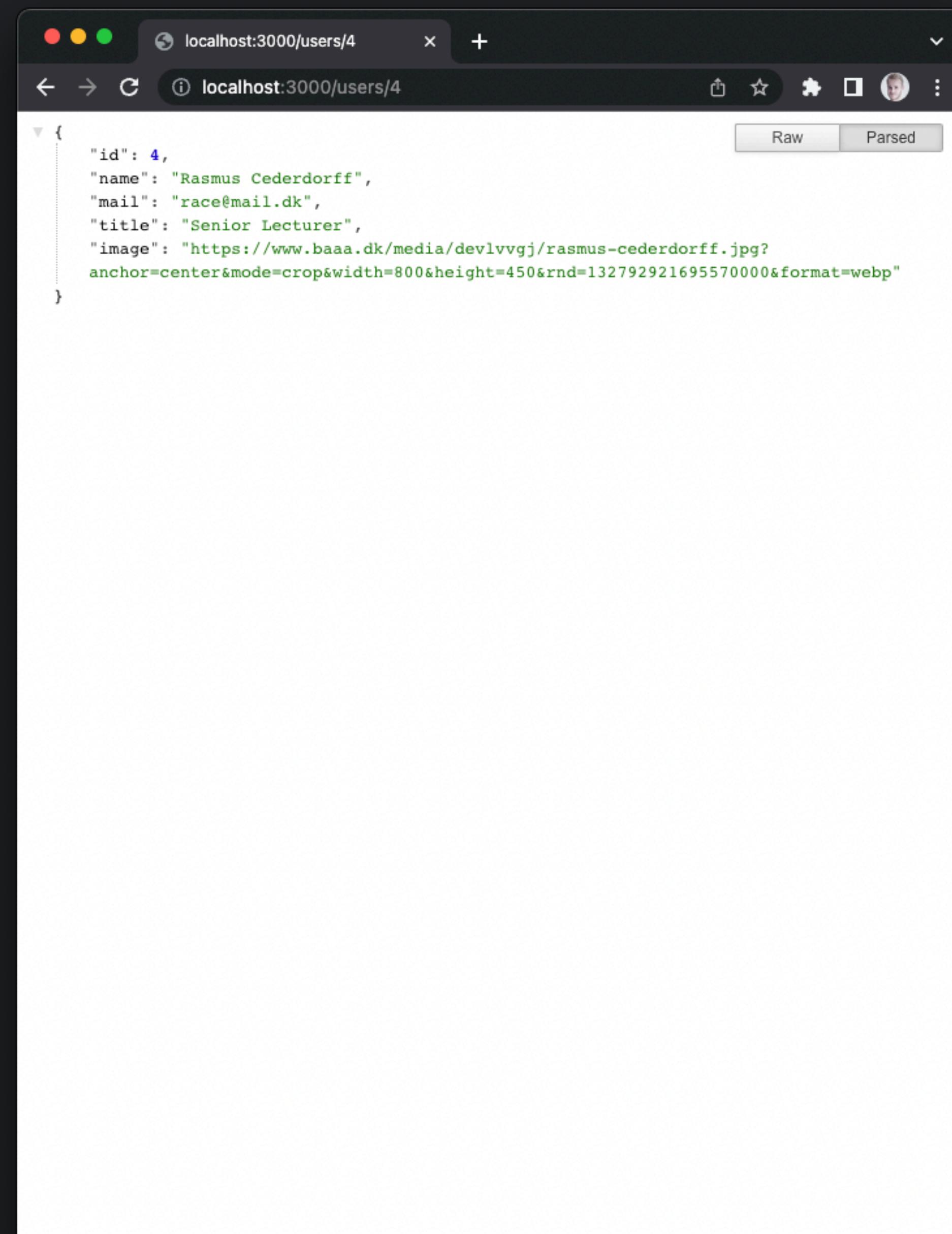
# Collections (JSON Array)



A screenshot of a web browser window titled "localhost:3000/users". The page displays a JSON array of user objects. Each object contains fields: id, name, mail, title, and image. The image field includes a URL and a query string for image processing. The browser interface shows "Raw" and "Parsed" buttons.

```
[{"id": 1, "name": "Birgitte Kirk Iversen", "mail": "bki@mail.dk", "title": "Senior Lecturer", "image": "https://www.baaa.dk/media/u4gorzsd/birgitte-kirk-iversen2.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921541630000&format=webp"}, {"id": 4, "name": "Rasmus Cederdorff", "mail": "race@mail.dk", "title": "Senior Lecturer", "image": "https://www.baaa.dk/media/devlvgj/rasmus-cederdorff.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921695570000&format=webp"}, {"id": 5, "name": "Dan Okkels Brendstrup", "mail": "dob@mail.dk", "title": "Lecturer", "image": "https://www.eaaa.dk/media/bdojel41/dan-okkels-brendstrup.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921559630000&format=webp"}, {"id": 6, "name": "Kasper Fischer Topp", "mail": "kato@mail.dk", "title": "Lecturer", "image": "https://www.eaaa.dk/media/lxzcybme/kasper-topp.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921618200000&format=webp"}, {"id": 7, "name": "Line Skjødt", "mail": "lskj@mail.dk", "title": "Senior Lecturer & Internship Coordinator", "image": "https://www.eaaa.dk/media/14qpfeq4/line-skj%C3%B8dt.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921638700000&format=webp"}, {"id": 8, "name": "Martin Aagaard Nøhr", "mail": "mnor@mail.dk", "title": "Lecturer", "image": "https://www.eaaa.dk/media/oayjq02h/martin-n%C3%B8hr.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921658800000&format=webp"}]
```

