Fetch, JSON, CRUD & REST

Web Development

kea
KØBENHAVNS ERHVERVSAKADEMI

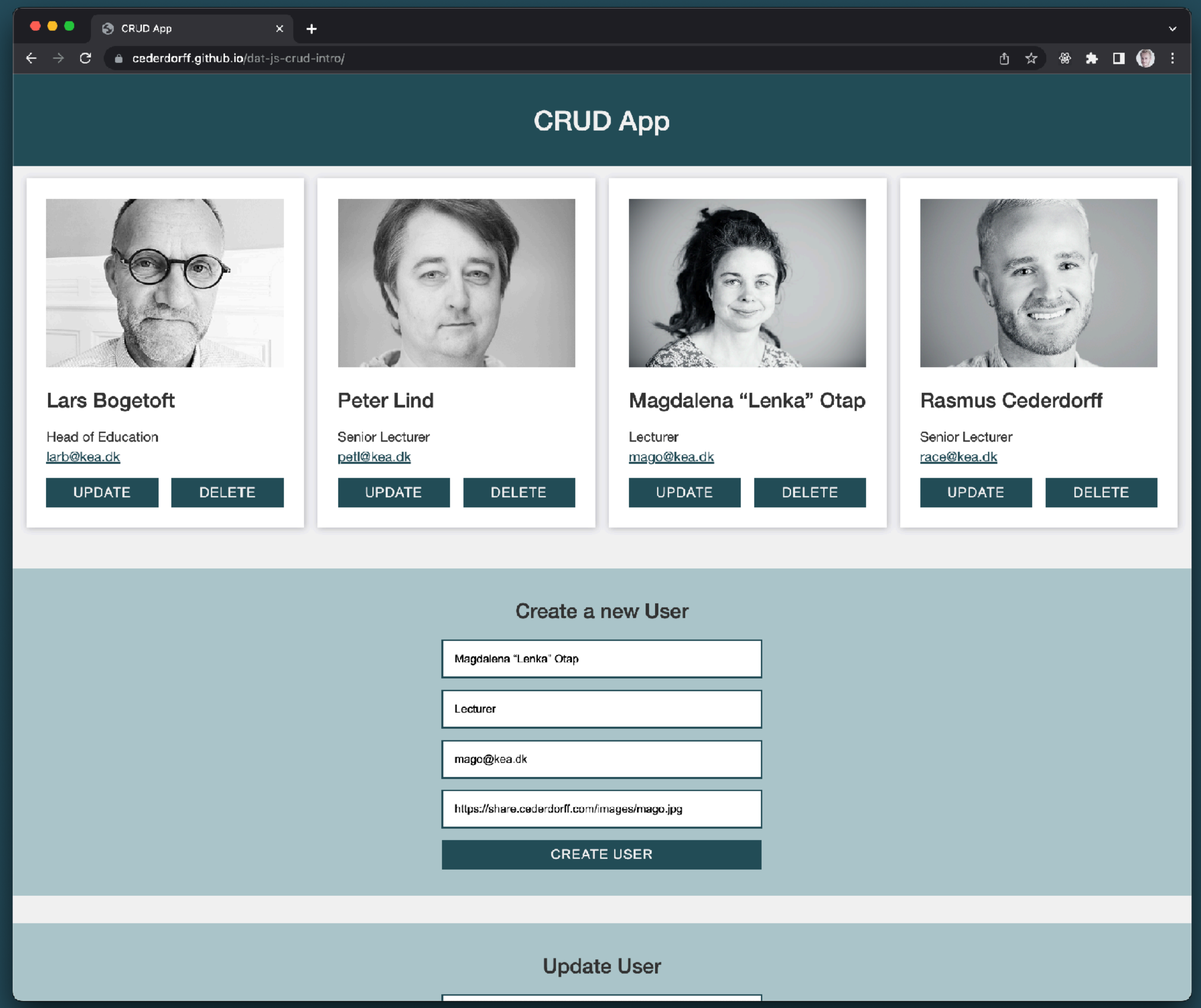
Content

- 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!

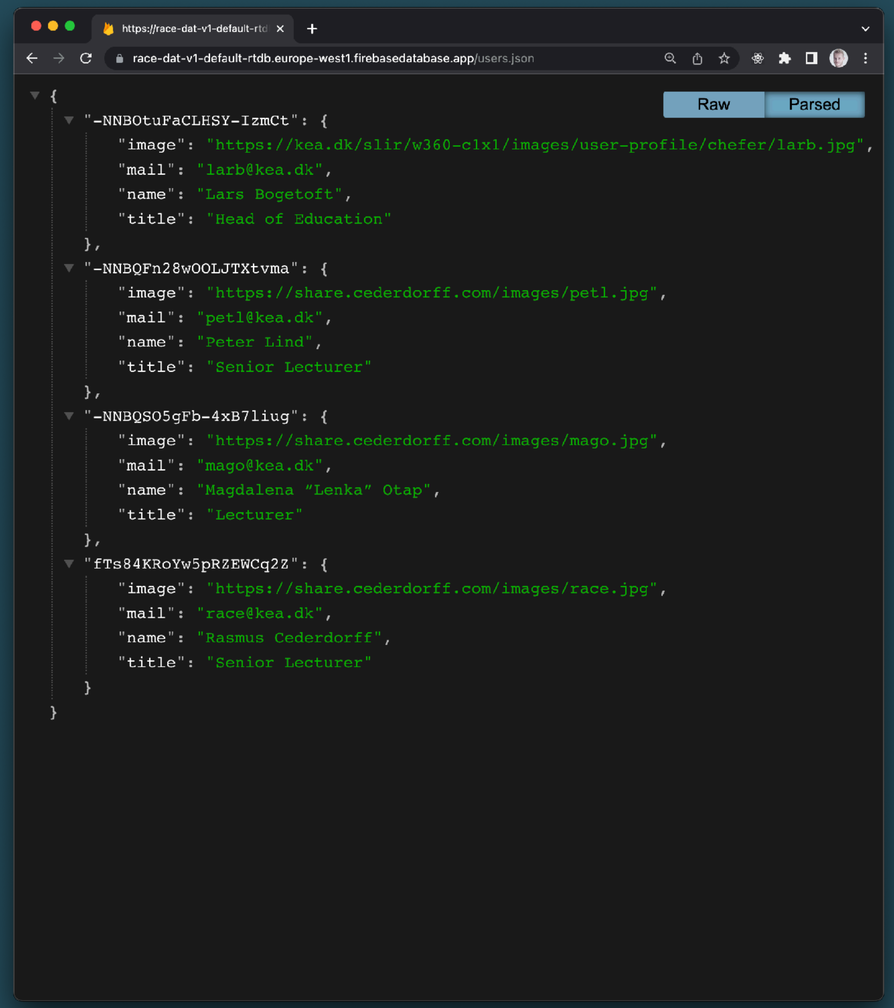
https://www.youtube.com/watch?v=0vPt7GI-2kc&ab_channel=MongoDB

Frontend (client)

