

Fetch, CRUD & REST

Let's turn on the Database!



Frontend (client)

User CRUD with Node.js REST API

Birgitte Kirk Iversen
Senior Lecturer
bki@mail.dk

Rasmus Cederdorff
Senior Lecturer
race@mail.dk

Dan Okkels Brendstrup
Lecturer
dob@mail.dk

Kasper Fischer Topp
Lecturer
kato@mail.dk

UPDATE DELETE

UPDATE DELETE

UPDATE DELETE

UPDATE DELETE

Backend (Server)

```
Raw    Parsed
```

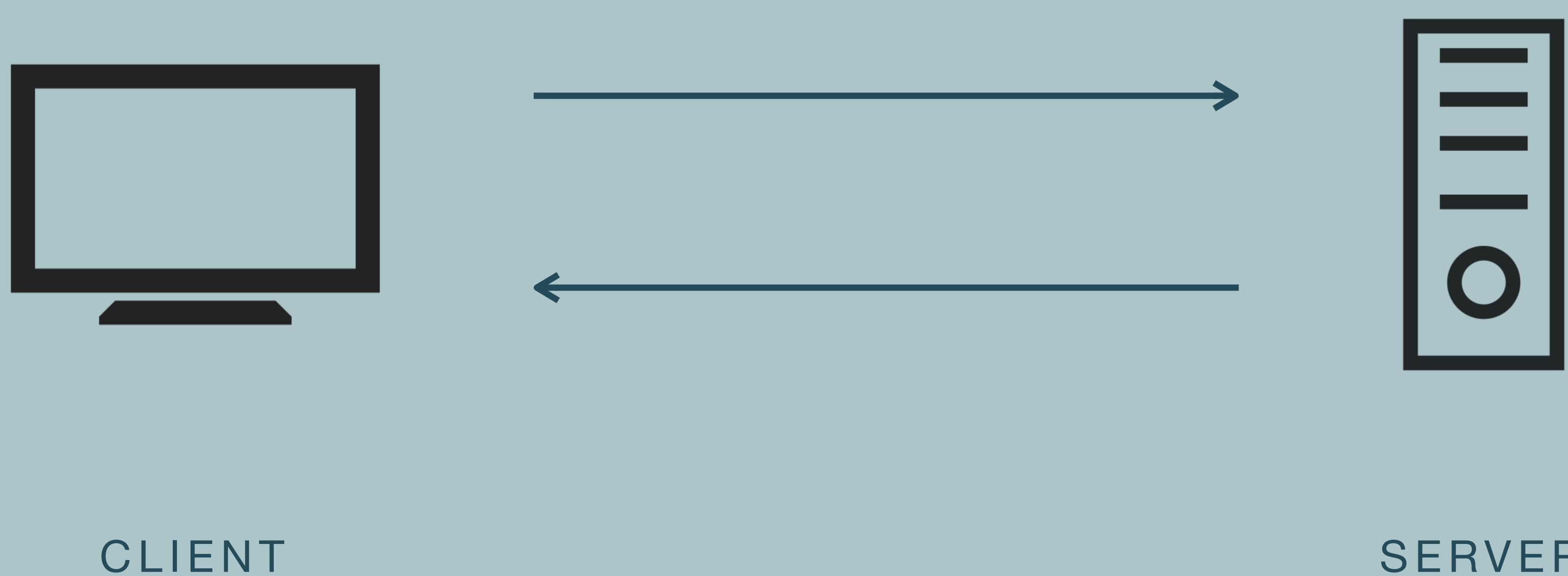
```
[{"id": 1, "name": "Birgitte Kirk Iversen", "mail": "bki@mail.dk", "title": "Senior Lecturer", "image": "https://www.baaa.dk/media/u4gorzsd/birgitte-kirk-iversen2.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921541630000&format=webp"}, {"id": 4, "name": "Rasmus Cederdorff", "mail": "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/bdoje141/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"}]
```

<https://github.com/cederdorff/rest-user-crud-frontend>

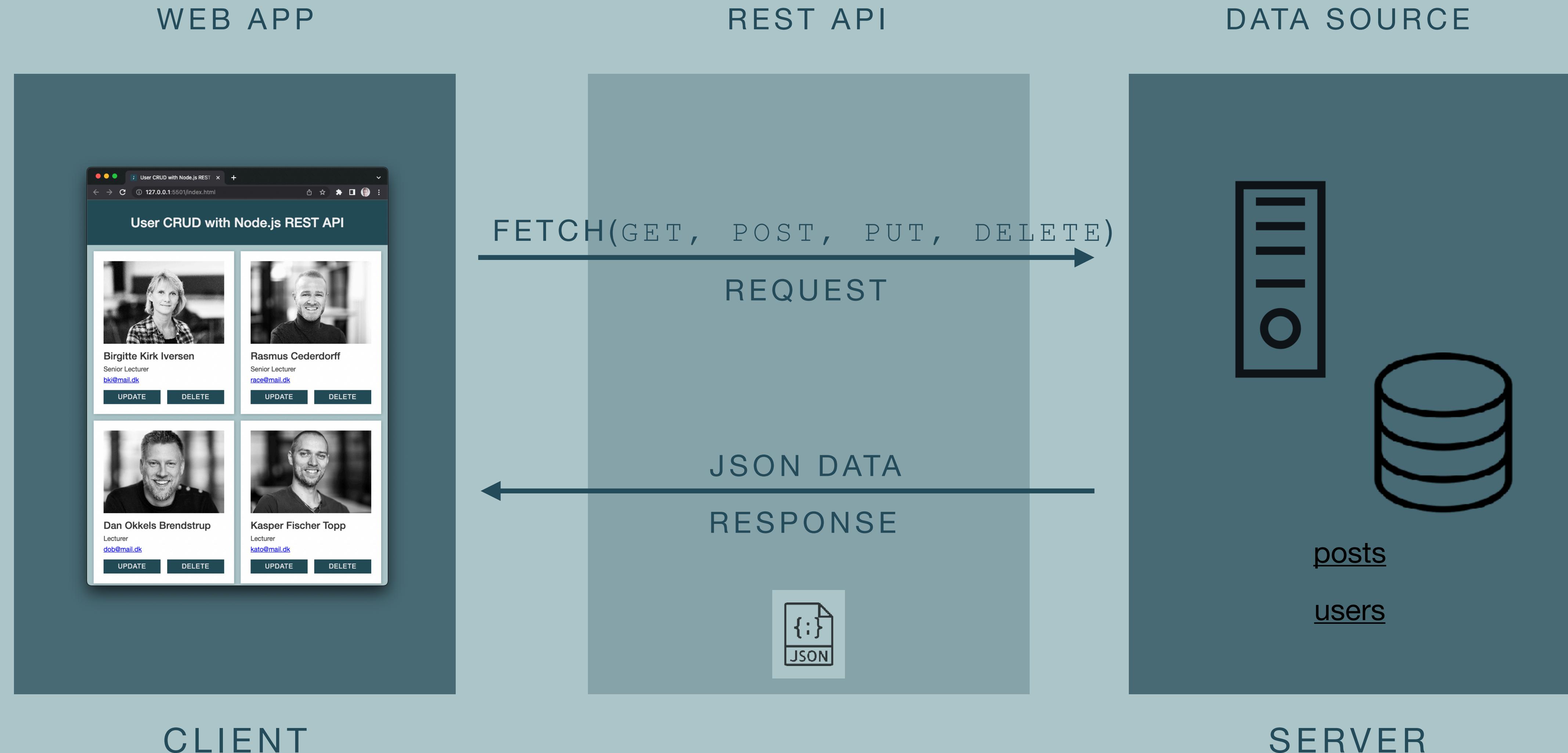
<https://github.com/cederdorff/rest-user-crud-backend>

Client-Server Model

... how to communicate with backend/ data source (server)