# Element (JSON Object)



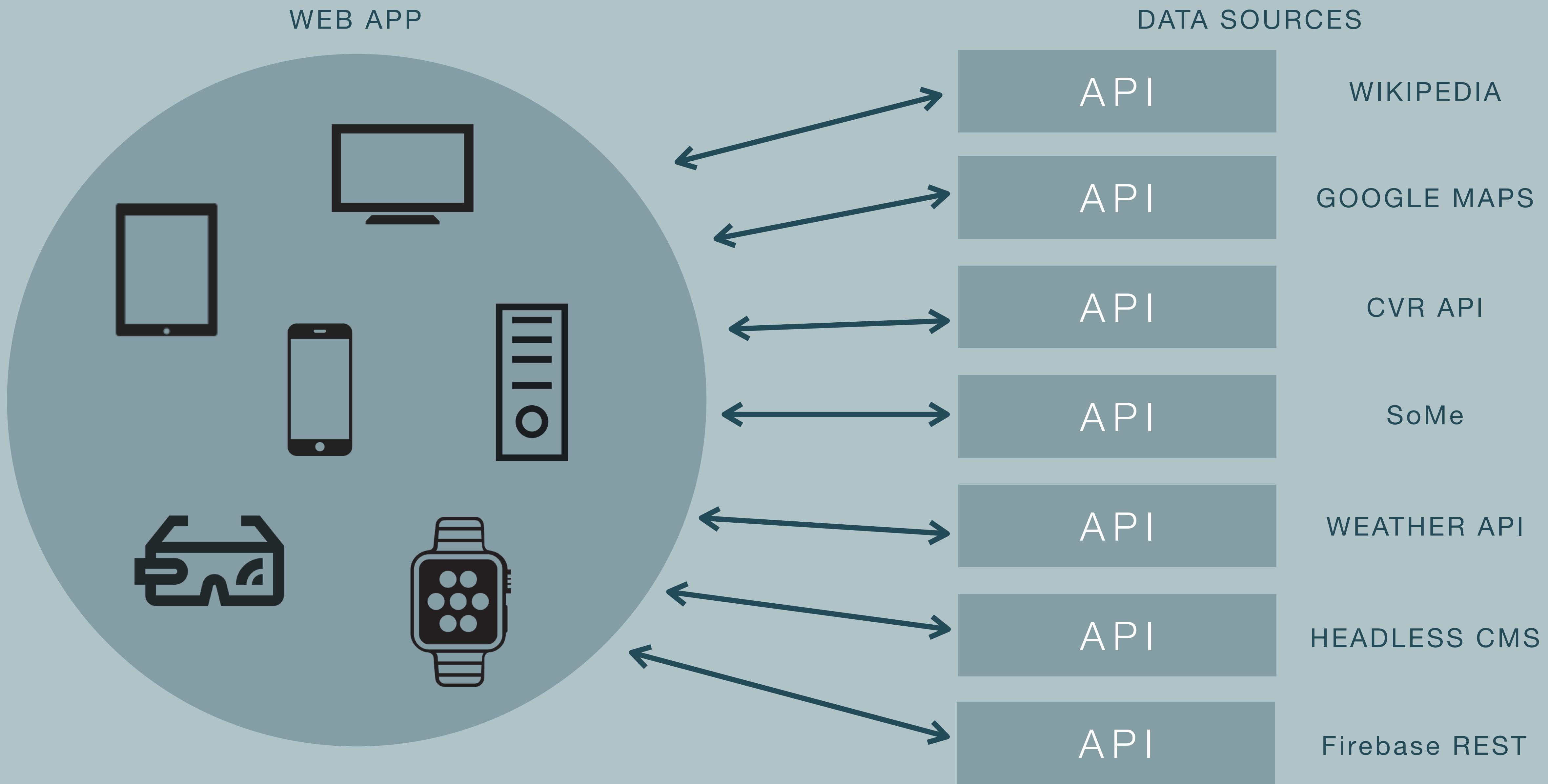
A screenshot of a web browser window titled "localhost:3000/users/4". The page displays a single JSON object representing a user with id 4. The object includes fields: id, name, mail, title, and image. The image field includes a URL and a query string for image processing. The browser interface shows "Raw" and "Parsed" buttons.

```
{ "id": 4, "name": "Rasmus Cederdorff", "mail": "race@mail.dk", "title": "Senior Lecturer", "image": "https://www.baaa.dk/media/devlvgj/rasmus-cederdorff.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921695570000&format=webp"}
```

# Advantages of REST

- Independent of platform and programming language
- Based on existing standards (on top of HTTP)
- Semantic URL → Nice and clean URLs → SEO
- Restful API
- Scalable
- Performance
- Exchange formats like JSON, XML, or both

# API



# HTTP REQUEST METHODS (verbs)

GET - POST - PUT - DELETE

HTTP (Hypertext Transfer Protocol) is the standard way to communicate between clients and servers (request-response protocol).