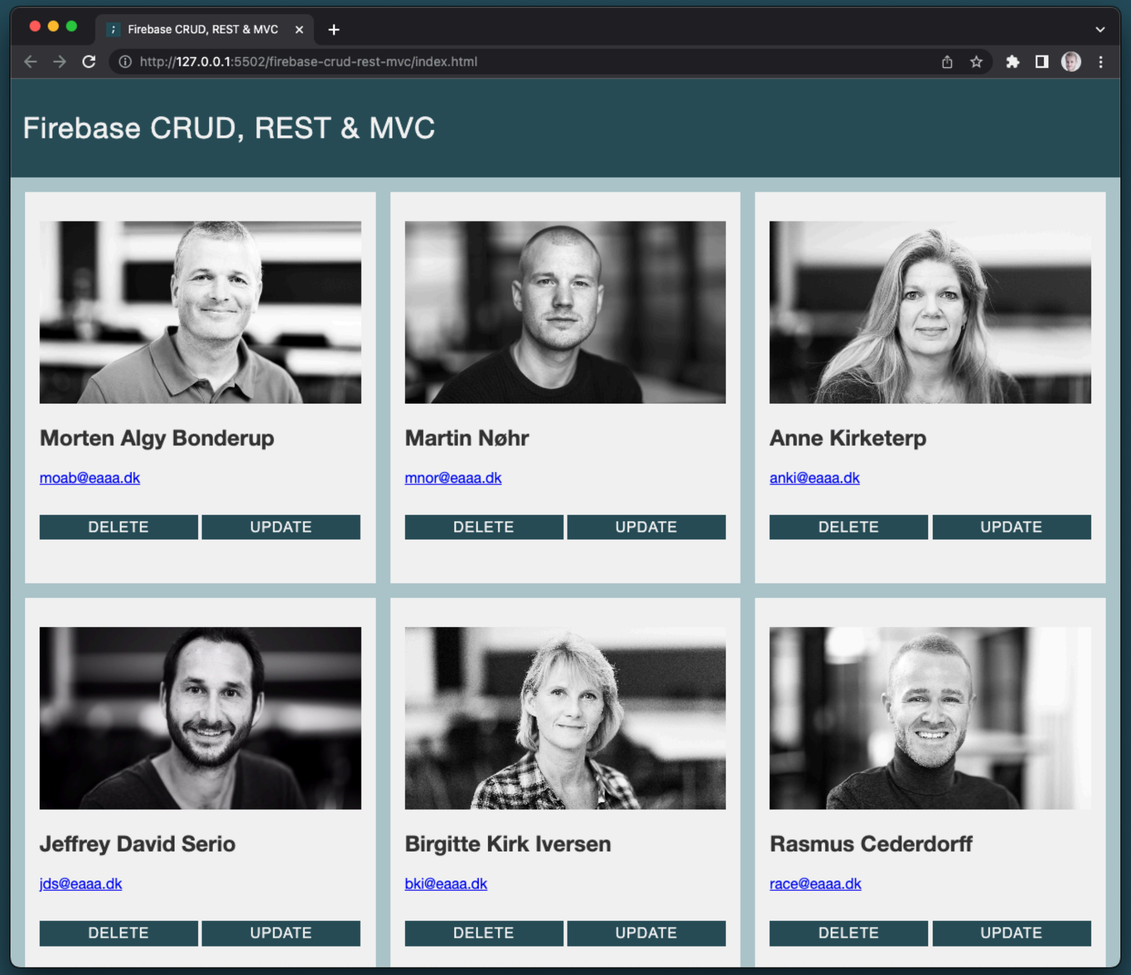


https://cederdorff.github.io/dat-js-crud-intro/

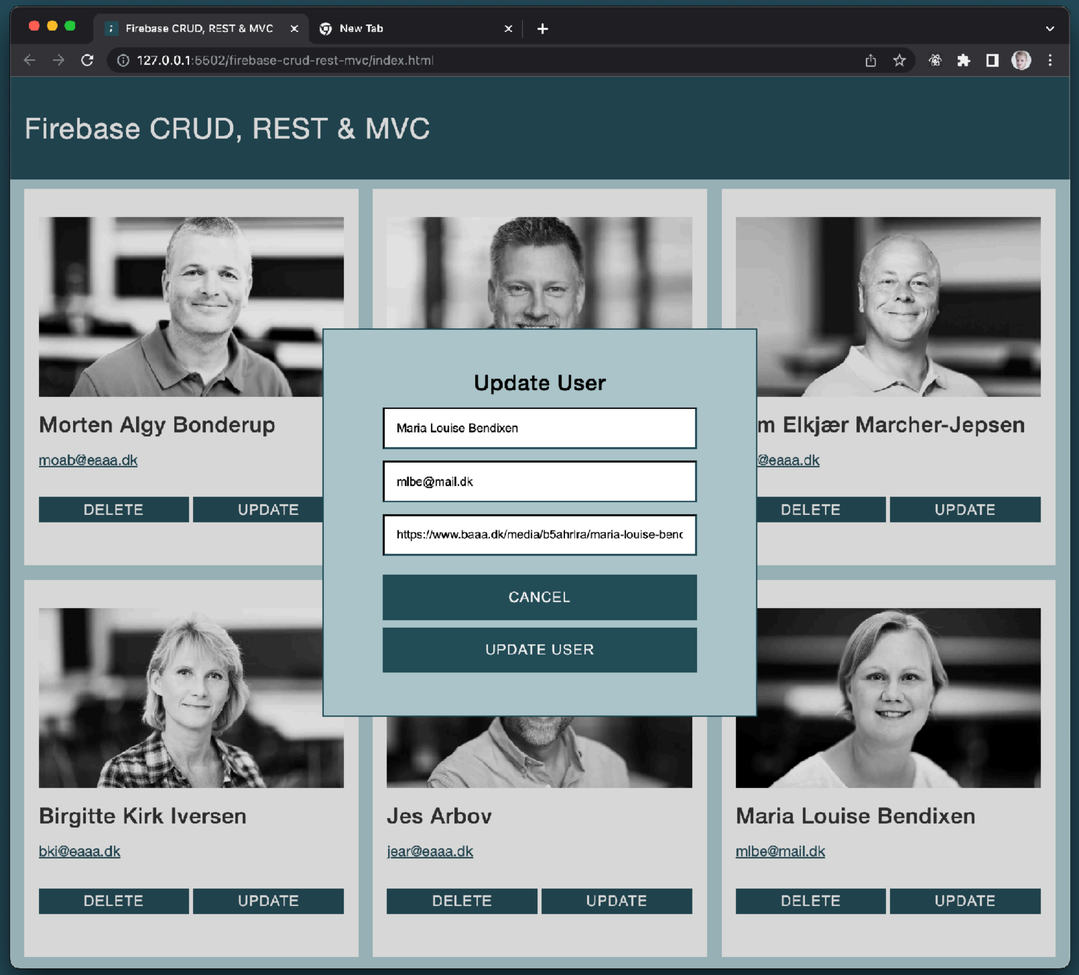
Backend (Server)