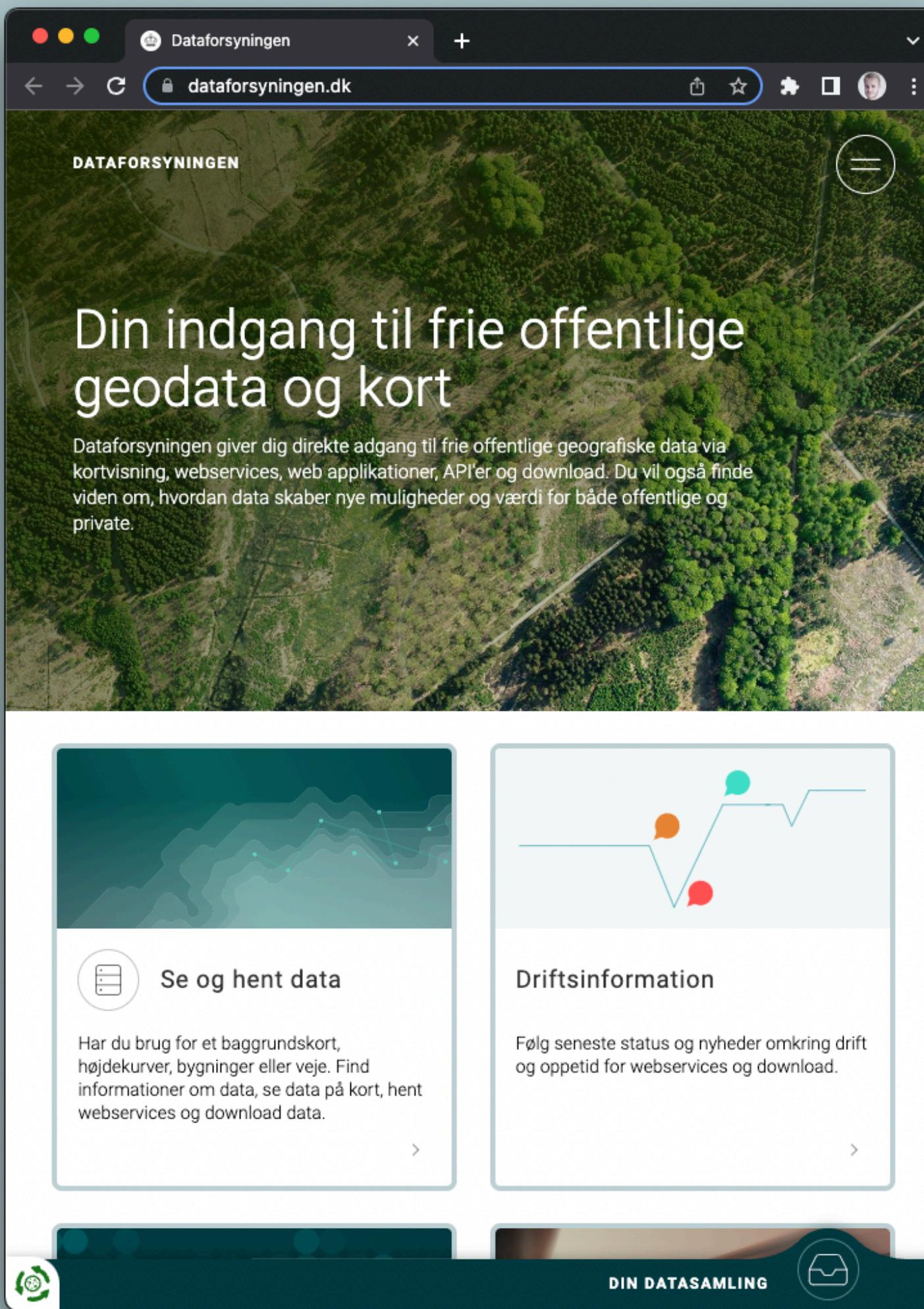


Fetch, HTTP Request & Response



Dataforsyningen REST API

... an example of a REST API

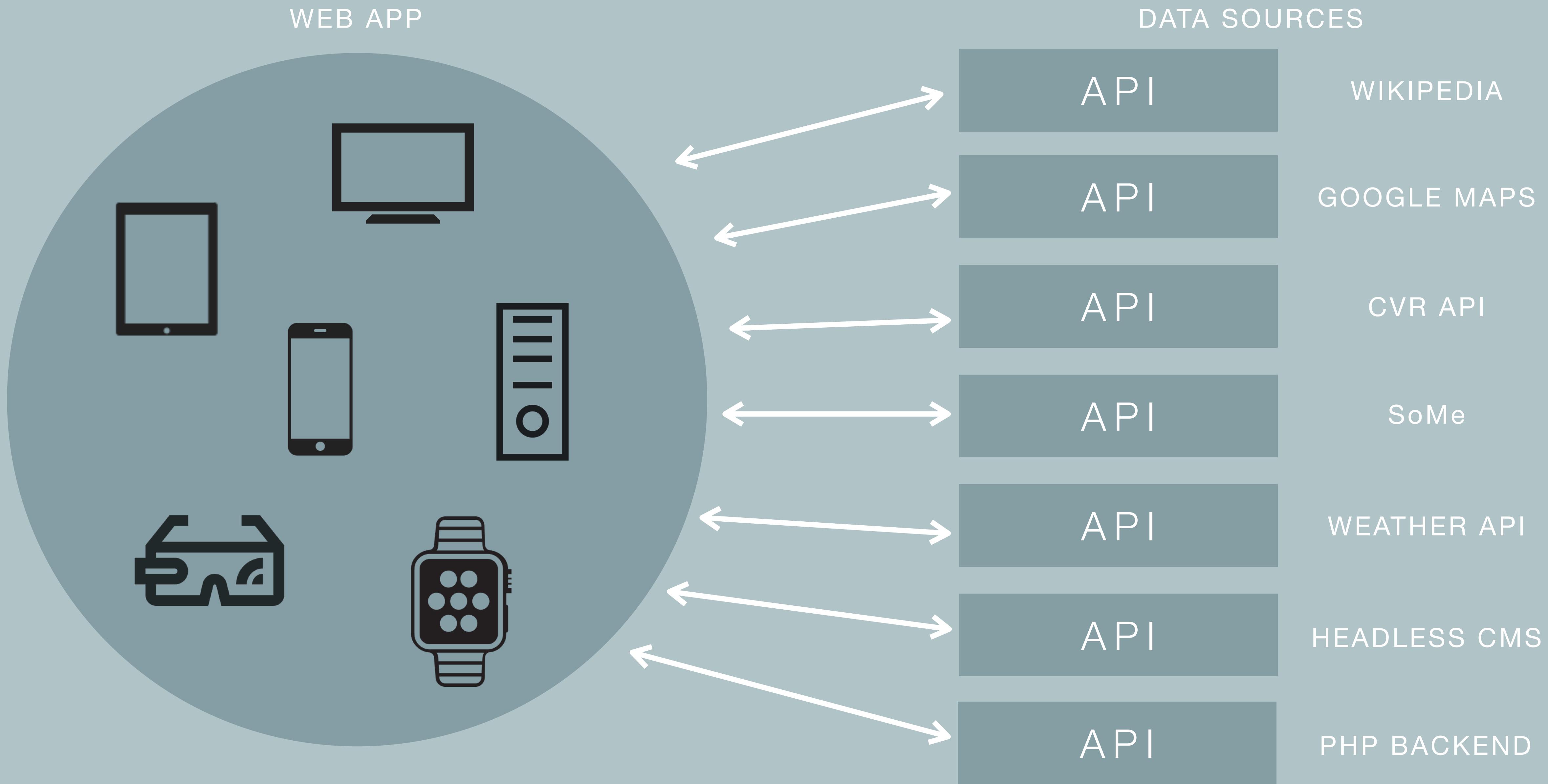


<https://dataforsyningen.dk/>

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](#)

API



React CRUD

React Firebase REST Post App https://race-rest.web.app

POSTS CREATE

 Morten Algy Bonderup
Senior Lecturer



Qui est esse

Est rerum tempore vitae sequi sint nihil reprehenderit dolor beatae ea dolores neque fugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis qui aperiam non debitis possimus qui neque nisi nulla

 Dan Okkels Brendstrup
Lecturer



Consequuntur deleniti eos quia temporibus ab aliquid at

Voluptatem cumque tenetur consequatur expedita ipsum nemo quia explicabo aut eum minima consequatur tempore cumque quae est et et in consequuntur voluptatem voluptates aut

 Kim Elkjær Marcher-Jepsen
Senior Lecturer



At nam consequatur ea labore ea harum

Cupiditate quo est a modi nesciunt soluta ipsa voluptas error itaque dicta in autem qui minus magnam et distinctio eum accusamus ratione error aut

 Birgitte Kirk Iversen
Senior Lecturer



Jes Arbov
Lecturer



 Maria Louise Bendixen
Senior Lecturer

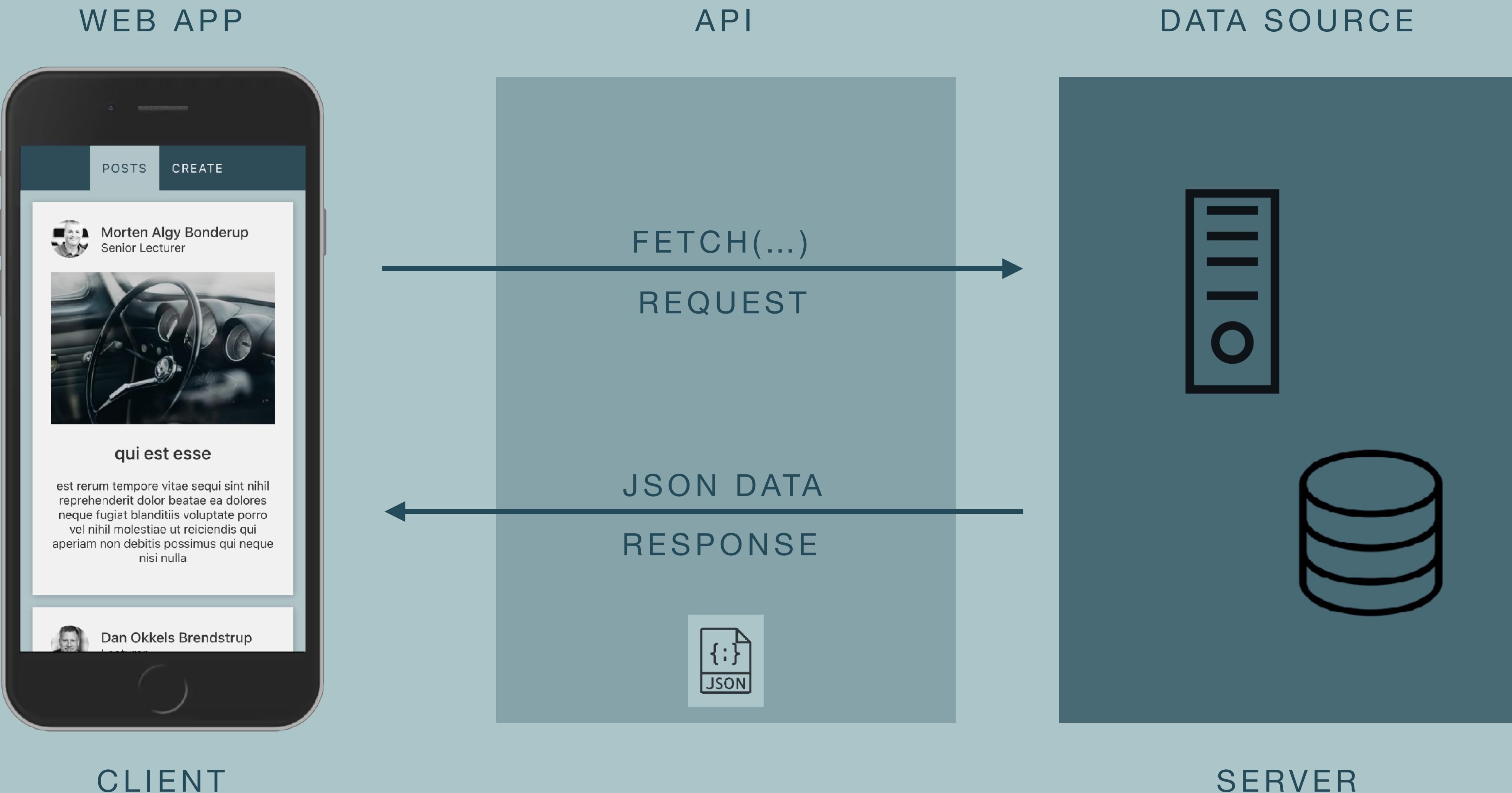


<https://race-rest.web.app/>

Content

- Client-Server Model
- 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?

Web Development



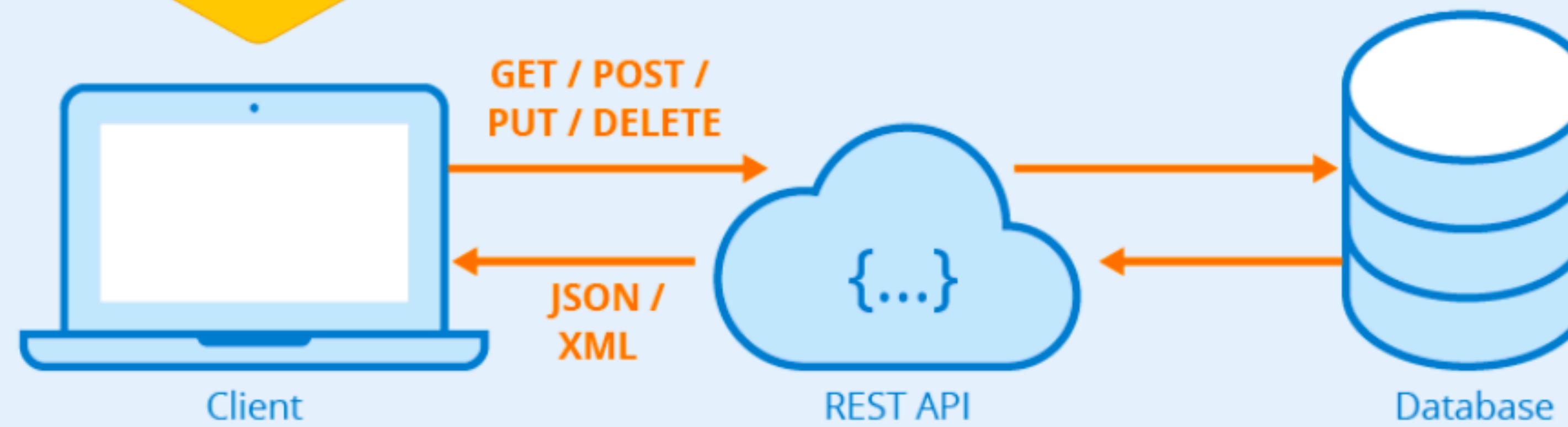
When someone asks you how to get data from a database in a js code