"HTTP defines a set of **request methods** to indicate the desired action to be performed for a given resource."

[https://www.w3schools.com/tags/ref\\_httpmethods.asp](https://www.w3schools.com/tags/ref_httpmethods.asp)

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

# CRUD vs REST & HTTP Verbs

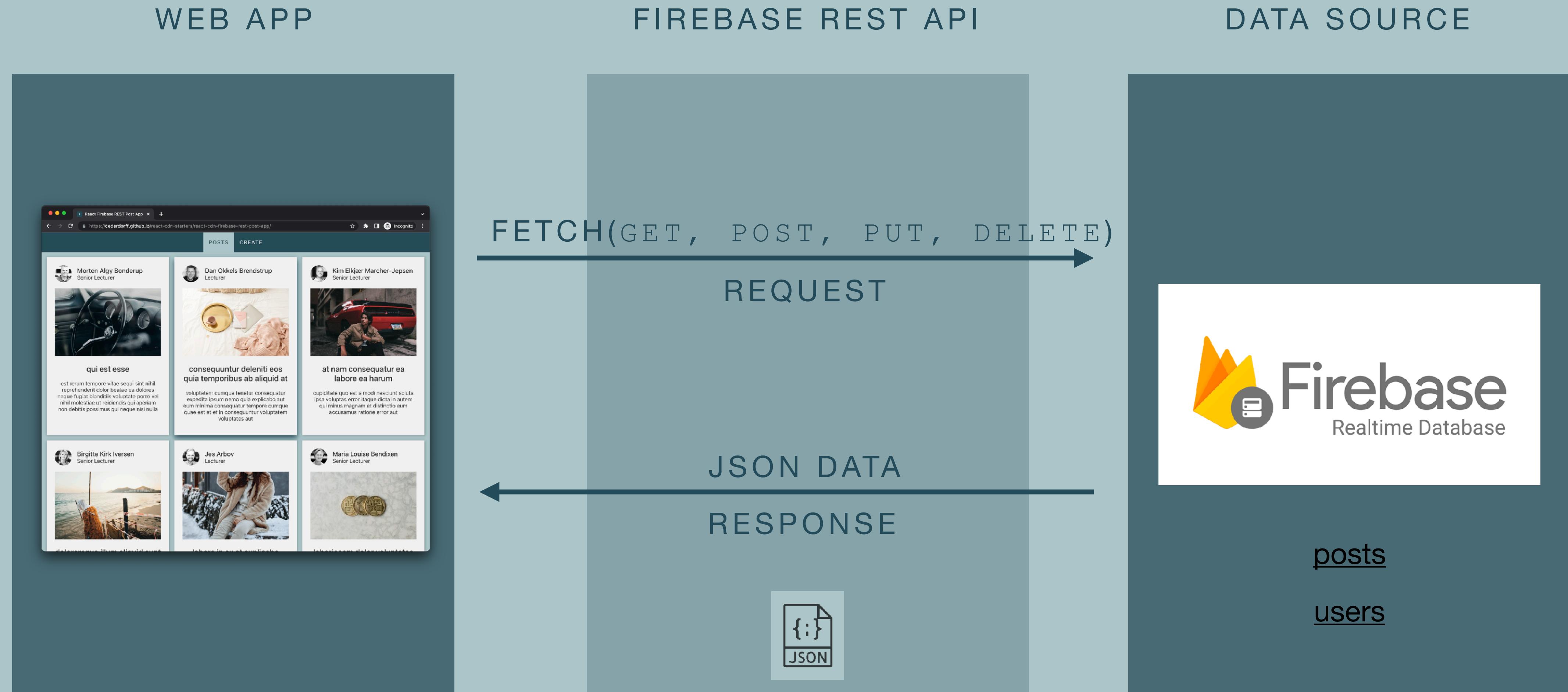
**CREATE** -> POST: create a new resource (object)

**READ** -> GET: retrieve a specific resource or a collection

**UPDATE** -> PUT: update a specific resource (by id)

**DELETE** -> DELETE: remove a specific resource by id

# Fetch, HTTP Request & Response



# Fetch and read posts

GET is the default request method for fetch.

```
async function getPosts() {  
  const url = "https://race-rest-default-rtbd.firebaseio.com/posts.json";  
  const response = await fetch(url);  
  const data = await response.json();  
  const postsArray = Object.keys(data).map(key => ({ id: key, ...data[key] })); // from object to array  
  return postsArray  
}
```

# REQUEST headers

“[...] contain more information about the resource to be fetched, or about the client requesting the resource.”

"A request header is an HTTP header that can be used in an HTTP request to provide information about the request context, so that the server can tailor the response. For example, the Accept-\* headers indicate the allowed and preferred formats of the response. Other headers can be used to supply authentication credentials (e.g. Authorization), to control caching, or to get information about the user agent or referrer, etc."