https://race-dat-v1-default-rtdb.europe-west1.firebasedatabase.app/users.json



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

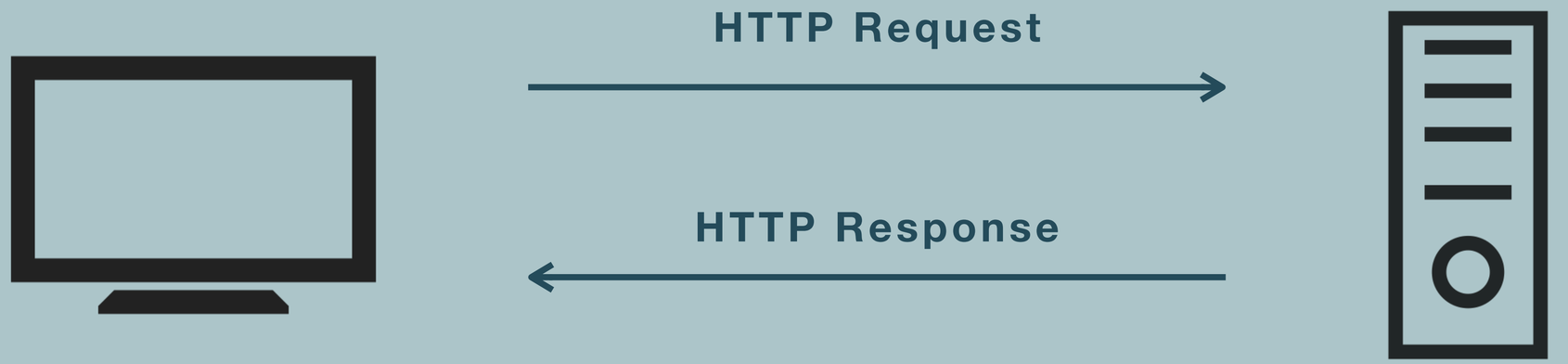
SERVER

Client-Server Model

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

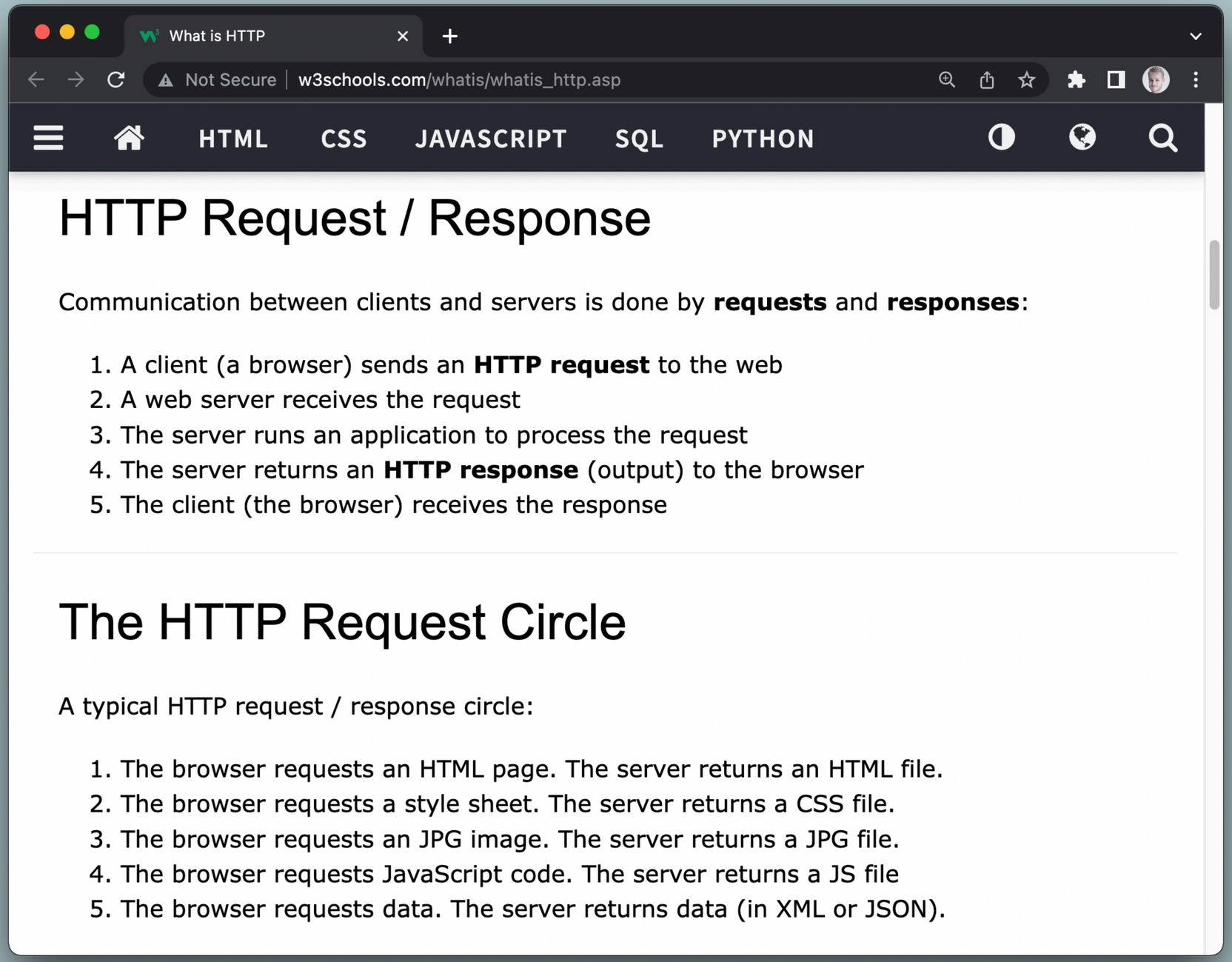
Client-Server Model

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



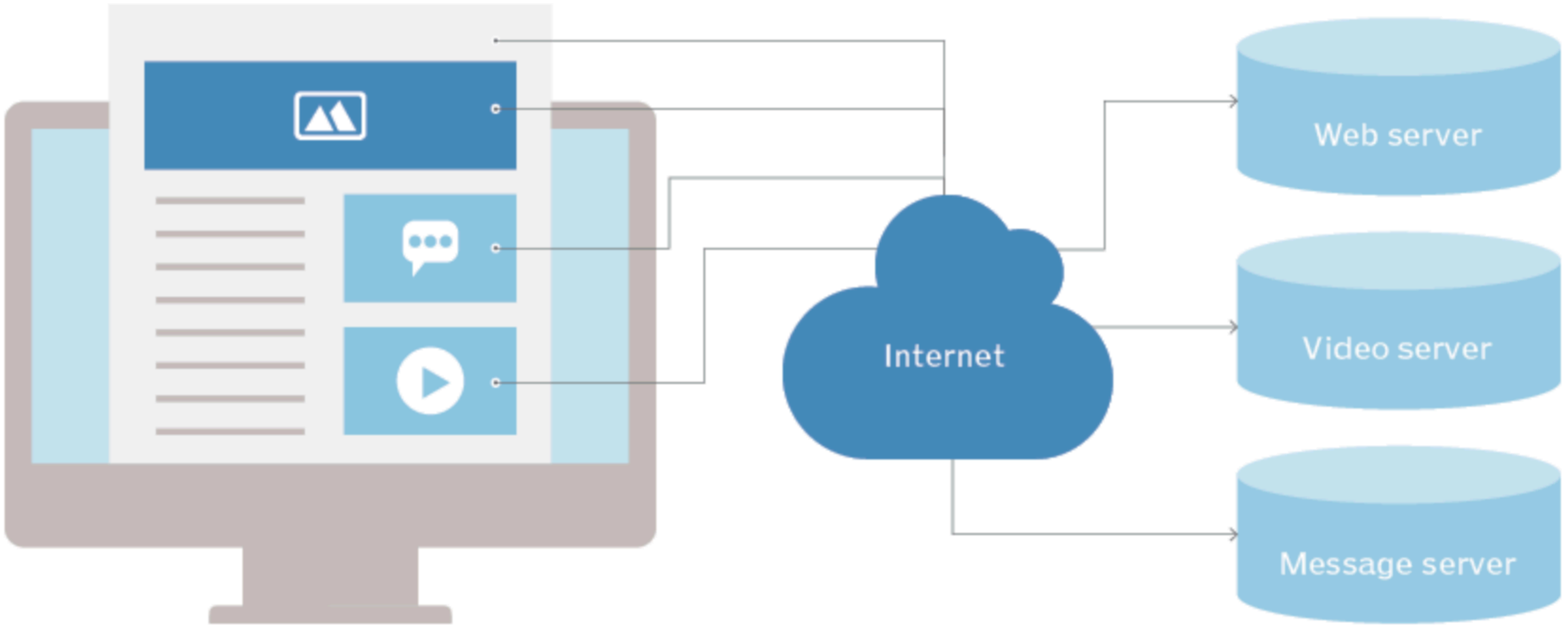
Hyper Text Transfer Protocol

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



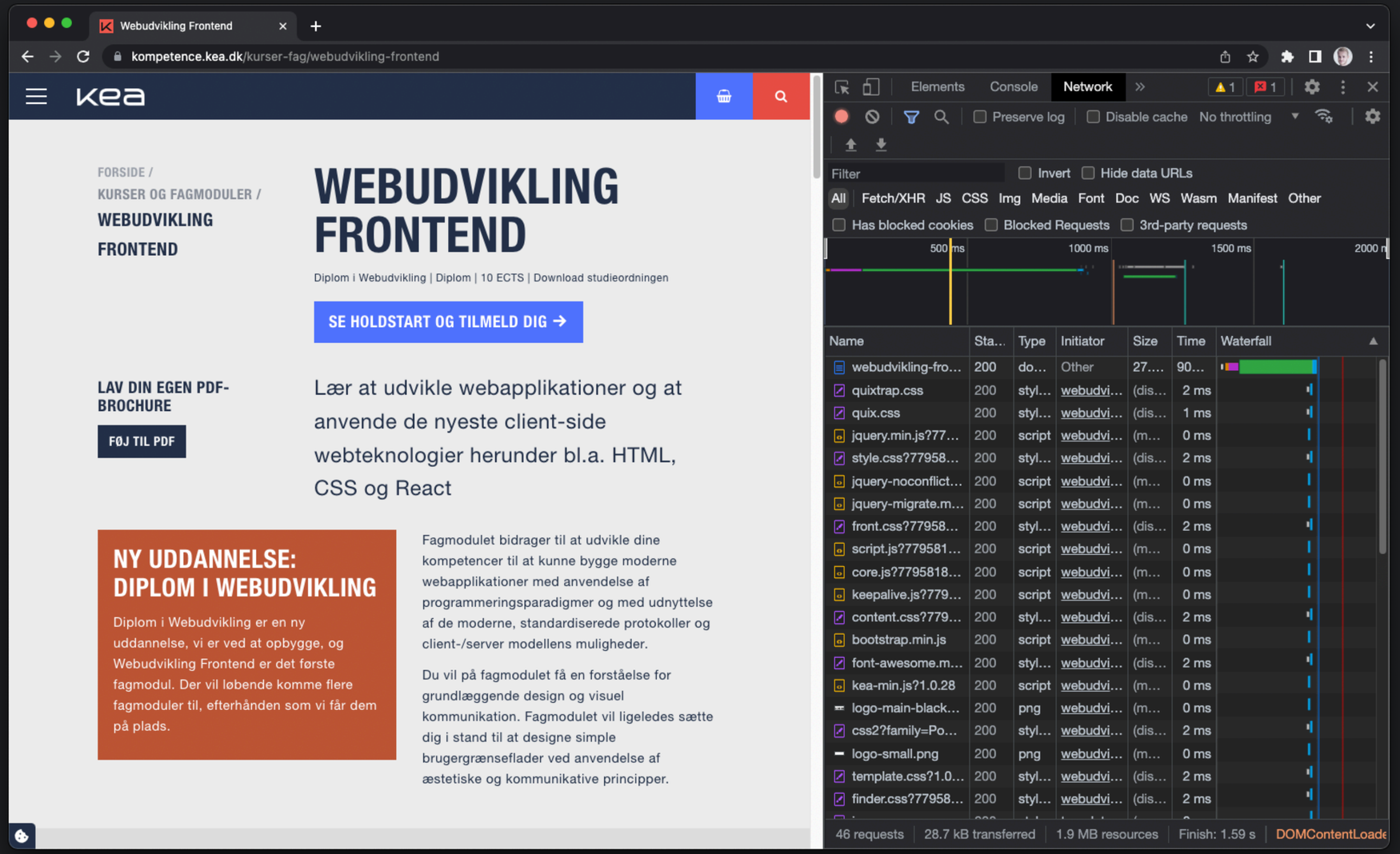
https://www.w3schools.com/whatis/whatis_http.asp

How HTTP Works



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

Network Tab


Webudvikling Frontend
kompetence.kea.dk/kurser-fag/webudvikling-frontend
kea
FORSIDE /
KURSER OG FAGMODULER /
WEBUDVIKLING
FRONTEND
WEBUDVIKLING FRONTEND
Diplom i Webudvikling | Diplom | 10 ECTS | Download studieordningen
SE HOLDSTART OG TILMELD DIG →
LAV DIN EGEN PDF-BROCHURE
FØJ TIL PDF
Lær at udvikle webapplikationer og at anvende de nyeste client-side webteknologier herunder bl.a. HTML, CSS og React
NY UDDANNELSE: DIPLOM I WEBUDVIKLING
Diplom i Webudvikling er en ny uddannelse, vi er ved at opbygge, og Webudvikling Frontend er det første fagmodul. Der vil løbende komme flere fagmoduler til, efterhånden som vi får dem på plads.
Fagmodulet bidrager til at udvikle dine kompetencer til at kunne bygge moderne webapplikationer med anvendelse af programmeringsparadigmer og med udnyttelse af de moderne, standardiserede protokoller og client-/server modellens muligheder.
Du vil på fagmodulet få en forståelse for grundlæggende design og visuel kommunikation. Fagmodulet vil ligeledes sætte dig i stand til at designe simple brugergrænseflader ved anvendelse af æstetiske og kommunikative principper.
Elements
Console
Network
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
All
Fetch/XHR
JS
CSS
Img
Media
Font
Doc
WS
Wasm
Manifest
Other
Has blocked cookies
Blocked Requests
3rd-party requests
500 ms
1000 ms
1500 ms
2000 ms
Name
Sta...
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-noconflict... 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?7795581... 200 script webudvi... (m... 0 ms
core.js?77958818... 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
46 requests
28.7 kB transferred
1.9 MB resources
Finish: 1.59 s
DOMContentLoade

Network-tabben

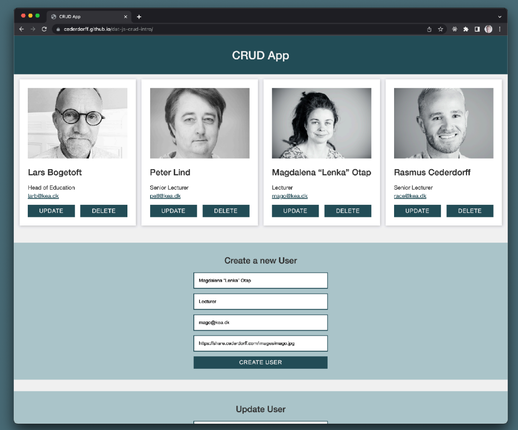
- Brug Network-tabben til at undersøge et website (fx kea.dk, dr.dk, google.dk, react-rest-and-auth.web.app eller et helt andet).
- Åben websitet i Chrome (eller en anden browser).
- Åben Developer Tool (se: How to open the dev tool in your browser) og gå til Network/Netværk.
- Genindlæs siden imens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

Web Development

WEB APP

REST API

DATA SOURCE



FETCH(GET, POST, PUT, DELETE)

REQUEST

JSON DATA

RESPONSE

users

CLIENT

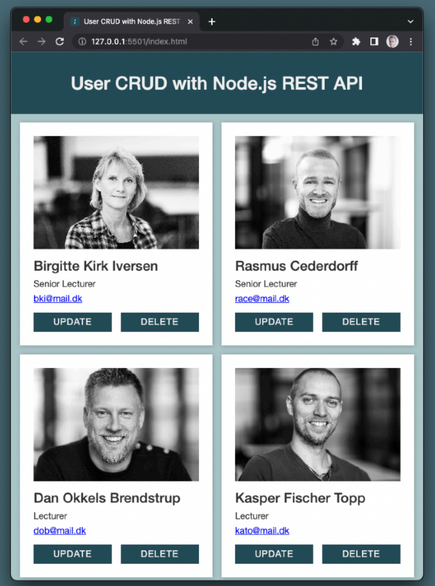
SERVER

Fetch, HTTP Request & Response

WEB APP

REST API

DATA SOURCE



FETCH(GET, POST, PUT, DELETE)

REQUEST

JSON DATA

RESPONSE

posts

users

CLIENT

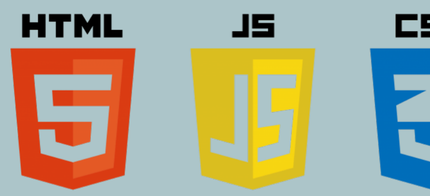
SERVER

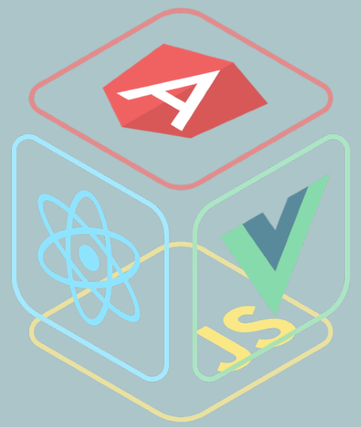
Web Development

FRONTEND

API

BACKEND





FETCH(...)