made with mematic

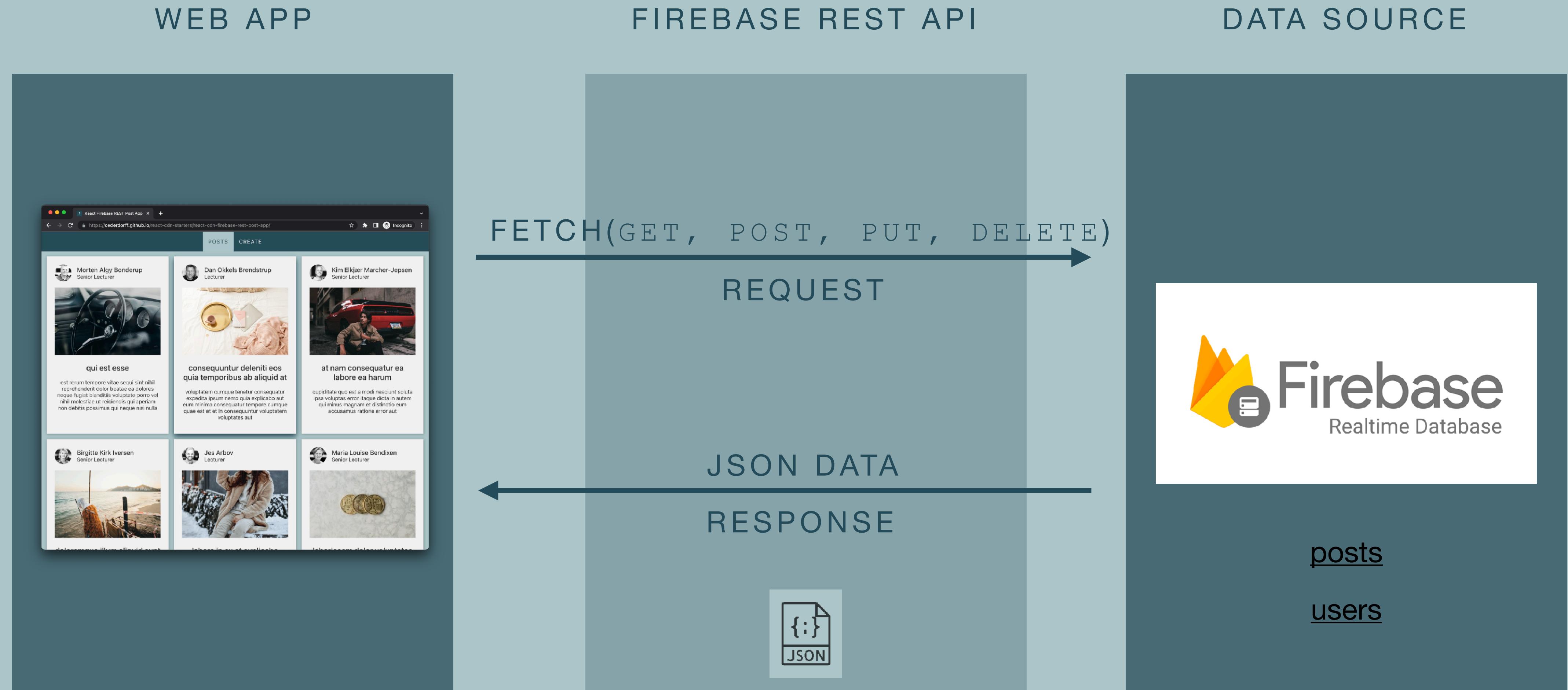


Firebase





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(...)

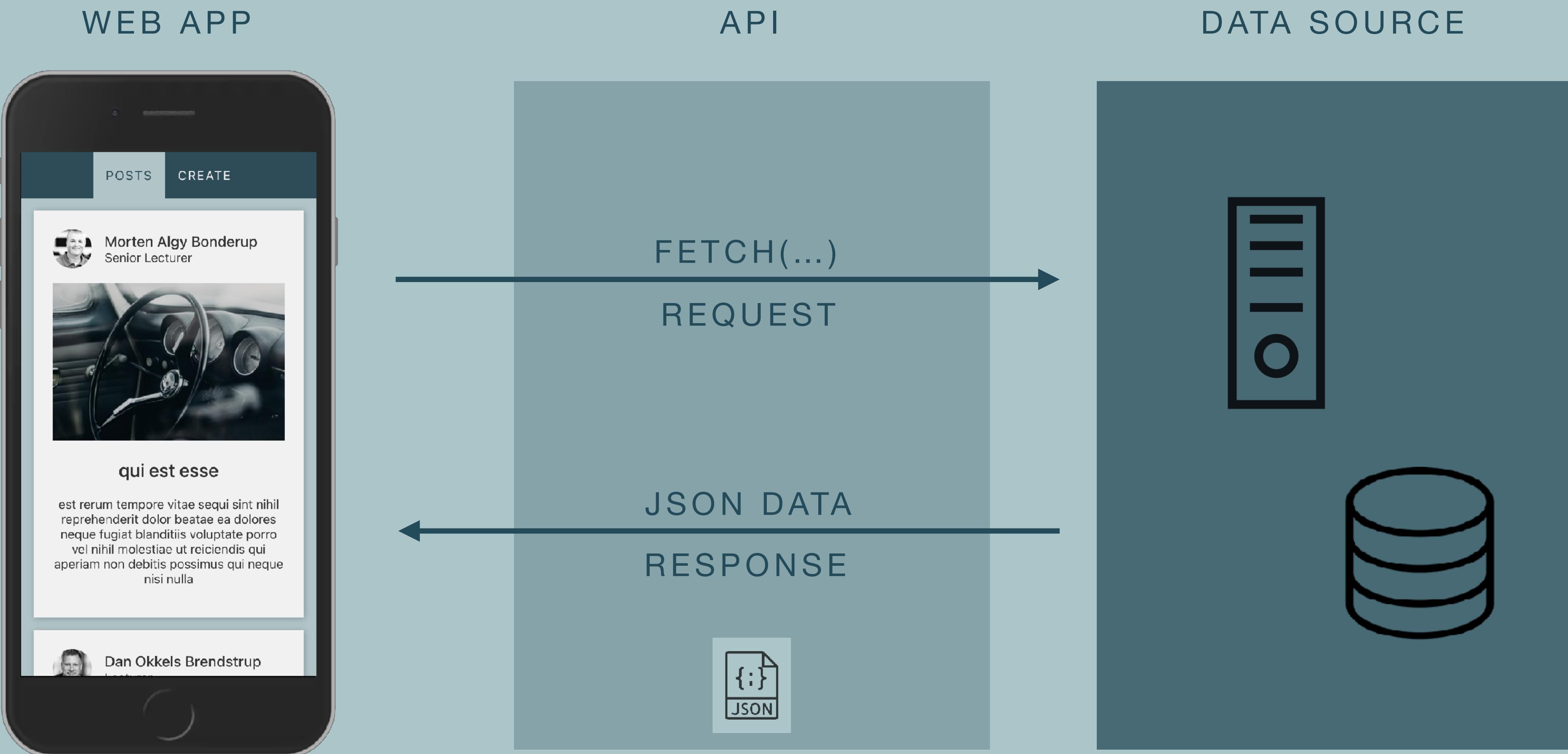
... get & post data from and to a data source
... can perform network requests to a server

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

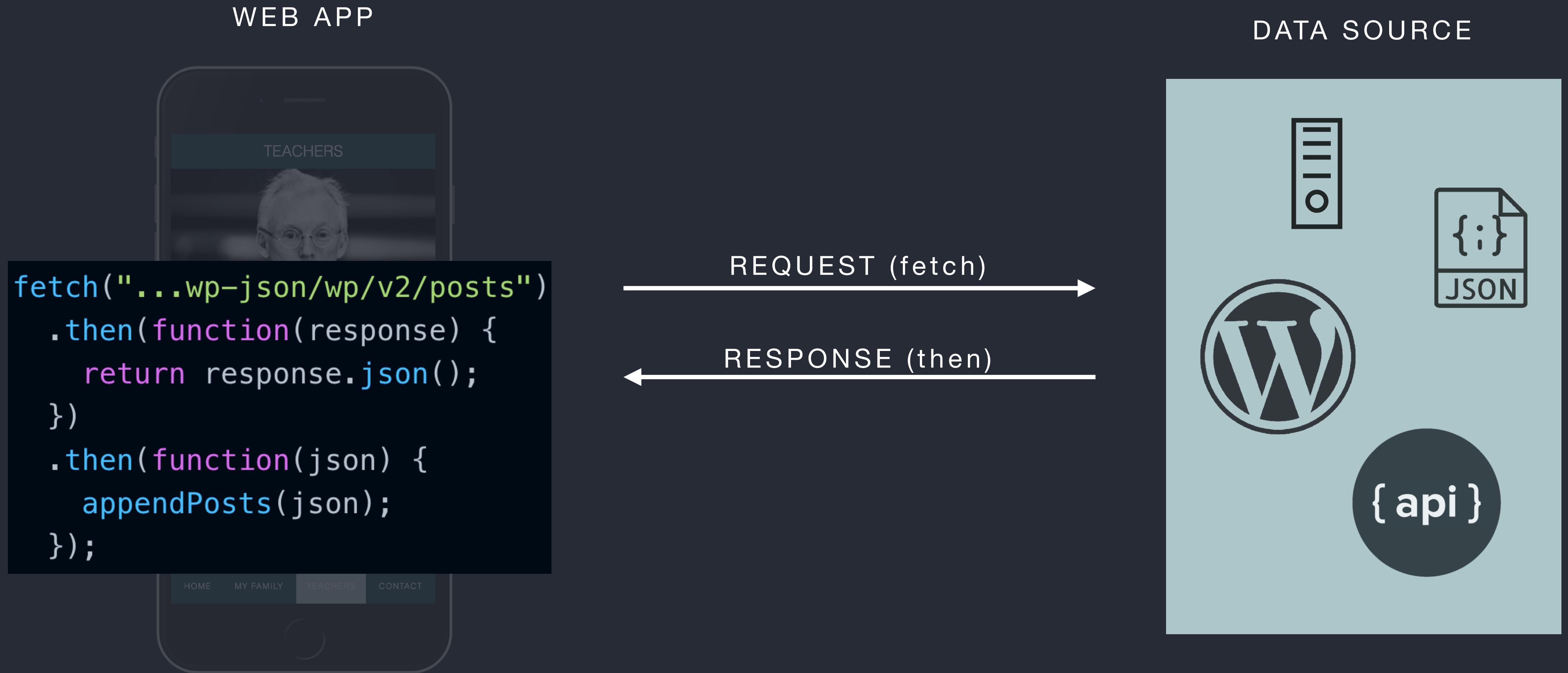
- `url` – the URL to access.
- `options` – optional parameters: method, headers etc.