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>

[https://developer.mozilla.org/en-US/docs/Glossary/Request\\_header](https://developer.mozilla.org/en-US/docs/Glossary/Request_header)

# Request body

With the HTTP Fetch Request, we can send (POST) data. The data is wrapped inside the **request body**.

The **request body** is one of the following:

- a string (often JSON-encoded string with data, an object, an array, etc.)
- form data (form/multipart)
- blob/ buffer source - binary data
- URL search params (x-www-form-urlencoded)

```
async function createUser(name, title, mail, image) {  
    // create a new user  
    const newUser = { name, title, mail, image };  
    // convert to JSON  
    const userAsJSON = JSON.stringify(newUser);  
    // make fetch post request with userAsJSON in request body  
    const response = await fetch(` ${endpoint}/users.json`, {  
        method: "POST",  
        body: userAsJSON  
    });  
  
    // check if ok (status 200)  
    if (response.ok) {  
        // if success update the DOM  
    }  
}
```

# Request body

POST

```
async function createUser(name, mail) {
  const user = {
    name: name,
    mail: mail
  };

  const response = await fetch(` ${endpoint}/users`, {
    method: "POST",
    body: JSON.stringify(user)
  });

  if (response.ok) {
    // if success, update the users grid
    updateUsersGrid();
  }
}
```

PUT

```
async function updateUser(id, name, mail) {
  const userToUpdate = { name: name, mail: mail };

  const response = await fetch(` ${endpoint}/users/${id}.json`, {
    method: "PUT",
    body: JSON.stringify(userToUpdate)
  });

  if (response.ok) {
    const data = await response.json();
    console.log("User updated:", data);
    updateUserGrid();
  } else {
    console.log("Sorry, something went wrong");
  }
}
```

# The slides is your documentation

## Read and use them carefully

when you ask Rasmus  
for help and he says  
"Read documentation"



# Fetch and create post

Using POST to create a new post object in the resource.

```
// new post object
const newPost = {
  title: "My new post",
  body: "Body description of a new post",
  image: "image url or image data string"
};

const url = "https://race-rest-default-firebaseio.com/posts.json";
const response = await fetch(url, {
  method: "POST", // fetch request using POST
  body: JSON.stringify(newPost) // newPost object to JSON
});
```

# Fetch and update post

Using PUT to an existing post object by given id.

```
const postId = "5tl4jHHSRaKEB0UW9nQd"; // id of the object to update
const postToUpdate = { title: "...", body: "...", image: "..." };

const url = `https://race-rest-default-firebase.firebaseio.com/posts/${postId}.json`;
const response = await fetch(url, {
  method: "PUT", // using HTTP method PUT
  body: JSON.stringify(postToUpdate)
});
```

# Fetch and delete post

Using DELETE to an object by given id.

```
const postId = "5tl4jHHSRaKEB0UW9nQd"; // id of the object to update
const url = `https://race-rest-default-rtdb.firebaseio.com/posts/${postId}.json`;

const response = await fetch(url, {
  method: "DELETE"
});
```