REQUEST

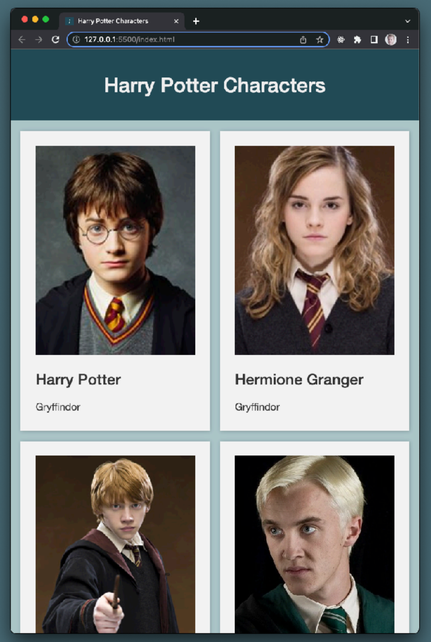
JSON DATA

RESPONSE

Fetch, HTTP Request & Response

WEB APP

DATA SOURCE



FETCH

REQUEST (GET)

JSON DATA

RESPONSE

potters

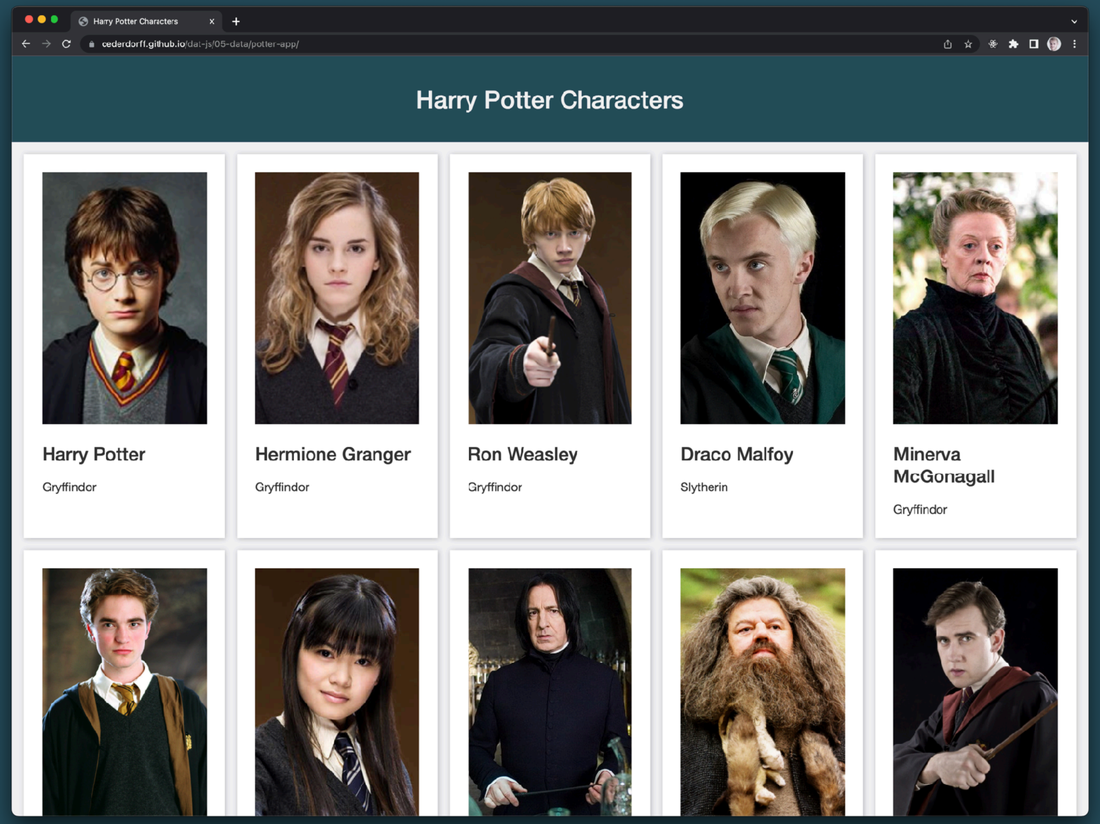
pokemons

southpark

CLIENT

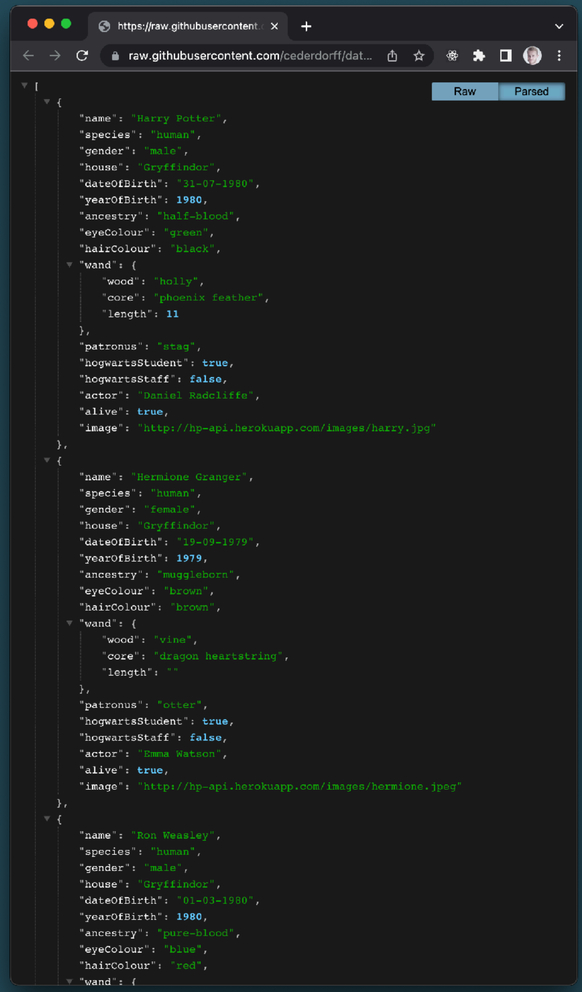
SERVER

Frontend (client)



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

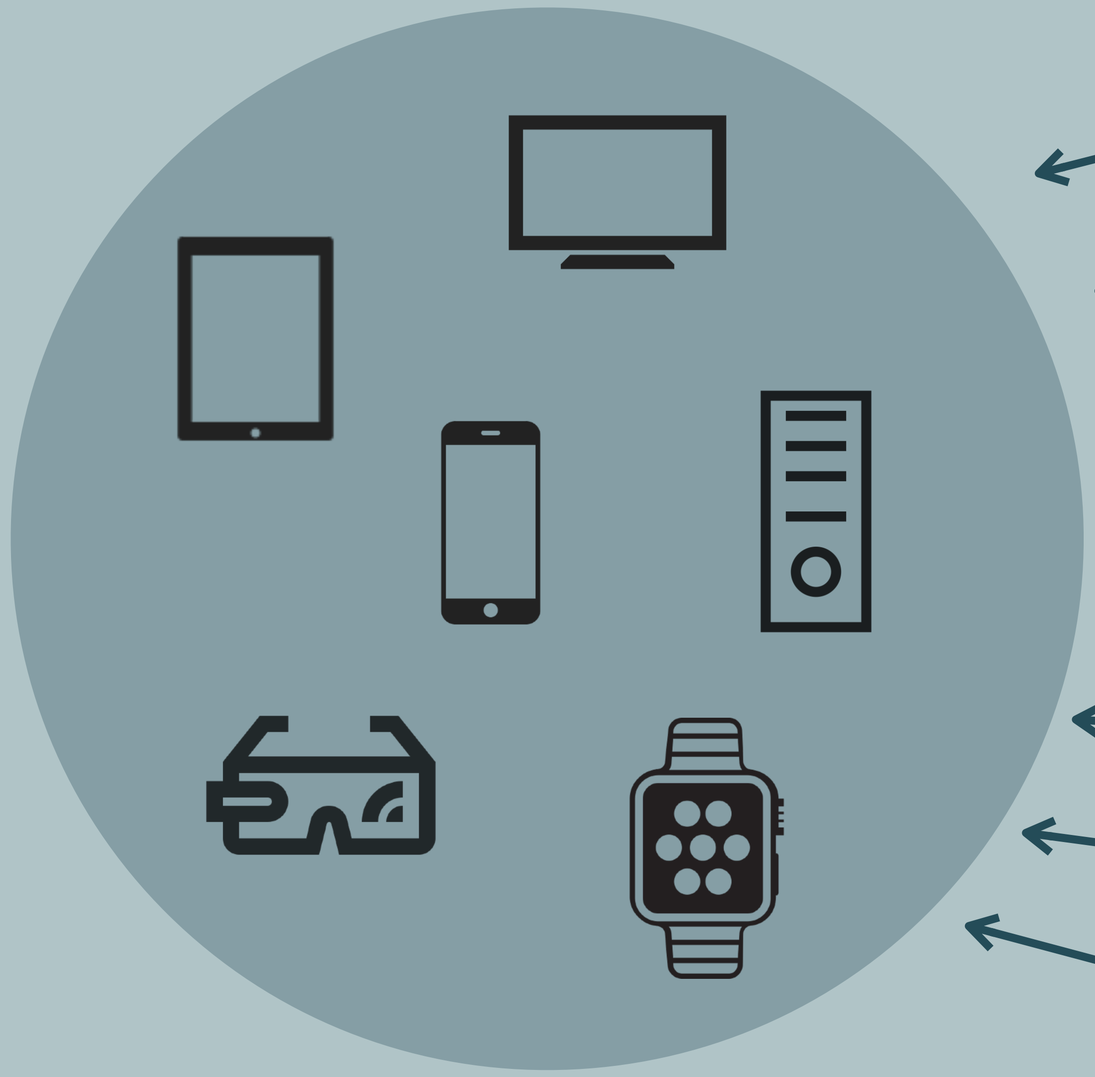
JSON Data Source (Server)



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

API

WEB APP



DATA SOURCES

API — WIKIPEDIA

API — GOOGLE MAPS

API — CVR API

API — SoMe

API — WEATHER API

API — HEADLESS CMS

API — Firebase REST

Web Development

WEB APP

API

DATA SOURCE

FETCH(...)