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"
},
{
```

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.

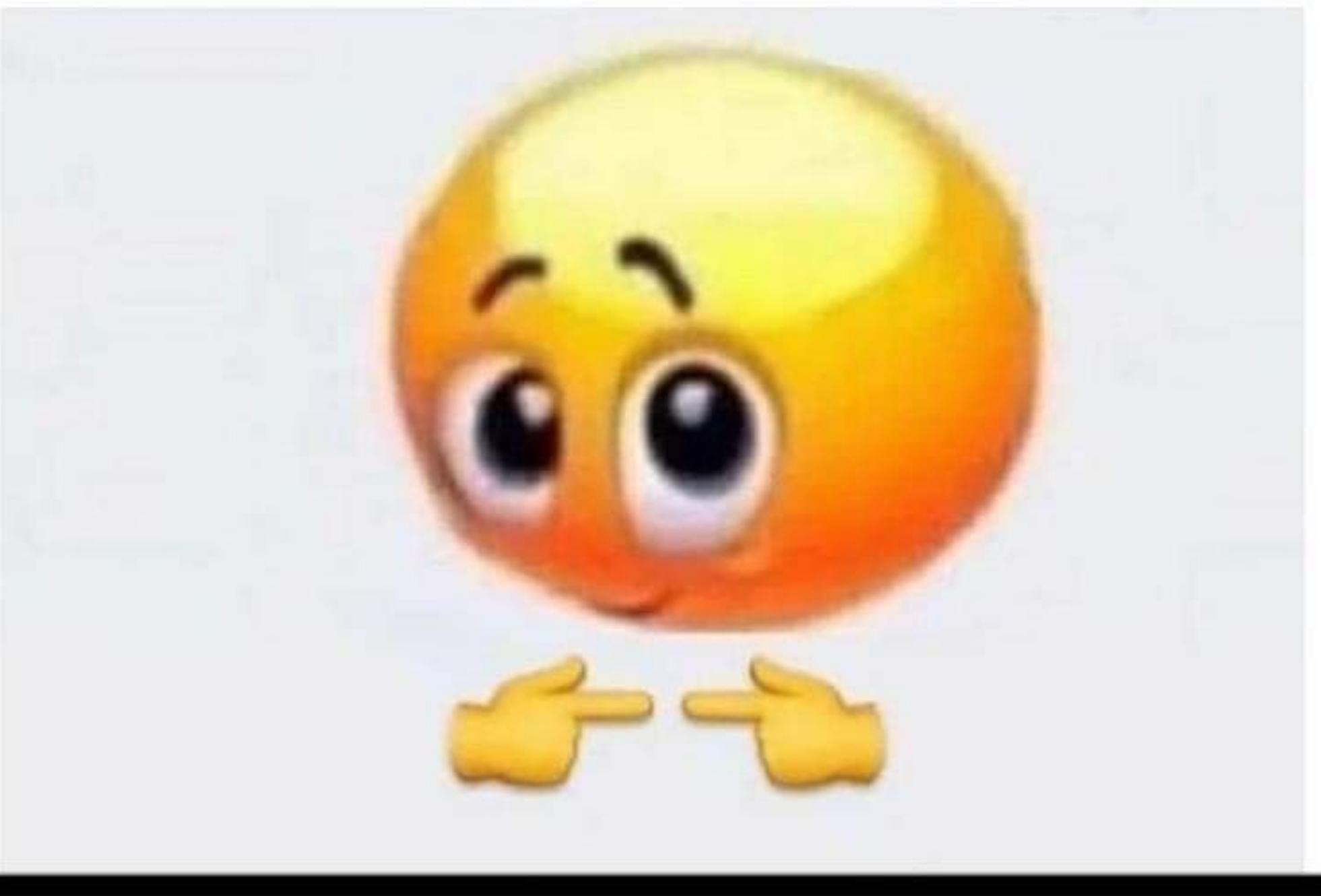
```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json")
  .then(function (response) {
    return response.json();
  })
  .then(function (data) {
    console.log(data);
  });

```

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

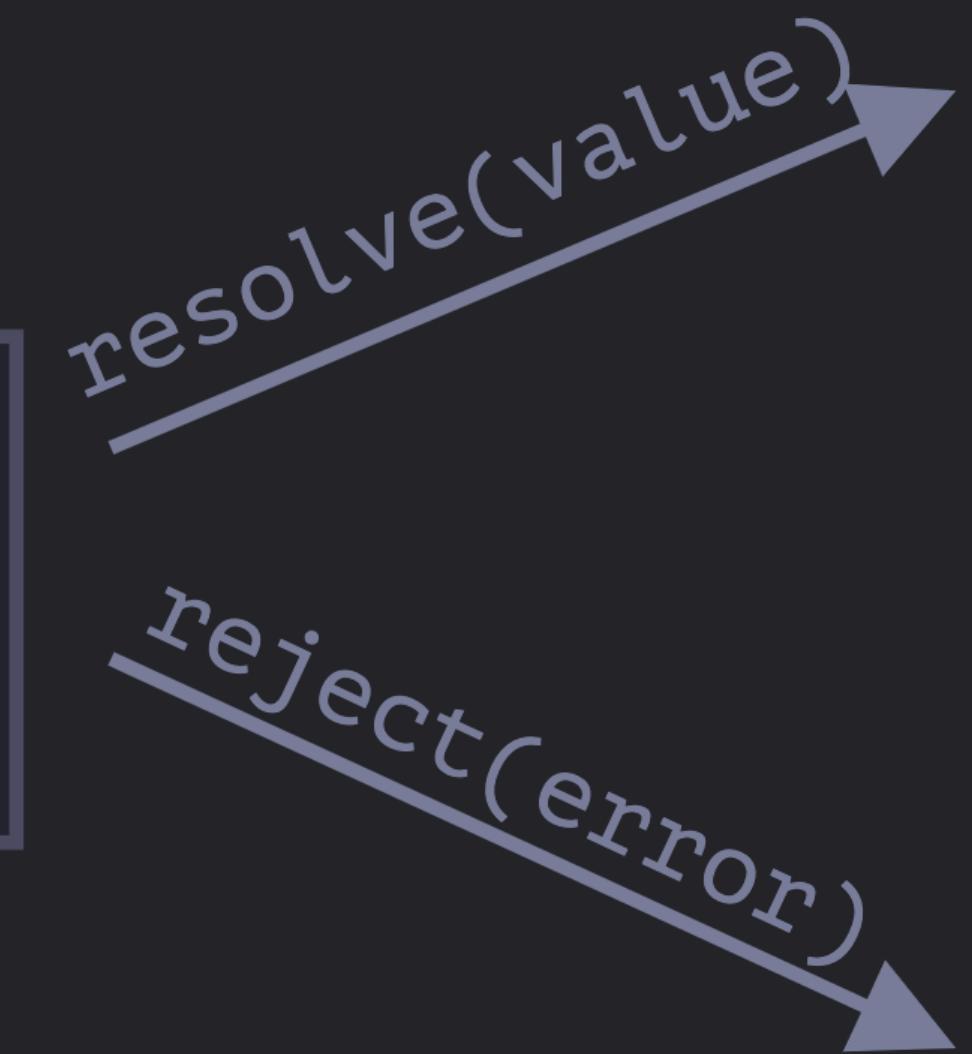
We can use `await` to wait for `fetch` to finish



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

```
new Promise(executor)
```

state: "pending"
result: undefined



state: "fulfilled"
result: value

state: "rejected"
result: error

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>
  );
}
```

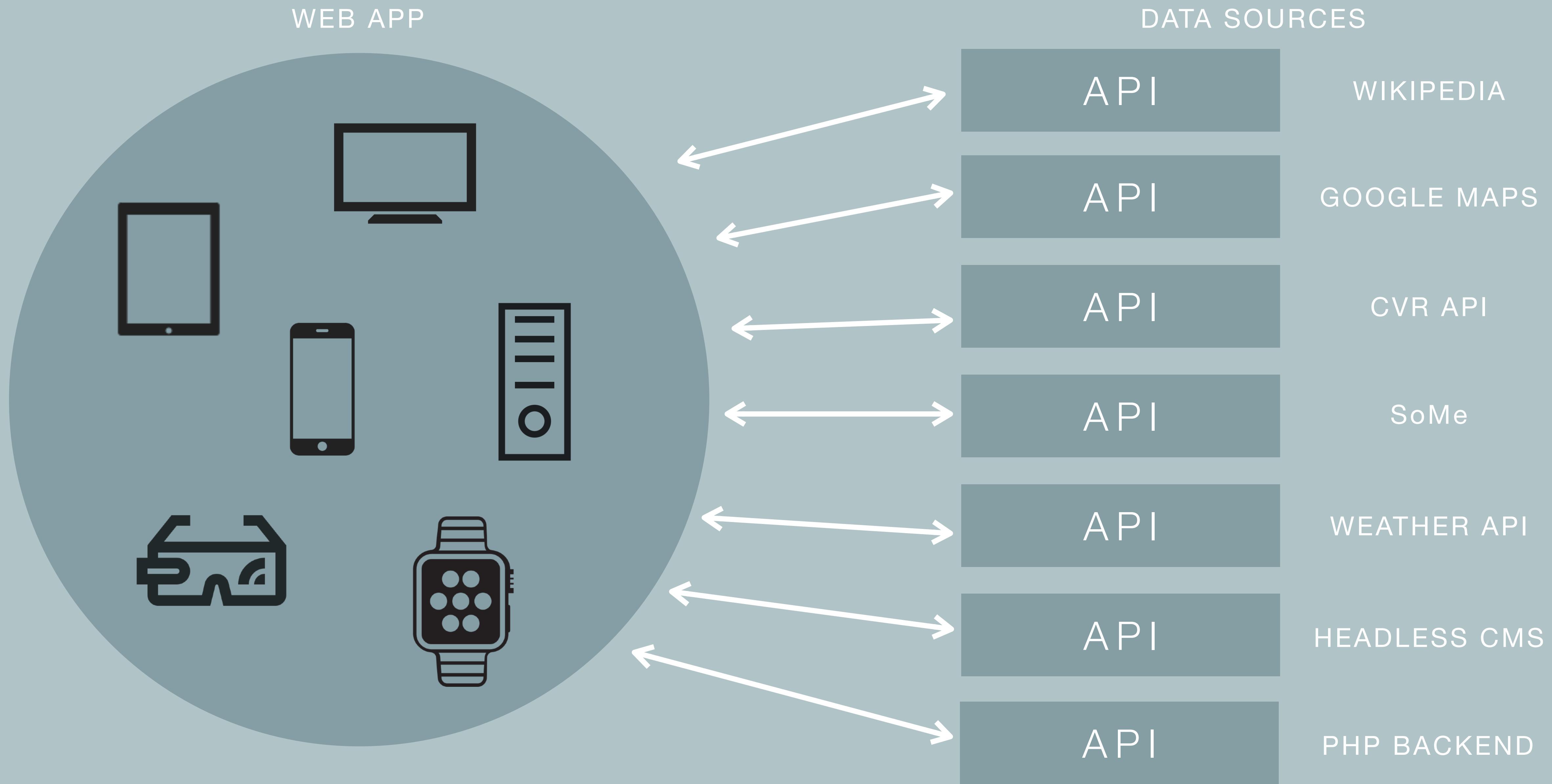
WHAT IS A DATA SOURCE?

- 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

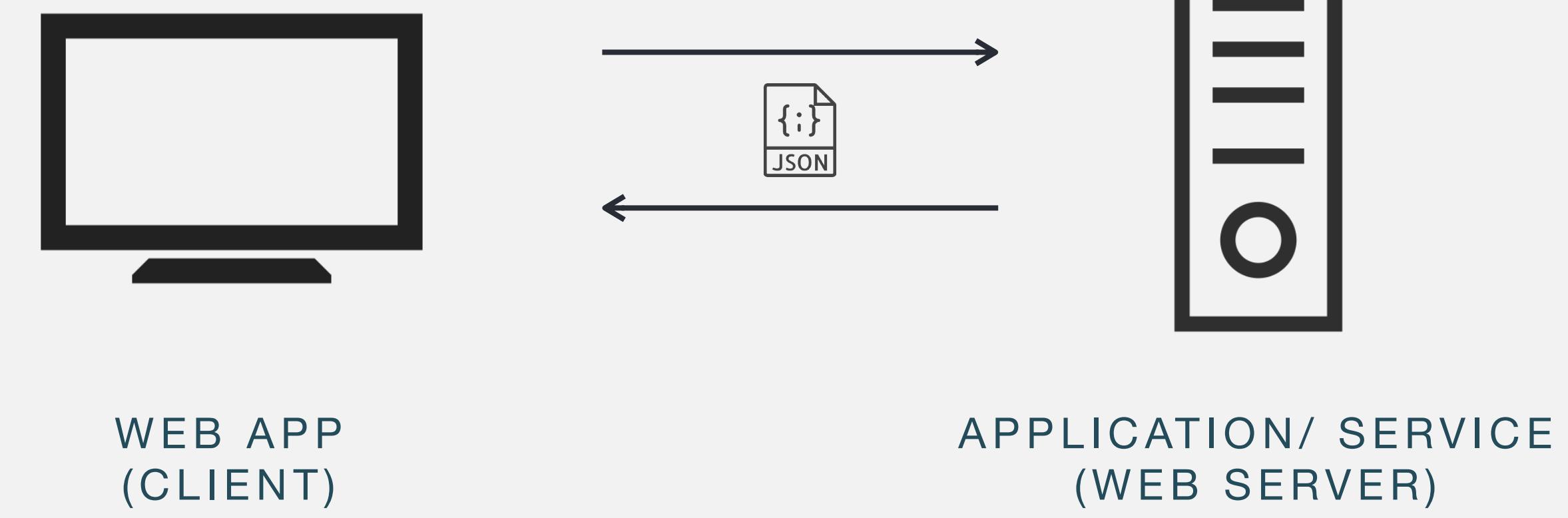
GET, POST & INTERACT WITH DATA & CONTENT

WHAT IS AN API?