# What is Firebase?

Platform, a suite of tools & Backend-as-a-Service  
for Web & App Development

# 100 *SECONDS OF*

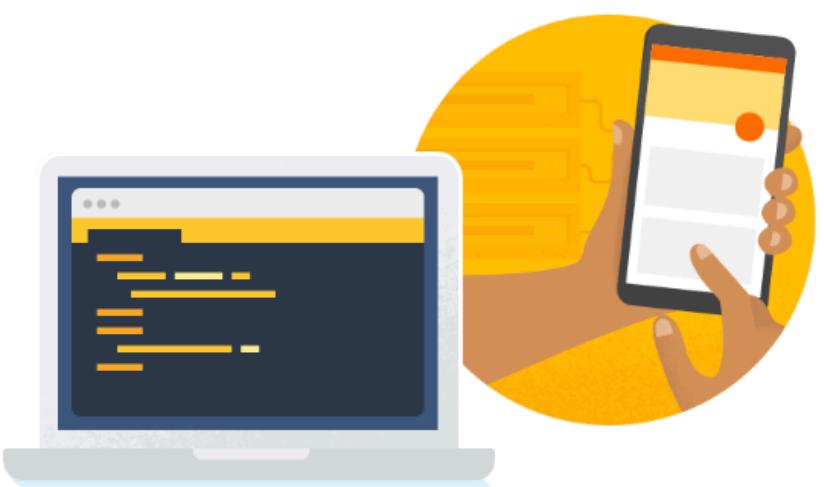


<https://www.youtube.com/watch?v=vAoB4VbhRzM>



# Introducing Firebase

<https://www.youtube.com/watch?v=iosNuldQoy8>



## Build better apps



### Cloud Firestore

Store and sync app data at global scale



### ML Kit BETA

Machine learning for mobile developers



### Cloud Functions

Run mobile backend code without managing servers



### Authentication

Authenticate users simply and securely



### Hosting

Deliver web app assets with speed and security



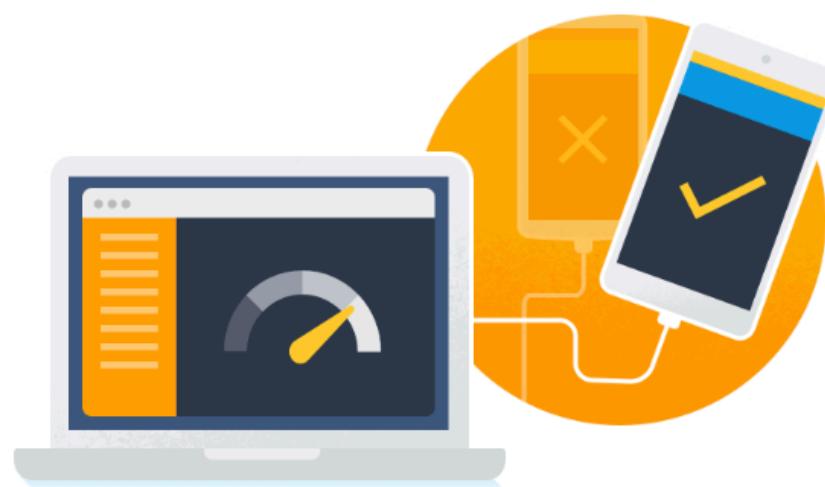
### Cloud Storage

Store and serve files at Google scale



### Realtime Database

Store and sync app data in milliseconds



## Improve app quality



### Crashlytics

Prioritize and fix issues with powerful, realtime crash reporting



### Performance Monitoring

Gain insight into your app's performance



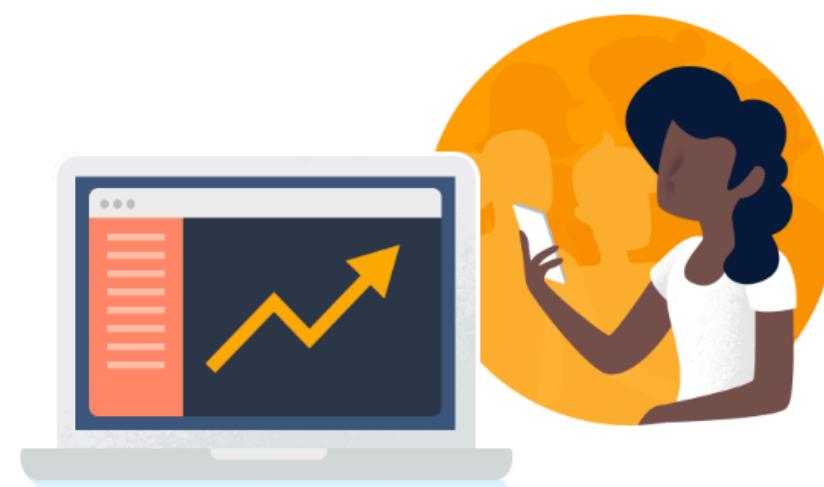
### Test Lab

Test your app on devices hosted by Google



### App Distribution BETA

Distribute pre-release versions of your app to your trusted testers



## Grow your business



### In-App Messaging BETA

Engage active app users with contextual messages



### Google Analytics

Get free and unlimited app analytics



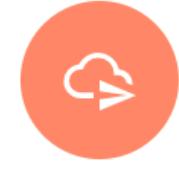
### Predictions

Smart user segmentation based on predicted behavior



### A/B Testing BETA

Optimize your app experience through experimentation



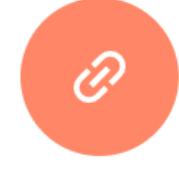
### Cloud Messaging

Send targeted messages and notifications



### Remote Config

Modify your app without deploying a new version



### Dynamic Links

Drive growth by using deep links with attribution



# Realtime Database & REST API

Store and sync data in real time  
REST API or SDK

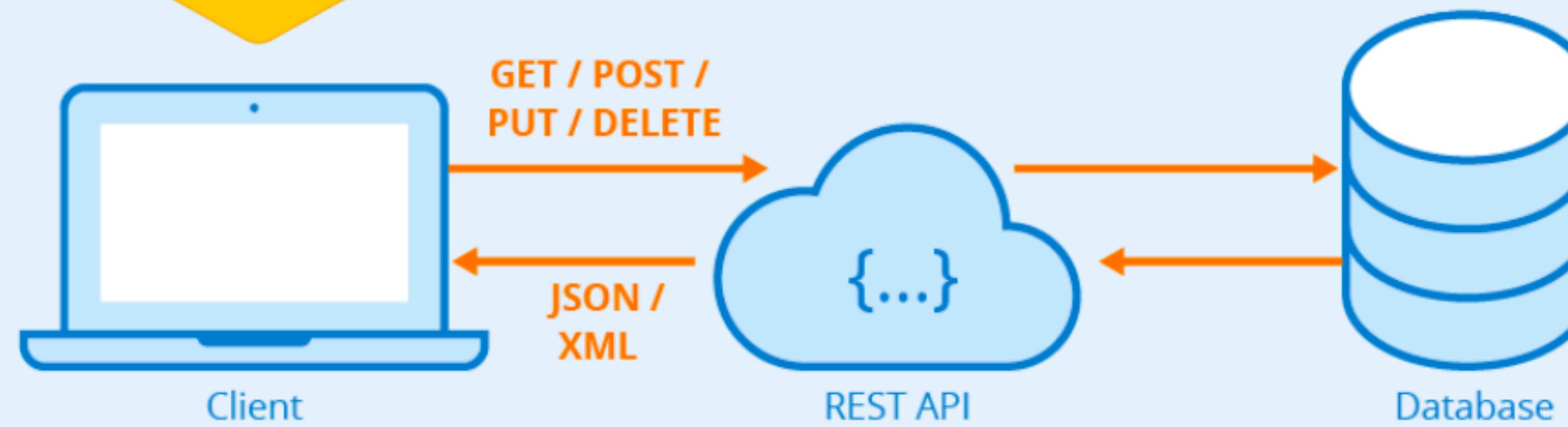
<https://firebase.google.com/products/realtime-database>