REQUEST

JSON DATA

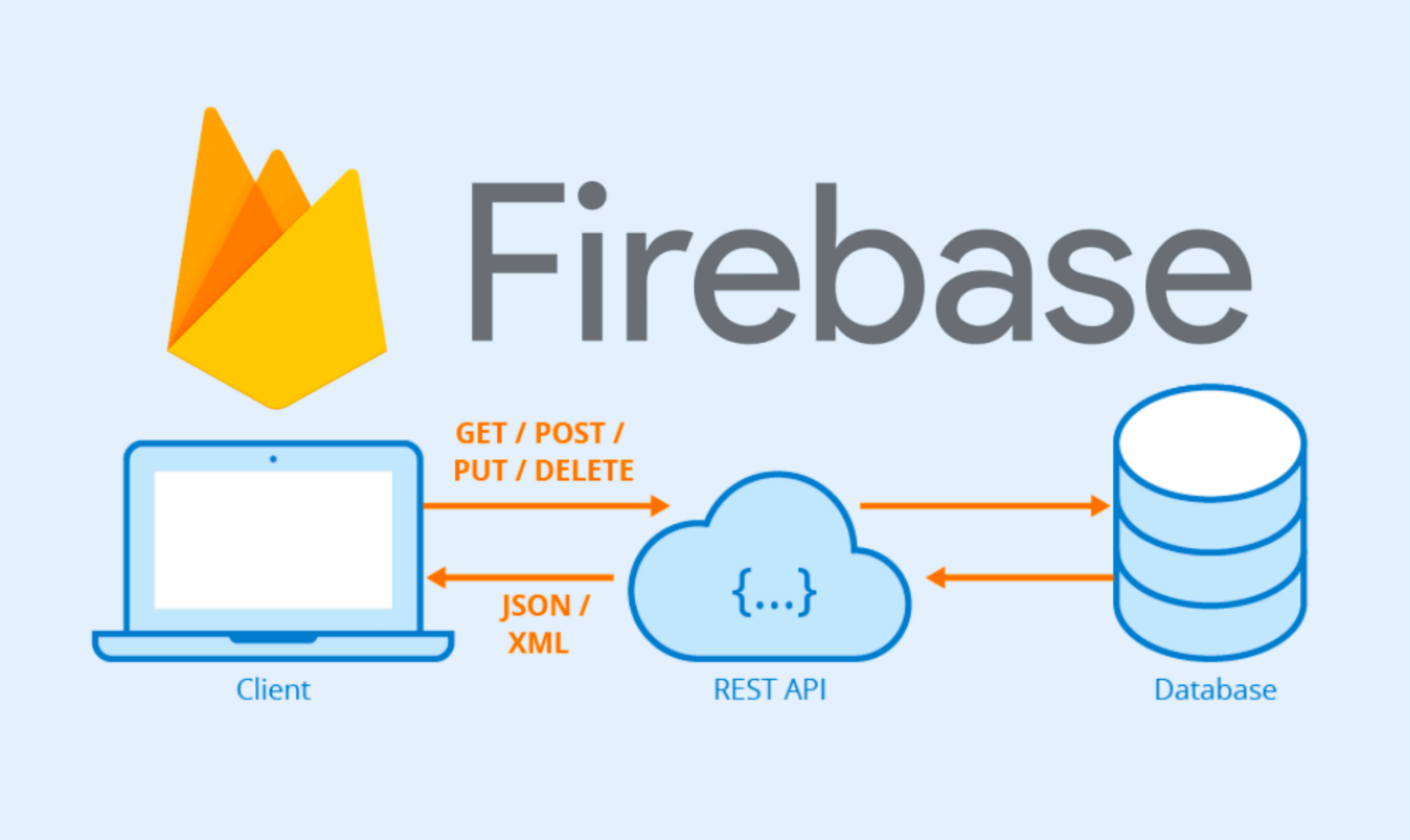
RESPONSE

JSON

CLIENT

SERVER

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



javascript.plainenglish.io/firebase-cloud-functions-tutorial-creating-a-rest-api-8cbc51479f80

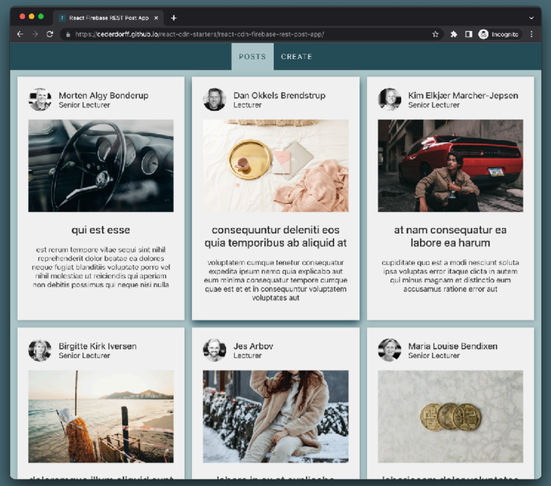

Rasmus
Firebase
made with mematic

Fetch, HTTP Request & Response

WEB APP

FIREBASE REST API

DATA SOURCE



FETCH(GET, POST, PUT, DELETE)

REQUEST

JSON DATA

RESPONSE

JSON

Firebase
Realtime Database

posts

users

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/ca
    .then(function (response) {
        return response.json();
    })
    .then(function (data) {
        console.log(data);
    });
//
//
// or with async/await
const response = await fetch("https://cederdorff.gi
const data = await 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

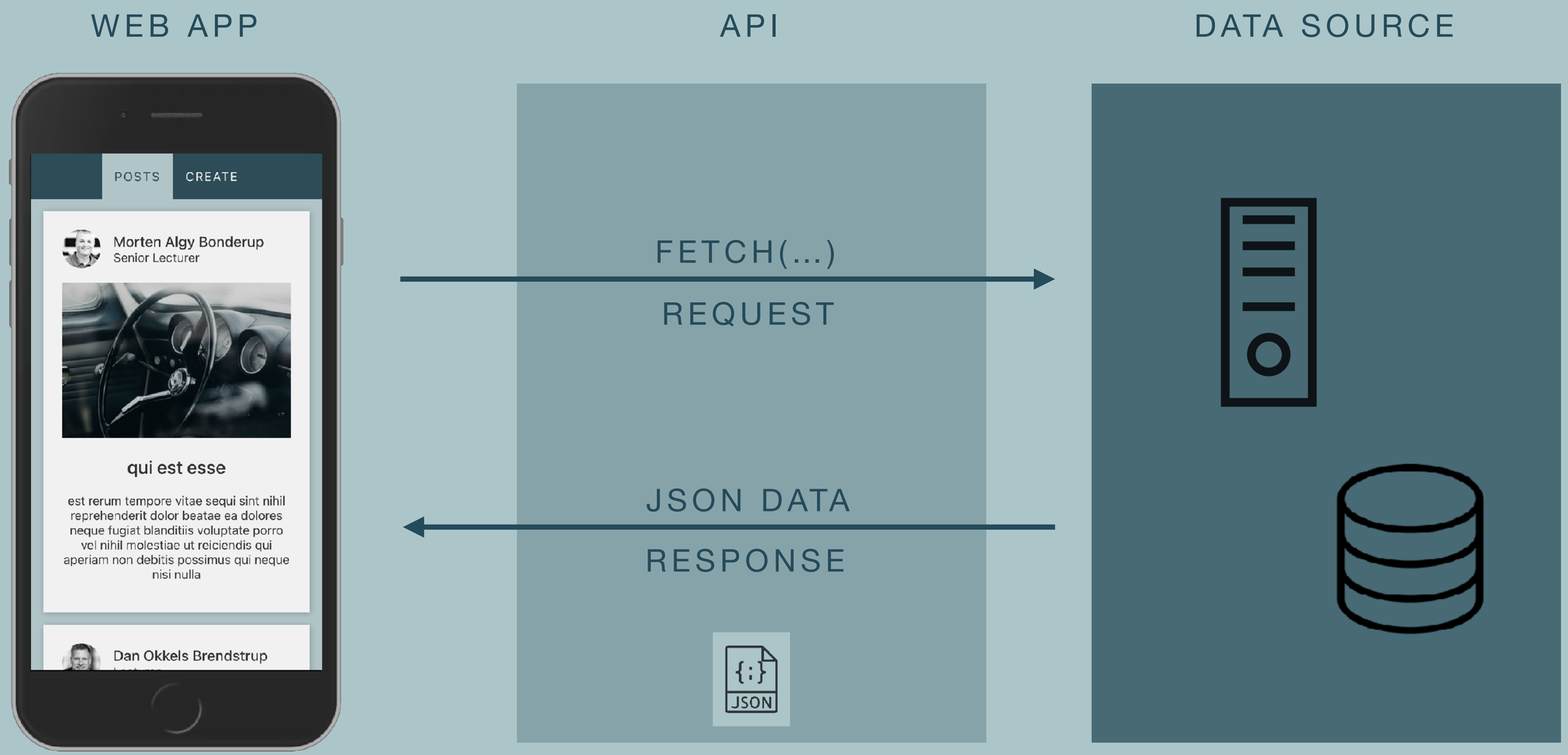
```
let promise = fetch(url, [options])
```

- `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`.

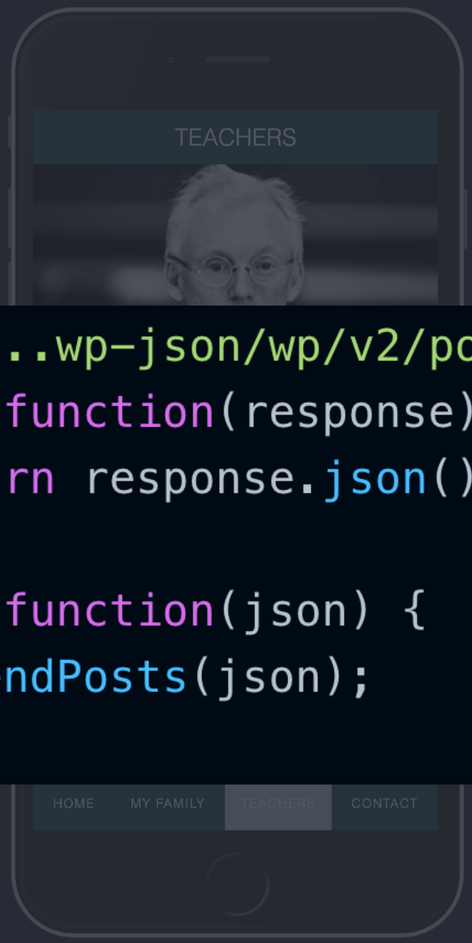
https://javascript.info/fetch

Fetch, fetch, fetch



Fetch

WEB APP




```
fetch("...wp-json/wp/v2/posts")
  .then(function(response) {
    return response.json();
  })
  .then(function(json) {
    appendPosts(json);
  });
```

REQUEST (fetch)

RESPONSE (then)

DATA SOURCE

```js
/*
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>
        <img src="${person.img}">
        <h4>${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;
}
```

```json
[
  {
    "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"
  }
]
```

fetch-family-members

```js
/*
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>
        <img src="${person.img}">
        <h4>${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)

```json
[
  {
    "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"
  }
]
```

fetch-family-members

```
/*
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>
        <img src="${person.img}">
        <h4>${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": "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-family-members

request (fetch)

response (JSON)

app.js — web-diplom-frontend

app.js

fetch-persons-grid > app.js > ...

```
let persons = []; // global variable

async function getPerson() {
    const response = await fetch(
        "https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json"
    ); // fetch request - fetch data from a given url
    persons = await response.json(); // setting global variable with fetched data
    displayPersons(persons); // calling displayPersons with persons as parameter
}

function displayPersons(listOfPersons) {
    let html = ""; // variable to store html
    //loop through all persons and create an article with content for each
    for (const person of listOfPersons) {
        html += /*html*/ `
            <article>
                <img src="${person.img}">
                <h2>${person.name}</h2>
                <p>${person.title}</p>
                <a href="mailto:${person.mail}">${person.mail}</a>
            </article>
        `; // generate and save html for every person in html variable
    }
    // set grid container content with person <article> elements
    // saved in html
    document.querySelector("#content").innerHTML = html;
}

getPerson(); // execute get persons to start the fun
```

main ⊗ 0 ⚠ 0 Ln 1, Col 1 Spaces: 4 UTF-8 LF {} JavaScript Go Live ✓ Prettier

https://raw.githubusercontent.c

raw.githubusercontent.com/cederdorff/web-diplom-frontend/mai...