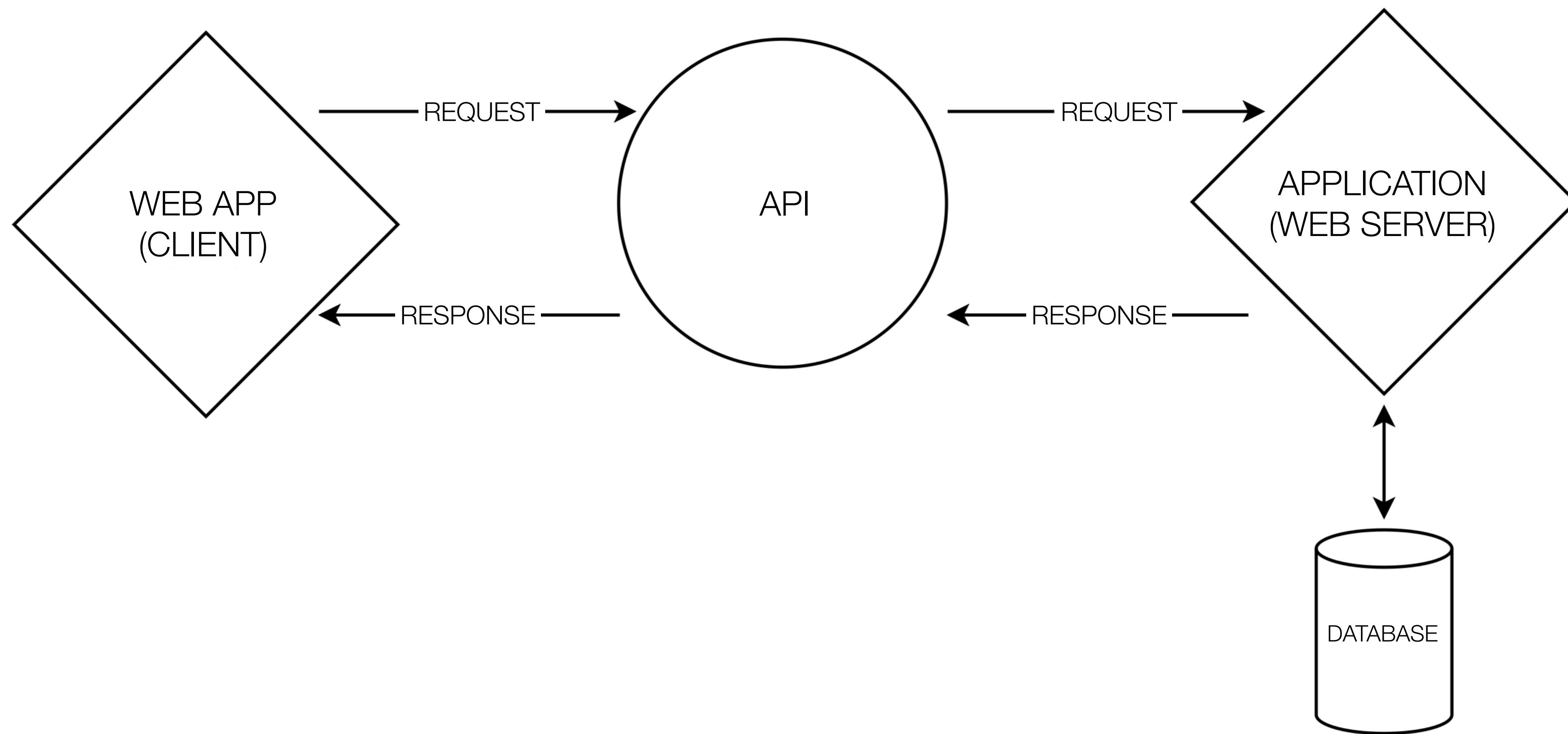
<https://www.youtube.com/watch?v=s7wmiS2mSXY>

APPLICATION PROGRAMMING INTERFACE

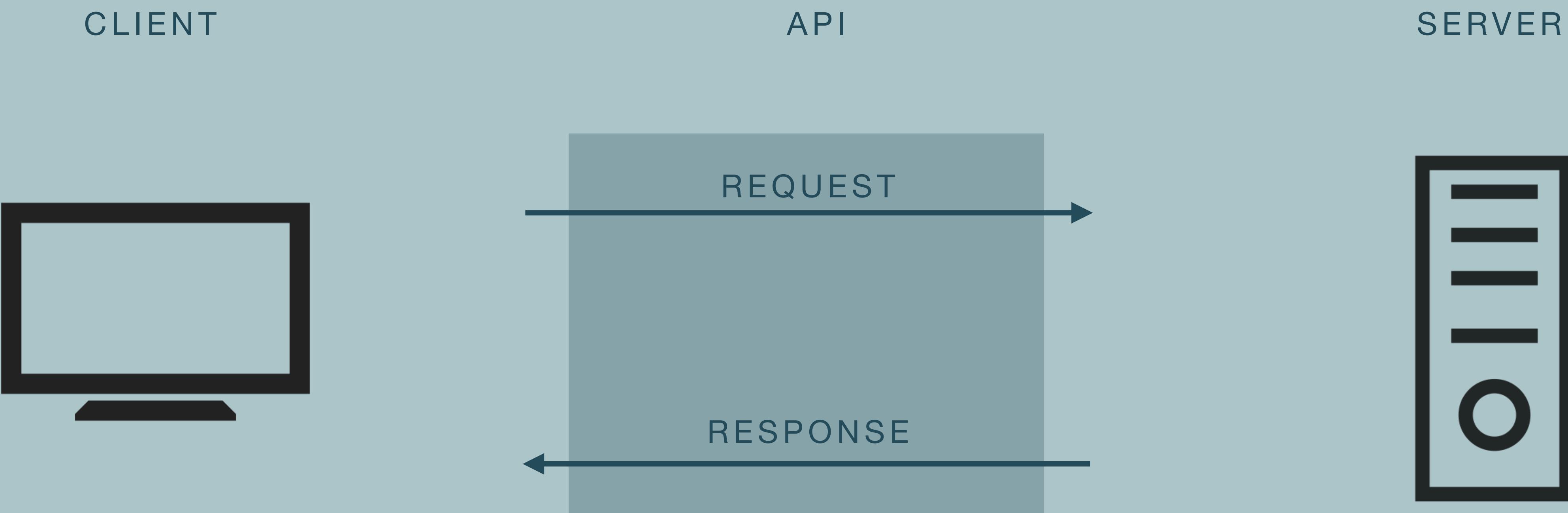
- An interface that makes it possible to access data in a controlled way.
- Communication between two (web) applications = Makes is possible for two programs to interact with each other.
- Platform independent: Can be used by different clients, devices and users: websites, web apps, mobile apps, webshops and other clients.
- The client does not need to know anything about the service or program provided by the API and visa versa.



APPLICATION PROGRAMMING INTERFACE

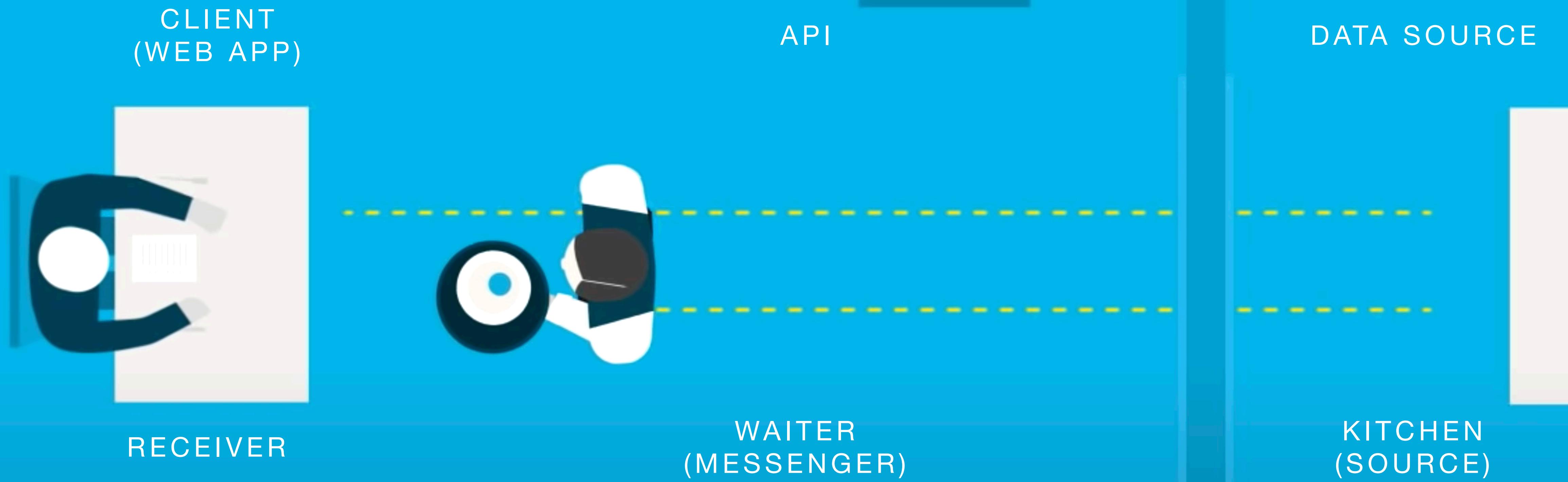


APPLICATION PROGRAMMING INTERFACE





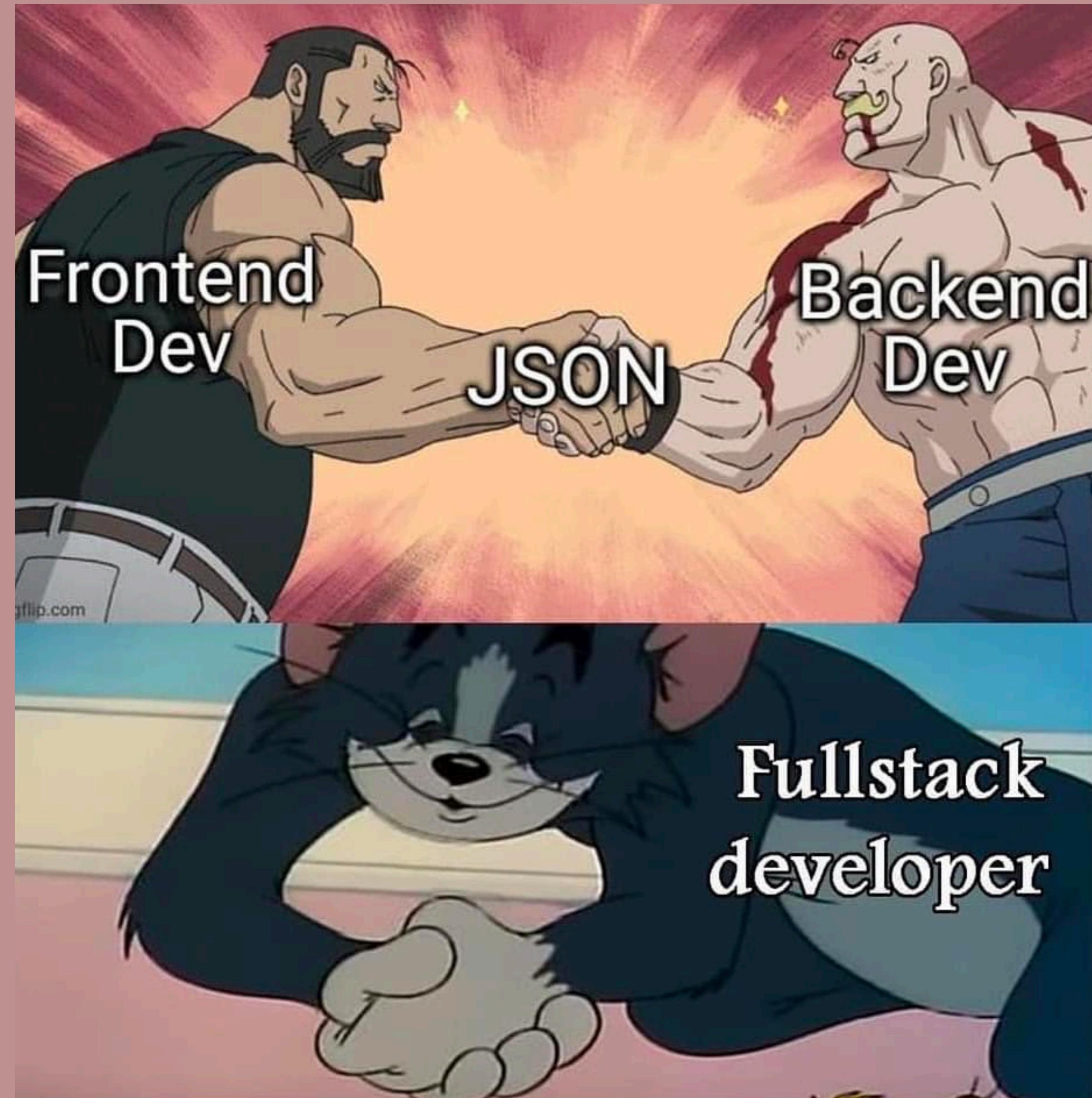
<https://www.youtube.com/watch?v=s7wmiS2mSXY>