# Database



# Firebase



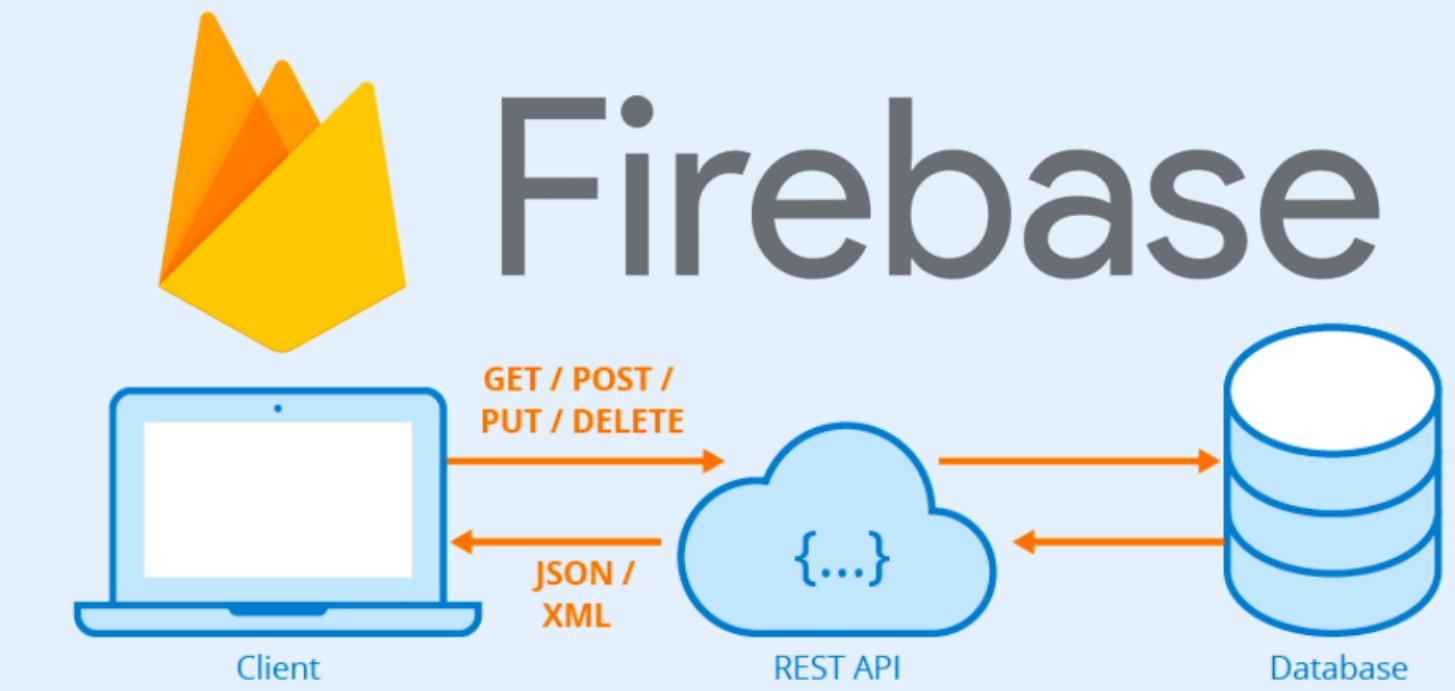
“

The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client.

We can use any Firebase Realtime Database URL as a REST endpoint. All we need to do is append .json to the end of the URL and send a request from our favorite HTTPS client.