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 Nøhr",
        "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/devlvvgj/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 Skjødt",
        "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",
```

request (fetch)

response (JSON)

app.js — web-diplom-frontend

app.js

fetch-persons-grid > app.js > ...

```
let persons = []; // global variable

async function getPerson() {
    const response = await fetch(
        "https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json"
    ); // fetch request - fetch data from a given url
    persons = await response.json(); // setting global variable with fetched data
    displayPersons(persons); // calling displayPersons with persons as parameter
}

function displayPersons(listOfPersons) {
    let html = ""; // variable to store html
    //loop through all persons and create an article with content for each
    for (const person of listOfPersons) {
        html += /*html*/ `
            <article>
                <img src="${person.img}">
                <h2>${person.name}</h2>
                <p>${person.title}</p>
                <a href="mailto:${person.mail}">${person.mail}</a>
            </article>
        `; // generate and save html for every person in html variable
    }
    // set grid container content with person <article> elements
    // saved in html
    document.querySelector("#content").innerHTML = html;
}

getPerson(); // execute get persons to start the fun
```

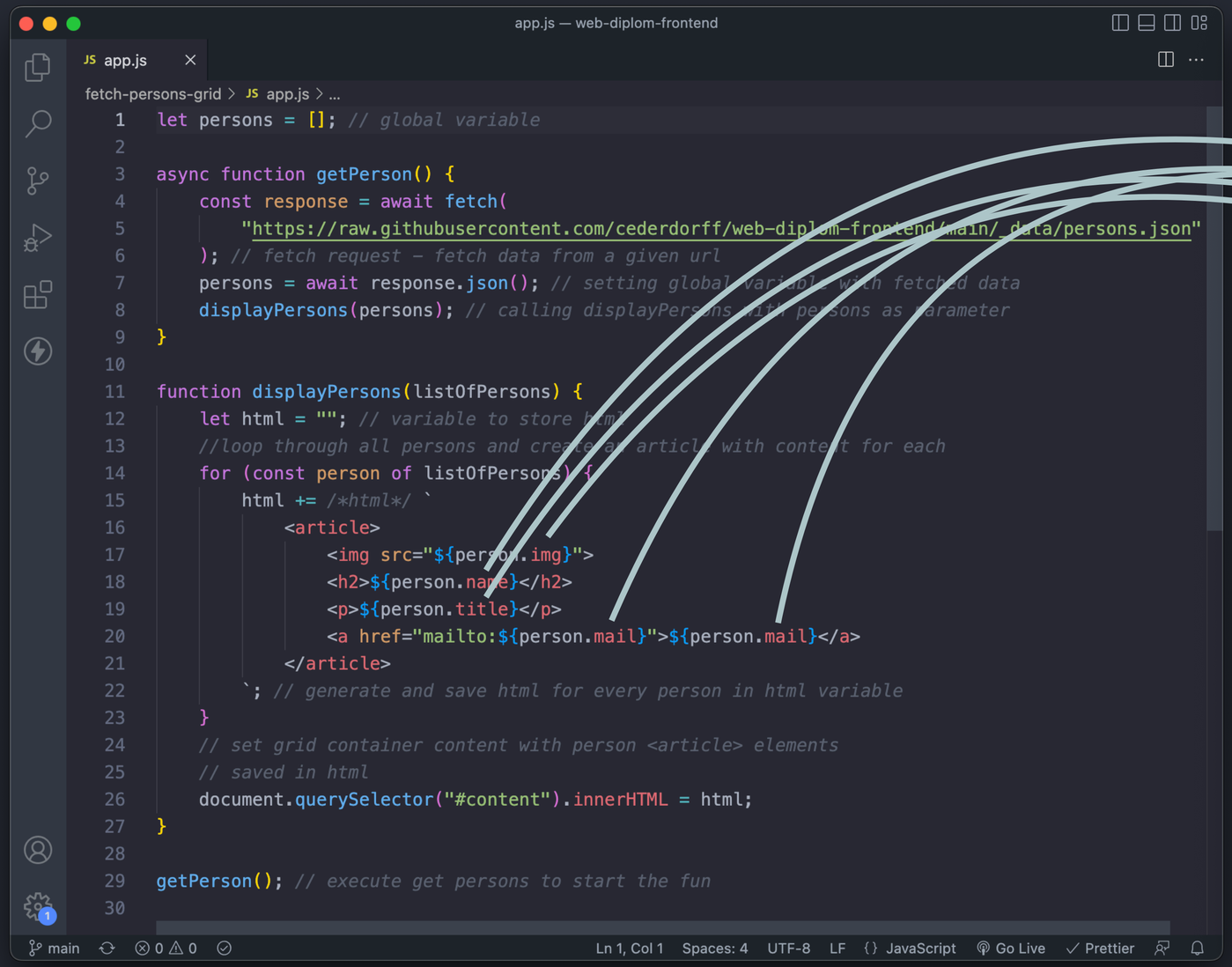
main ⊗ 0 ⚠ 0 Ln 1, Col 1 Spaces: 4 UTF-8 LF {} JavaScript Go Live ✓ Prettier

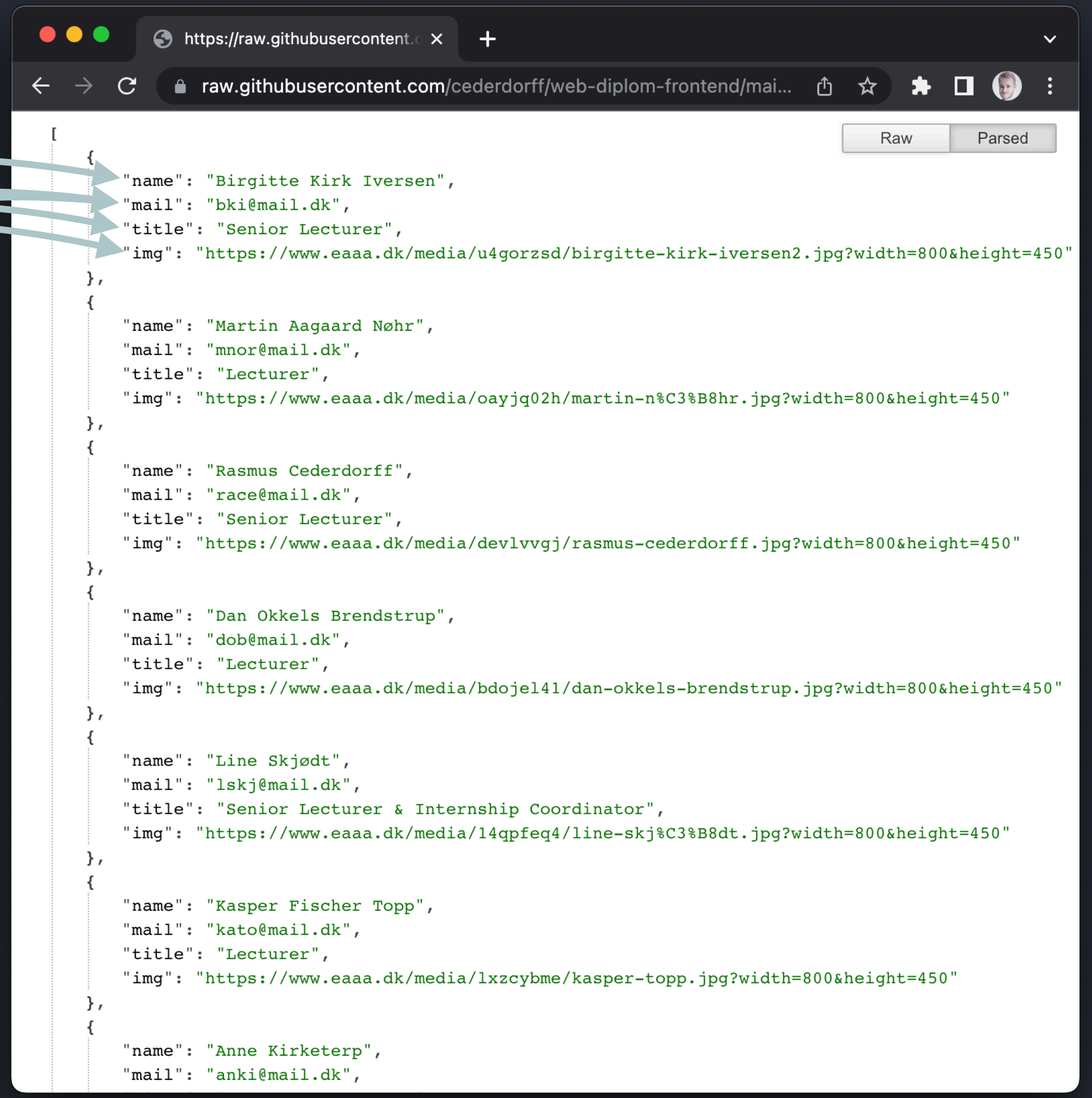
https://raw.githubusercontent.c

raw.githubusercontent.com/cederdorff/web-diplom-frontend/mai...

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 Nøhr",
        "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/devlvvgj/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 Skjødt",
        "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",
```





fetch-persons-grid

index.html — web-diplom-frontend

app.js

```
fetch-persons-grid > app.js > ...
```

```js
let persons = []; // global variable

async function getPerson() {
    const response = await fetch(
        "https://raw.githubusercontent.com/cederdorff/web-diplom-frontend/main/_data/persons.json"
    ); // fetch request - fetch data from a given url
    persons = await response.json(); // setting global variable with fetched data
    displayPersons(persons); // calling displayPersons with persons as parameter
}

function displayPersons(listOfPersons) {
    let html = ""; // variable to store html
    //loop through all persons and create an article with content for each
    for (const person of listOfPersons) {
        html += /*html*/ `
            <article>
                <img src="${person.img}">
                <h2>${person.name}</h2>
                <p>${person.title}</p>
                <a href="mailto:${person.mail}">${person.mail}</a>
            </article>
        `; // generate and save html for every person in html variable
    }
    // set grid container content with person <article> elements
    // saved in html
    document.querySelector("#content").innerHTML = html;
}

getPerson(); // execute get persons to start the fun
```

index.html

```
fetch-persons-grid > index.html > html > body > main > section#content.grid-container
```