JSON

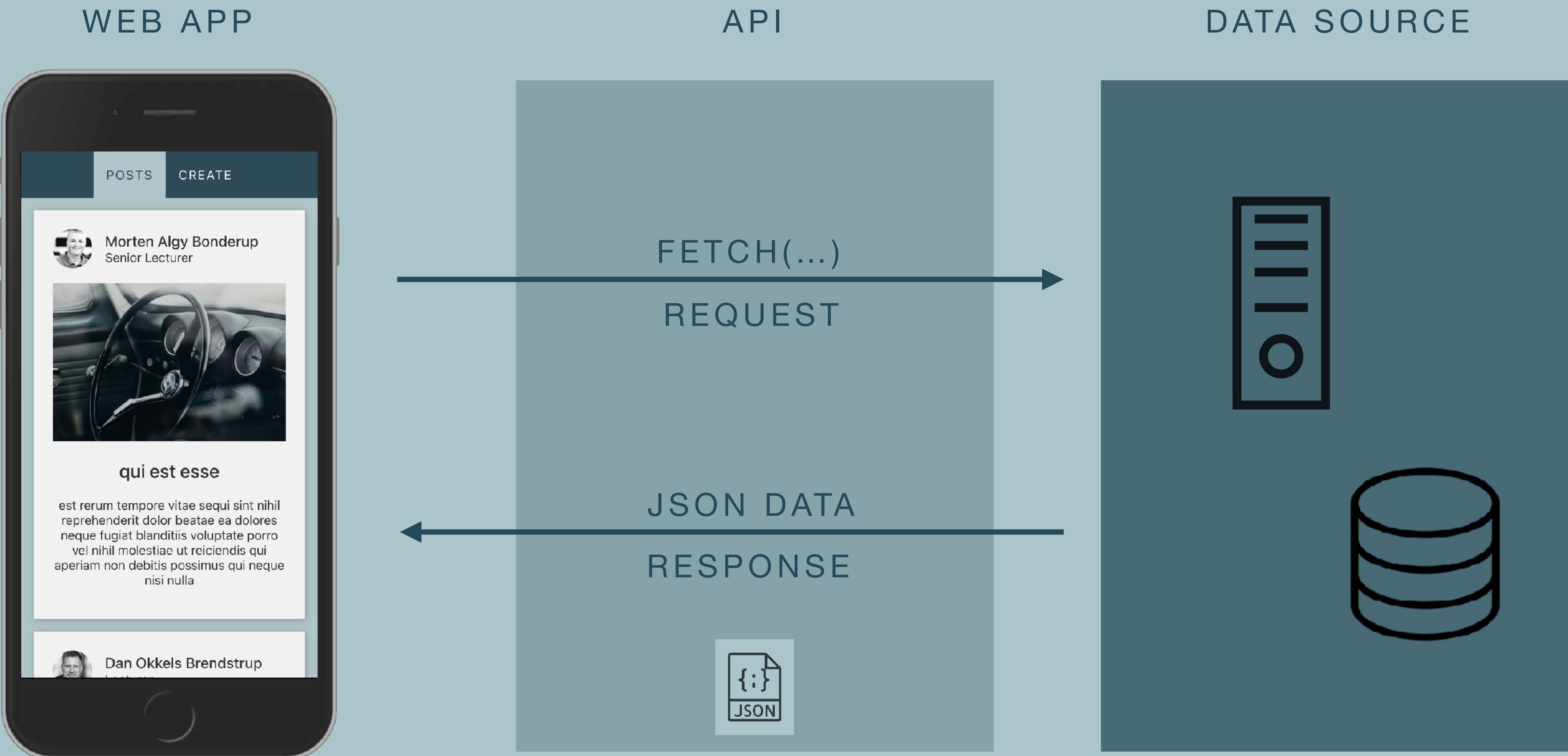
JavaScript Object Notation

... 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

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

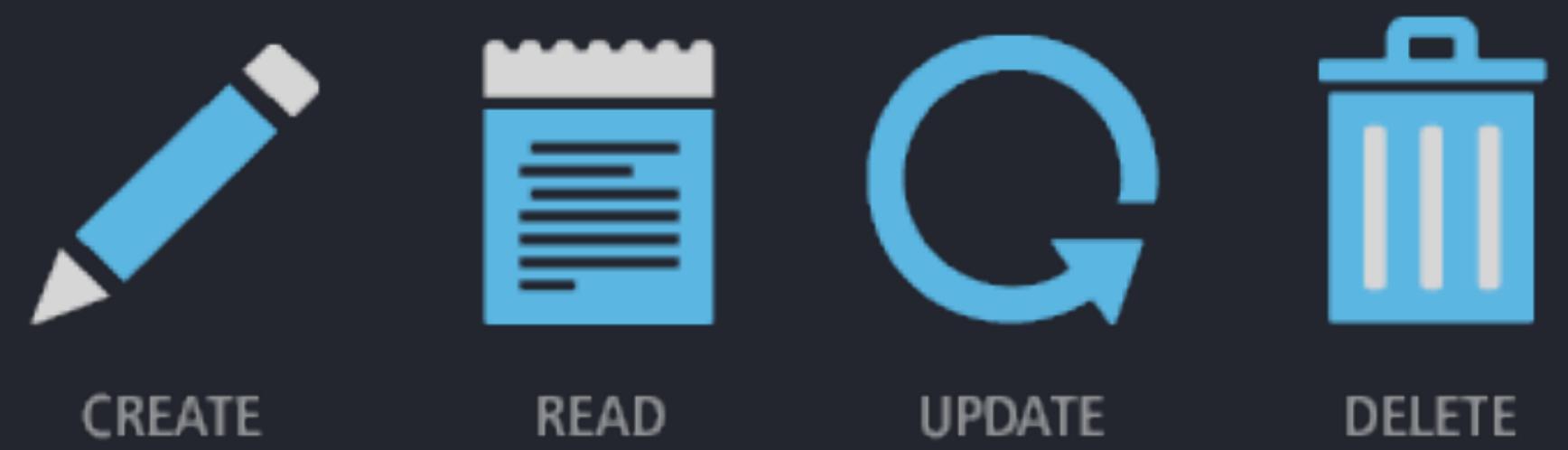
```
const user = {  
    name: "John",  
    age: 30,  
    gender: "male",  
    lookingFor: "female"  
};  
  
// === JSON.stringify === //  
const jsonUser = JSON.stringify(user);  
console.log(jsonUser); // {"name":"John","age":30,"gender":"male","lookingFor":"female"}  
  
// === JSON.parse ===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

Fetch and create post

newPost object is parsed
to JSON and posted to the
ressource wrapped on
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

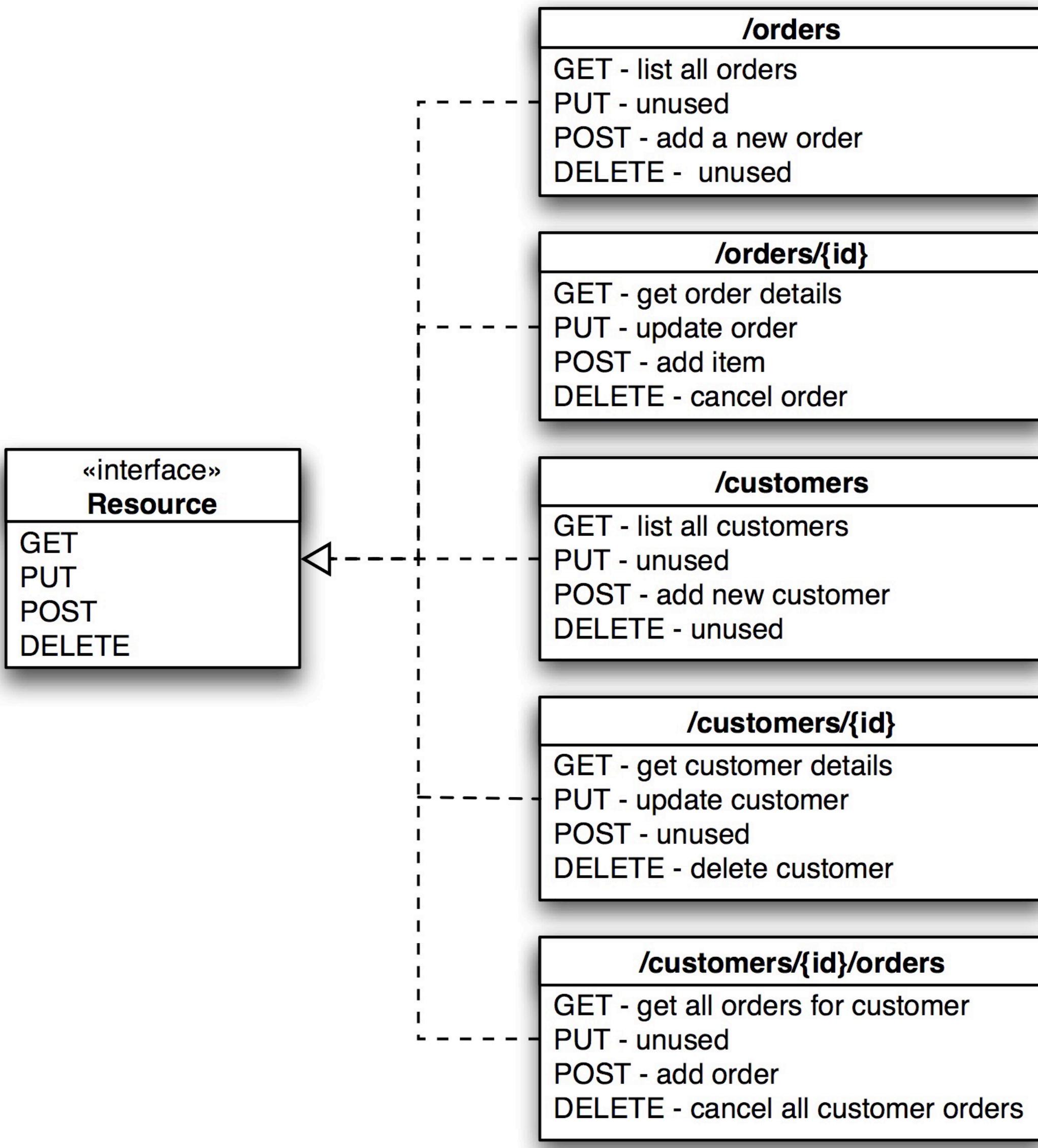
DELETE