”

<https://firebase.google.com/docs/database/rest/start>



# How data is structured

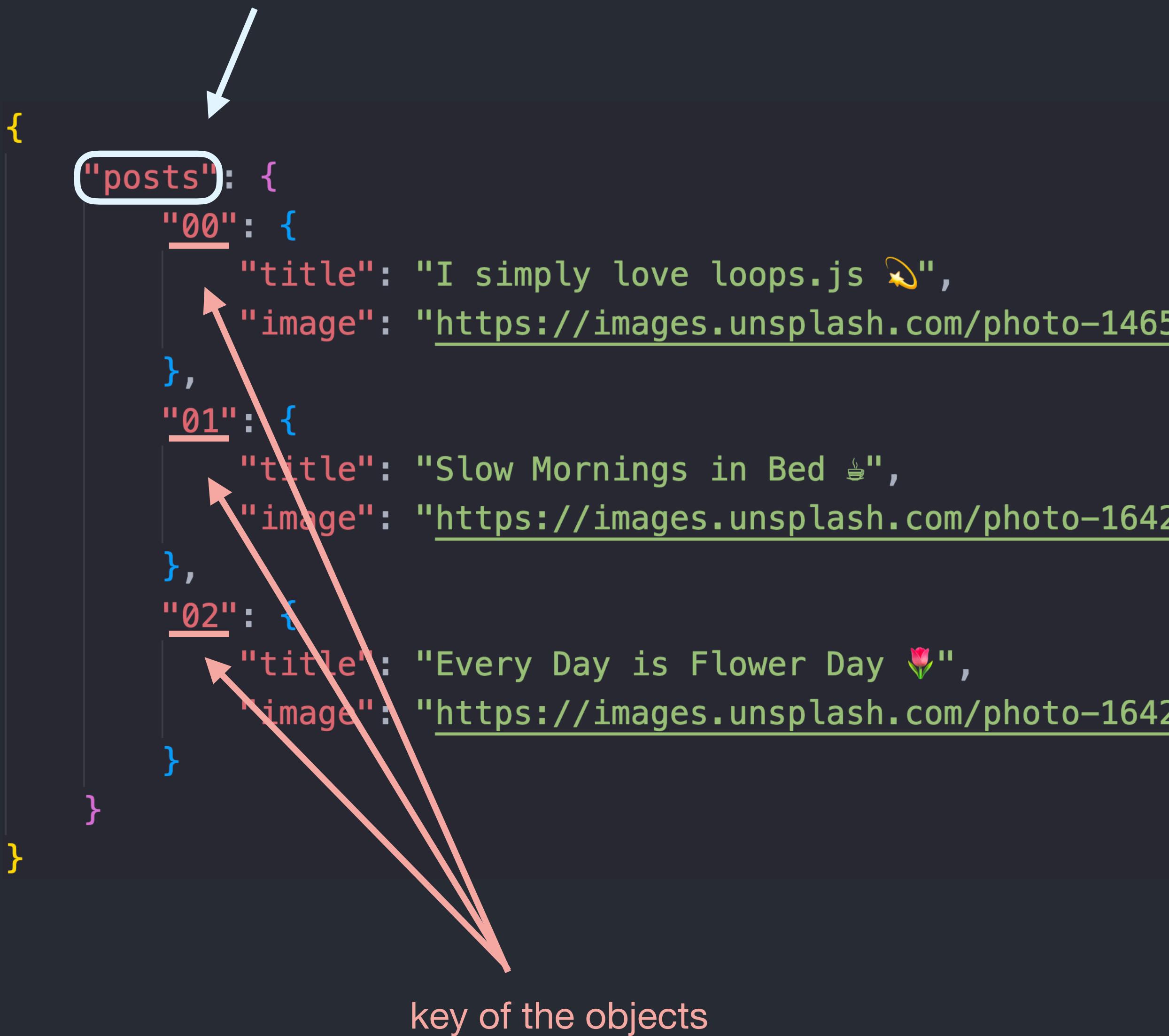
```
{  
  "posts": {  
    "00": {  
      "title": "I simply love loops.js 🌟",  
      "image": "https://images.unsplash.com/photo-1465...  
    },  
    "01": {  
      "title": "Slow Mornings in Bed ☕",  
      "image": "https://images.unsplash.com/photo-1642...  
    },  
    "02": {  
      "title": "Every Day is Flower Day 🌸",  
      "image": "https://images.unsplash.com/photo-1642...  
    }  
  }  
}
```

- One big object with objects.
- All objects have a unique key.
- “posts”, “00”, “01” and “02” are all keys.
- “posts” is the key of the collection of post objects (list of posts).
- “00”, “01” and “02” are keys of post objects.
- “title” and “image” are keys inside of every post object. “title” and “image” contains values.
- Remember a property contains a key and a value.

```
{  
  "posts": {  
    "00": {  
      "title": "I simply love loops.js 🌟",  
      "image": "https://images.unsplash.com/photo-1465...  
    },  
    "01": {  
      "title": "Slow Mornings in Bed ☕",  
      "image": "https://images.unsplash.com/photo-1642...  
    },  
    "02": {  
      "title": "Every Day is Flower Day 🌸",  
      "image": "https://images.unsplash.com/photo-1642...  
    }  
  }  
}
```

- Just think of an array of objects (posts array with post objects).
- We are just using an object to hold all the objects instead of an array.
- It's kind of a dictionary. The posts object contains unique property names (keys) which easily can be "looked up".
- Then every post object (with title and image props) can be returned:
  - `["posts"]["02"]`
  - `["posts"]["00"]`
  - `["posts"]["01"]`
  - `["posts"]["01"]["title"]`

key and name of the collections



- All posts: <https://post-rest-api-default.firebaseio.com/posts.json>
- One specific post: <https://post-rest-api-default.firebaseio.com/posts/00.json>
- And specific prop: <https://post-rest-api-default.firebaseio.com/posts/00/title.json>

key and name of the collections

{

"posts": {

"00": {

    "title": "I simply love loops.js 🌟",  
    "image": "<https://images.unsplash.com/photo-1465111111111111>