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="app.css">
    <title>Fetch Persons</title>
</head>

<body>
    <header>
        <h1>Fetch Persons</h1>
    </header>
    <main>
        <section id="content" class="grid-container"></section>
    </main>
    <script src="app.js"></script>
</body>

</html>
```

DOM Manipulation

main ⊗ 0 ⚠ 0 — 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 javascipt 😄

//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/canva
    .then(function (response) {
        return response.json();
    })
    .then(function (data) {
        console.log(data);
    });
```

Fetch is Asynchronous

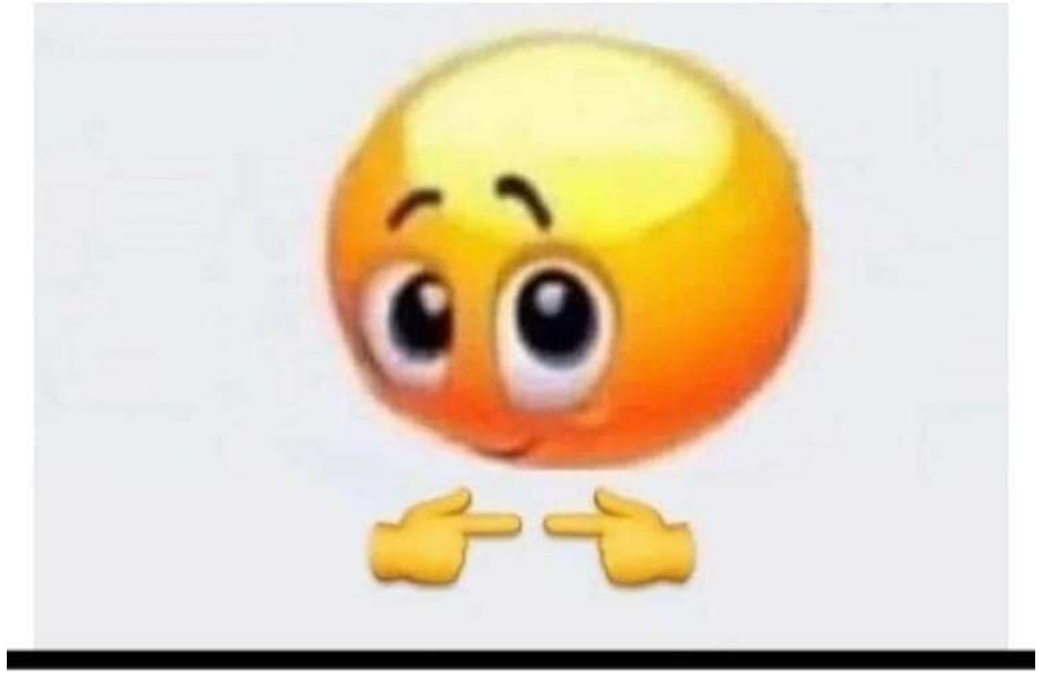
And it’s about making HTTP requests in JavaScript.
... and a way to get & post data from and to a data source.

```
// 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.githubuserconten
    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

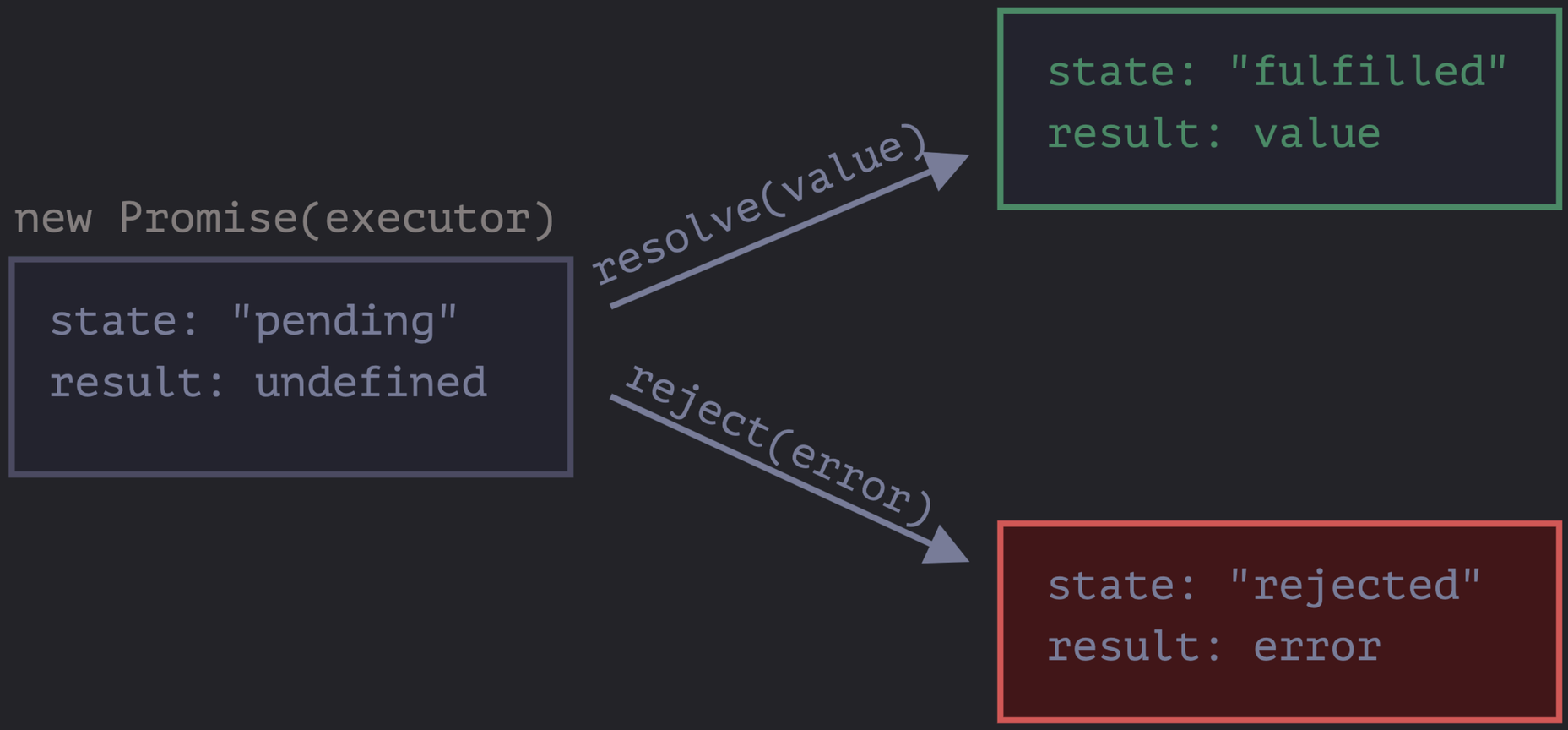
`await` makes a function wait for a Promise

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

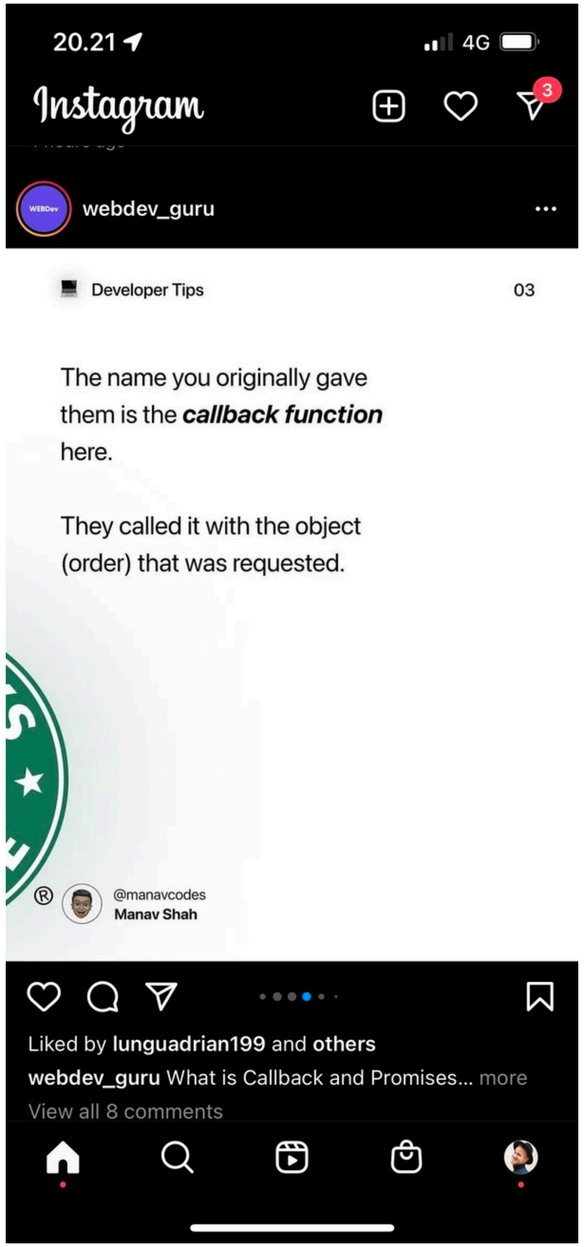
Fetch returns a promise

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

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



https://javascript.info/promise-basics



https://www.instagram.com/p/CVQGdNIjWMx/