- 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.

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-kald til to perform create, read, update and delete with the HTTP methods GET, POST, PUT and DELETE

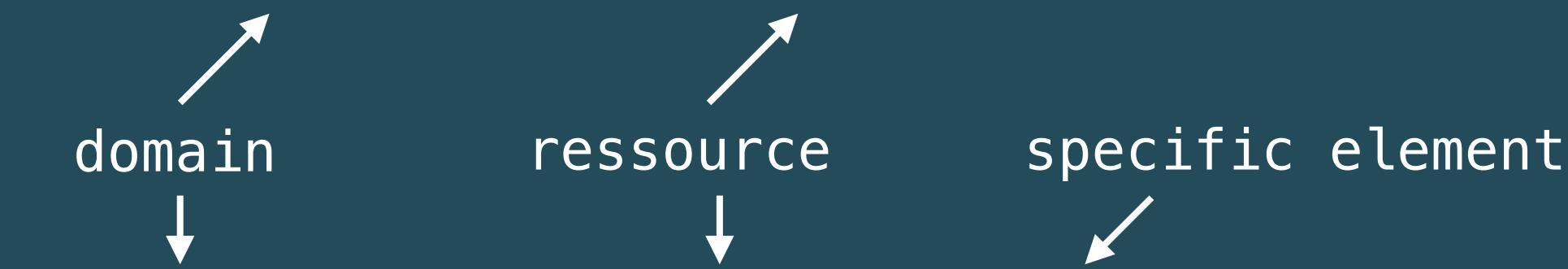
Principles of REST



- Ressources and endpoints:
 - <http://example.com/products>
 - <http://example.com/users>
 - <http://domain.com/orders>
- IDs:
 - <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

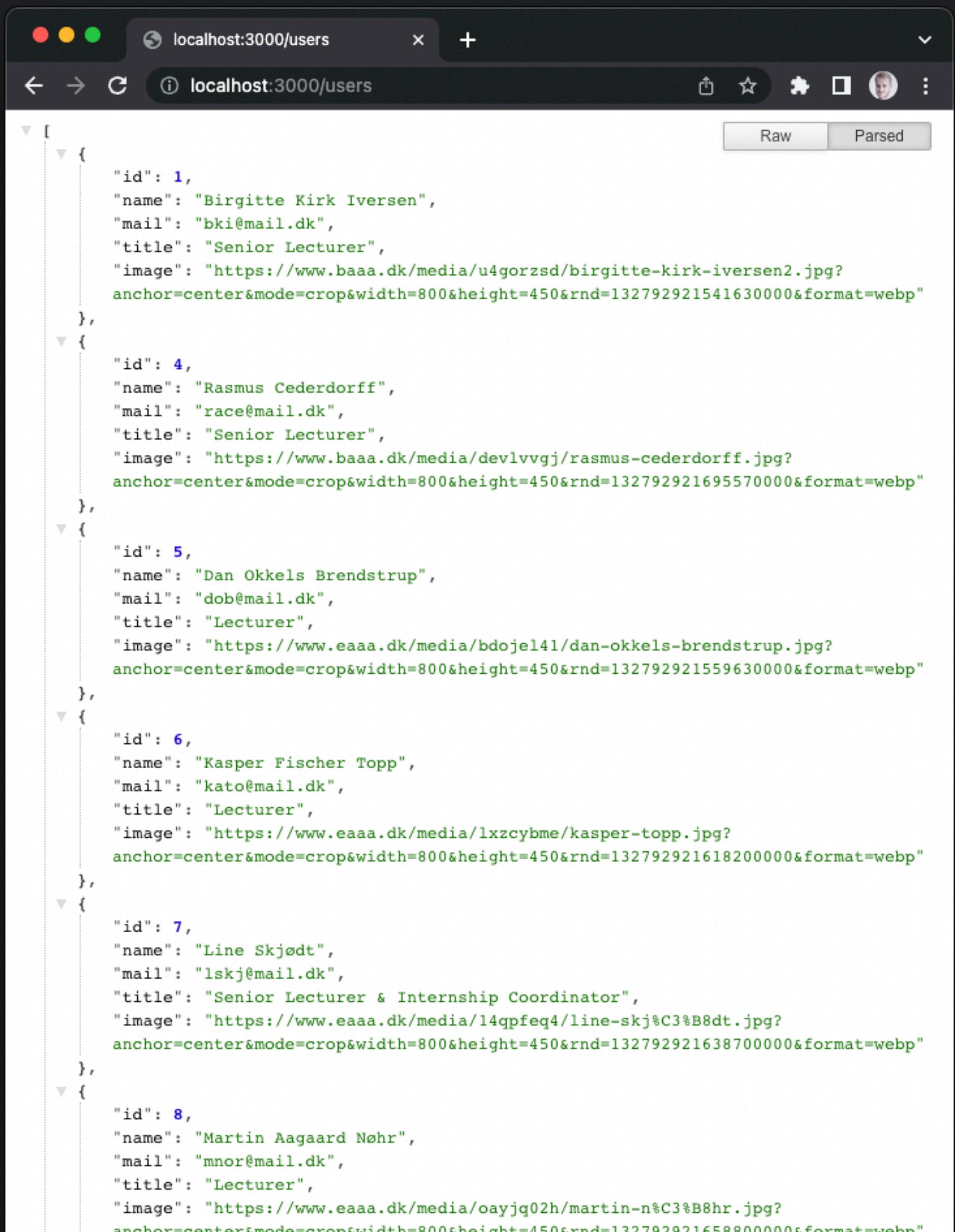


- http://some-url.com/products/product1

- 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 |

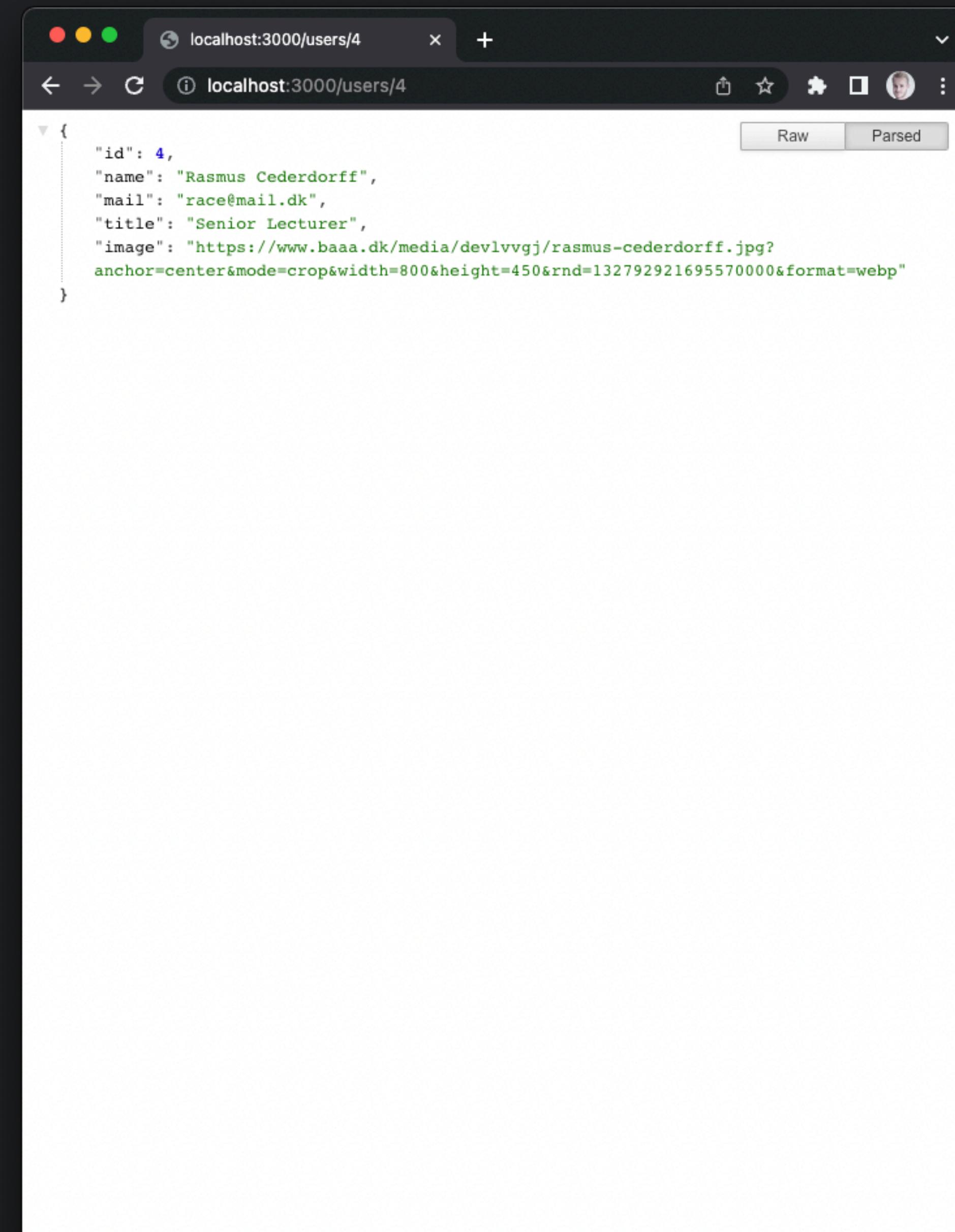
Collections (JSON Array)



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 a "Raw" and "Parsed" button.

```
[{"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)