},

"01": {

    "title": "Slow Mornings in Bed ☕",  
    "image": "<https://images.unsplash.com/photo-1642111111111111>

},

"02": {

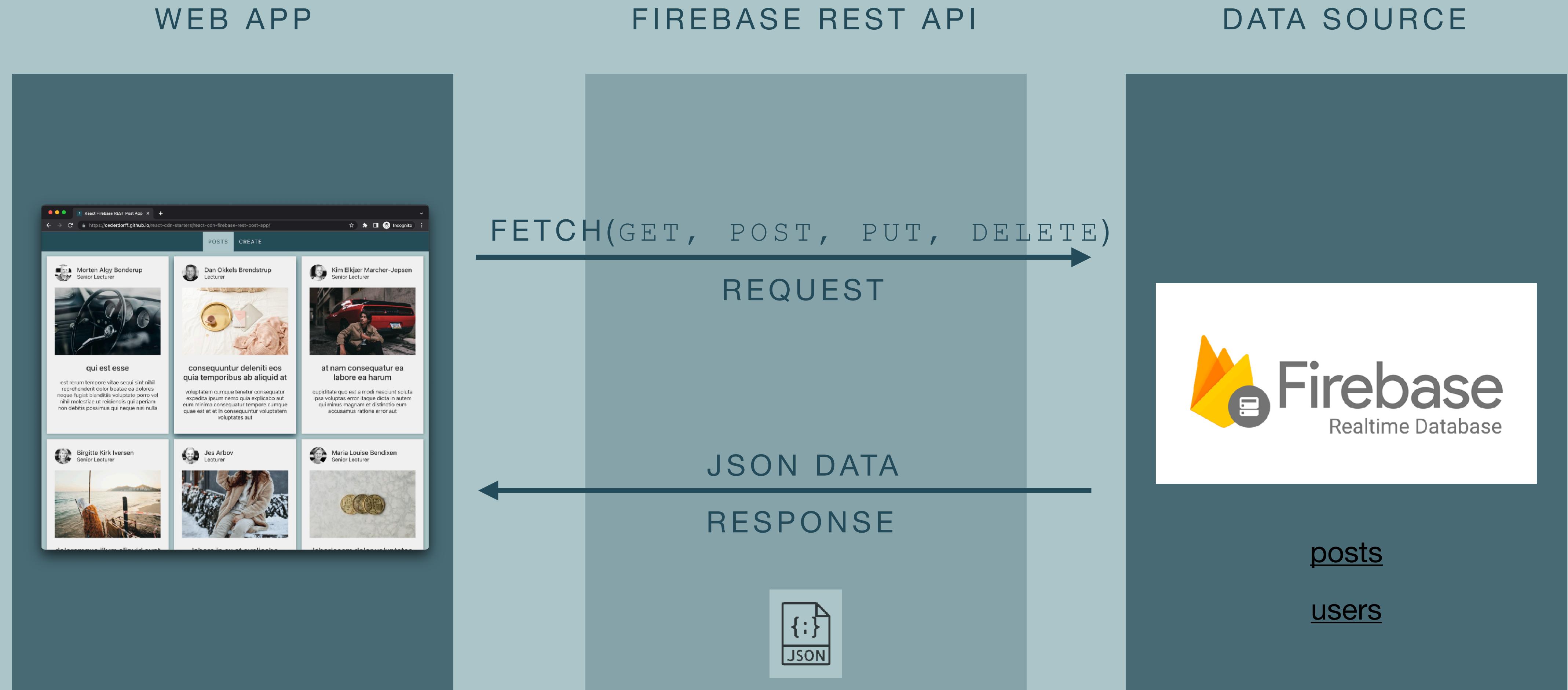
    "title": "Every Day is Flower Day 🌸",  
    "image": "<https://images.unsplash.com/photo-1642111111111111>

}



key of the objects

# Fetch, HTTP Request & Response



# Realtime Database REST API

[https://firebase.google.com/docs/  
database/rest/start](https://firebase.google.com/docs/database/rest/start)

```
export default function PostsPage() {
  const [posts, setPosts] = useState([]);
  const [showLoader, dismissLoader] = useIonLoading();

  async function getPosts() {
    const response = await fetch("https://race-rest-default-rtbd.firebaseio.com/posts.json");
    const data = await response.json();
    // map object into an array with objects
    const postsArray = Object.keys(data).map(key => ({ id: key, ...data[key] }));
    return postsArray;
  }

  async function getUsers() {
    const response = await fetch("https://race-rest-default-rtbd.firebaseio.com/users.json");
    const data = await response.json();
    // map object into an array with objects
    const users = Object.keys(data).map(key => ({ id: key, ...data[key] }));
    return users;
  }
}
```



Code  
Every  
Day