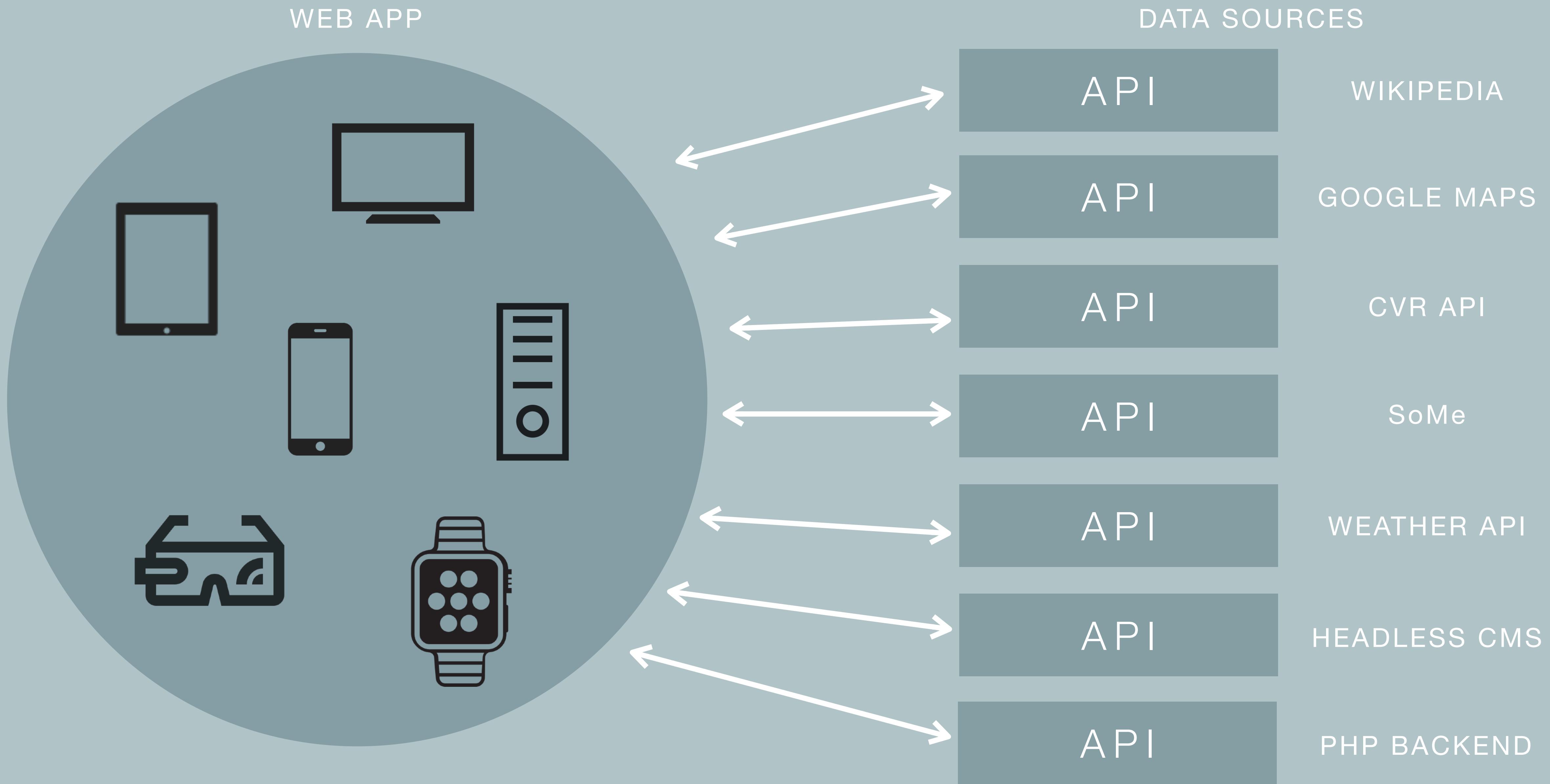
A screenshot of a web browser window titled "localhost:3000/users/4". The page displays a single JSON object representing a user with id 4. The object includes fields: id, name, mail, title, and image. The image field includes a URL and a query string for image processing. The browser interface shows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

```
{ "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://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

CRUD vs REST & HTTP Verbs

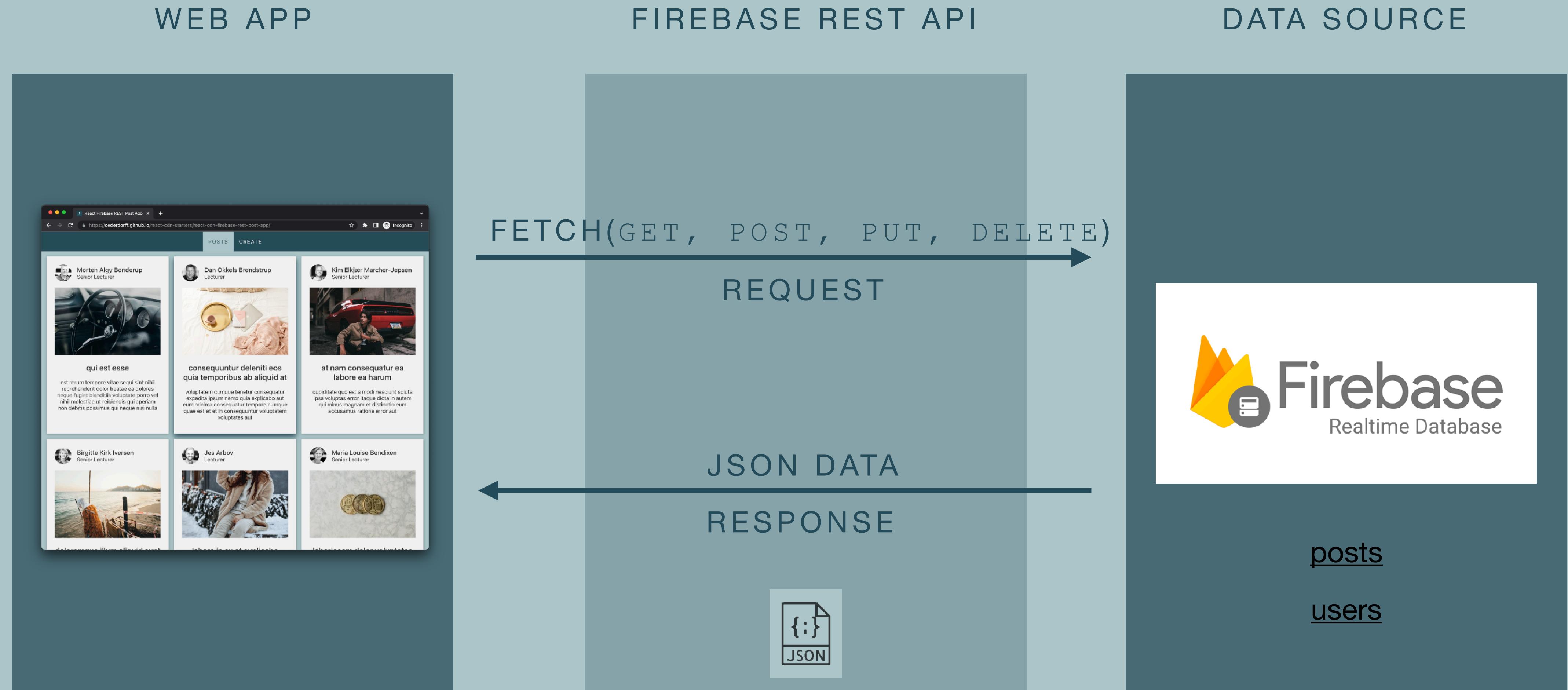
CREATE -> POST: create a new resource (object)

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

REQUEST body

When making HTTP requests we sometimes need to send data.

The data is wrapped inside of the request body.

The request body is one of the following:

a string (often JSON encoded string with data, object, arrays, etc.)

form data (form/multipart)

blob/ buffer source - binary data

URL search params (x-www-form-urlencoded)

<https://javascript.info/fetch#post-requests>

Today the slides is your documentation

Read and use
them carefully

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



REQUEST body

```
const url = "http://localhost:3000/json-api/posts/";
await fetch(url, {
  method: "POST",
  body: JSON.stringify(newPost)
});
```

```
async function savePost(postToUpdate) {
  const response = await fetch(url, {
    method: "PUT",
    body: JSON.stringify(postToUpdate)
  });
  const data = await response.json();
  console.log(data);
  navigate("/");
}
```

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"
});
```

Fetch & useEffect in React

In useEffect, perform the sideeffect (fetch) to get data from a data source.
And save the response data in a state using useState.

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

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

  return (
    <section className="page">
```

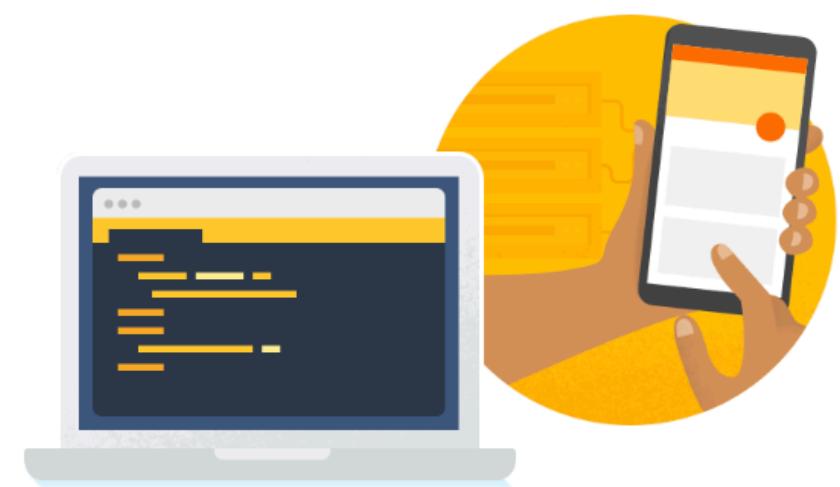


What is Firebase?

Platform & Backend-as-a-Service
for Web & App Development



Introducing Firebase



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



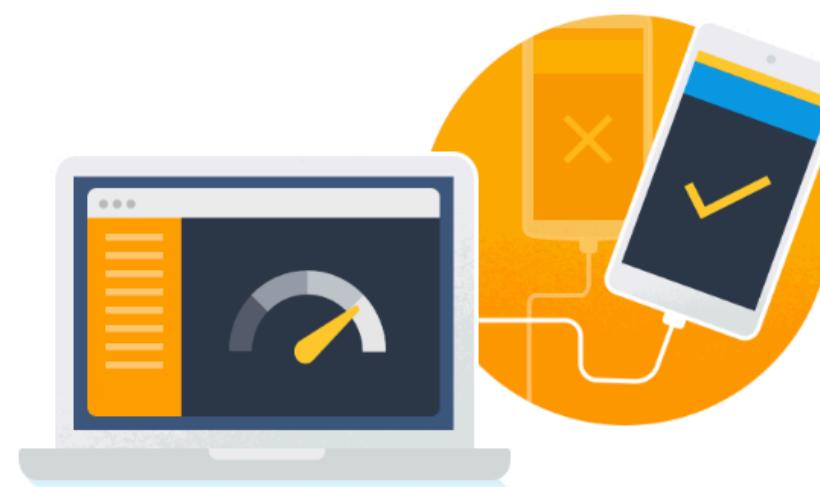
Cloud Storage

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



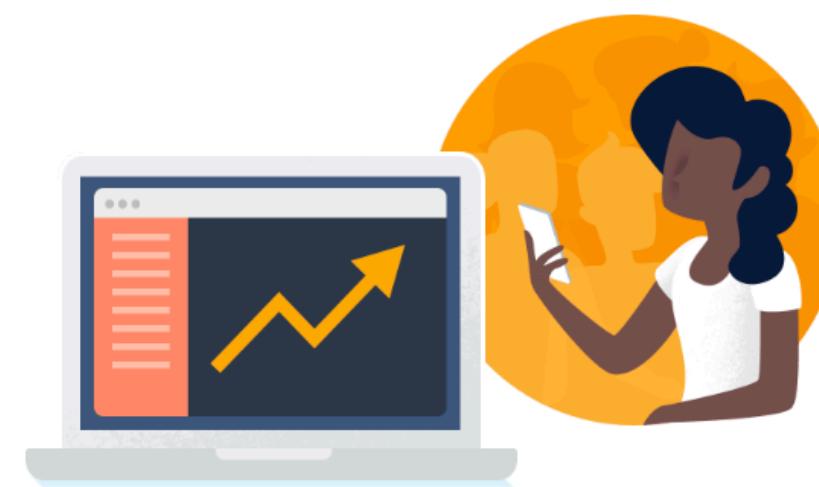
Test Lab

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

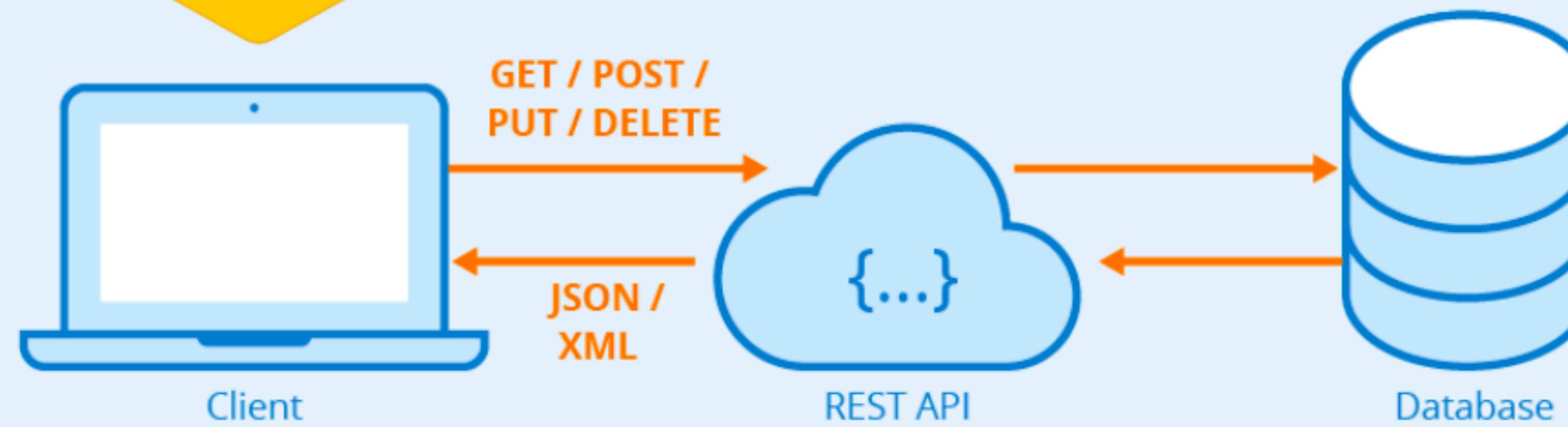
<https://firebase.google.com/products/realtime-database>



Database



Firebase



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