```
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
  }
);
```

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

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.

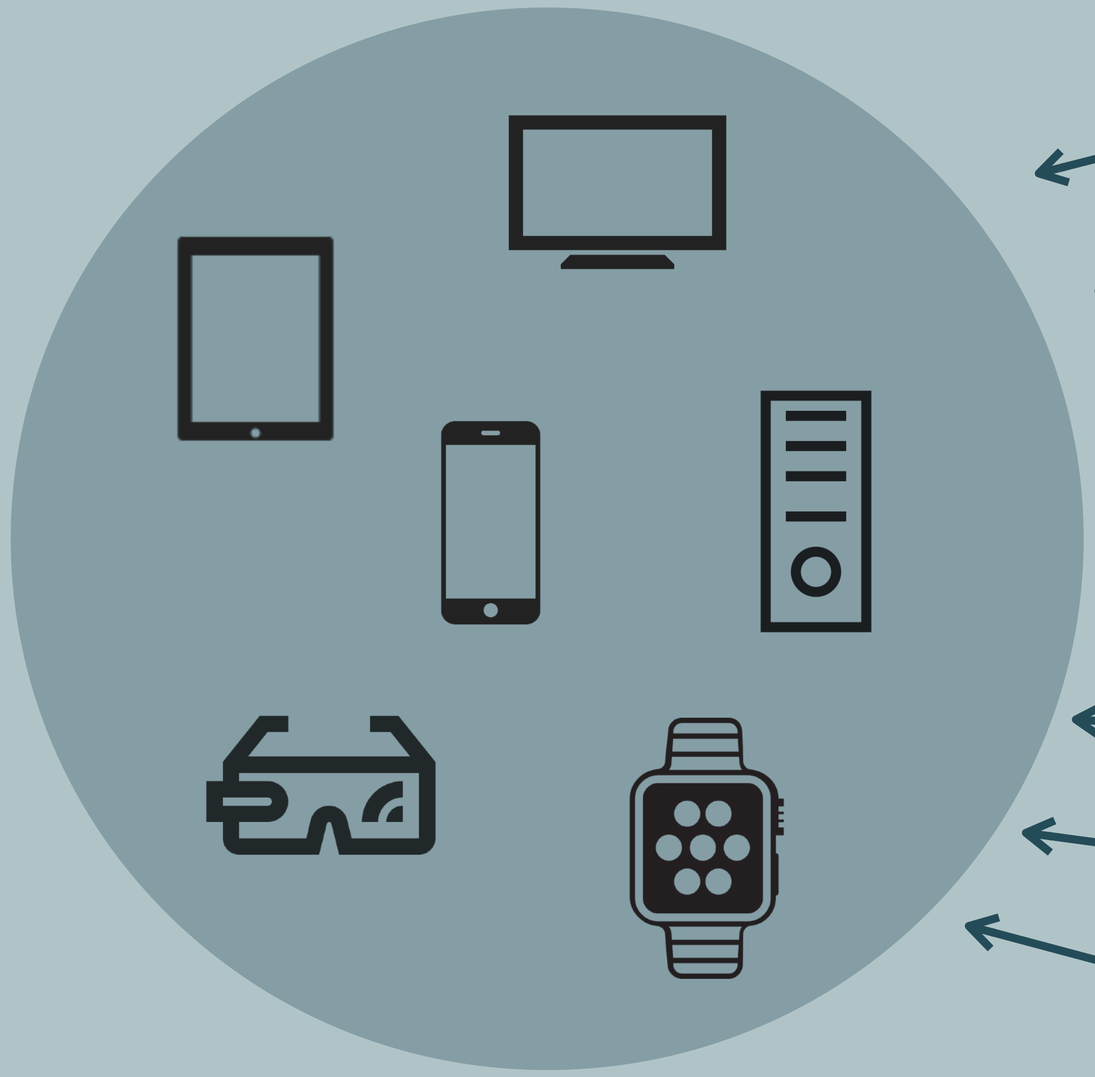
```
async function getPosts() {
    const url = "https://raw.githubusercontent.co
    const response = await fetch(url);
    const data = await response.json();
    setPosts(data);
}
getPosts();
```

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

API

WEB APP



DATA SOURCES

API — WIKIPEDIA

API — GOOGLE MAPS

API — CVR API

API — SoMe

API — WEATHER API

API — HEADLESS CMS

API — Firebase REST

API

Application **P**rogramming **I**nterface

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

What’s an API?

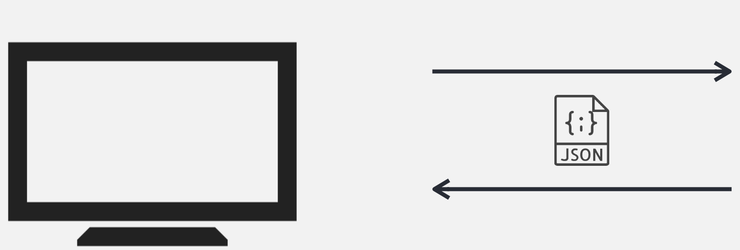
https://www.youtube.com/watch?v=s7wmiS2mSXY

API

APPLICATION
PROGRAMMING **I**NTERFACE

Application Programming Interface

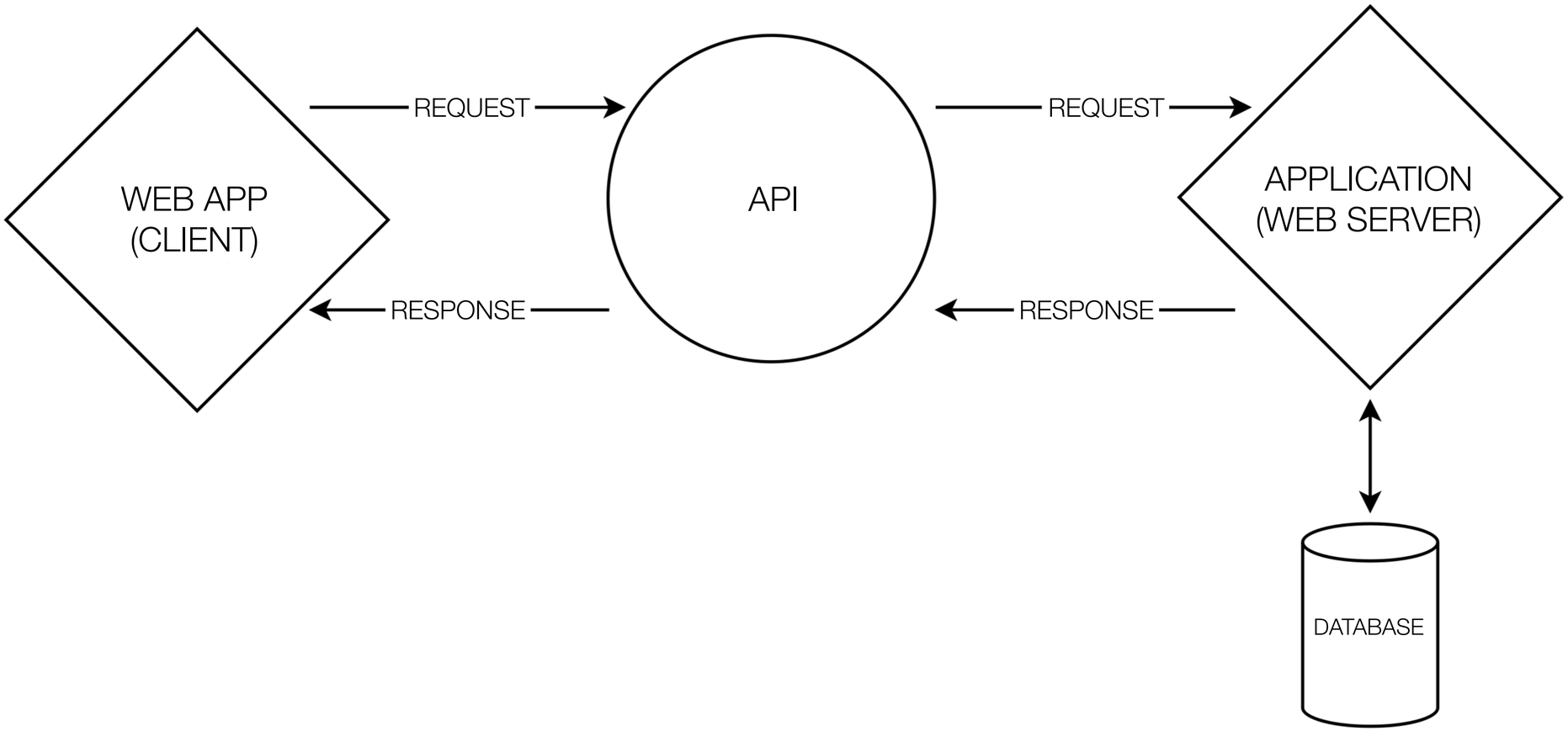
- An interface that makes it possible for two systems to communicate and exchange data in a standardised way.
- Ex communication between a web app and a web server.
- An API is the glue or bridge between different applications, platforms, or services.
- Platform independent: Can be used by different 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)

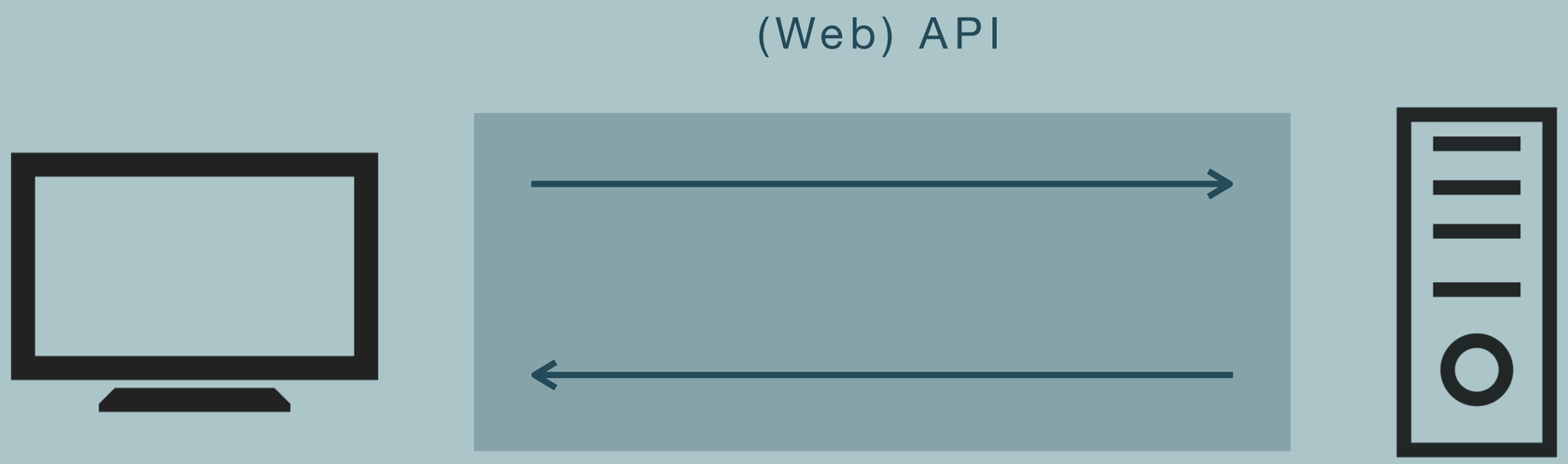
APPLICATION/ SERVICE
(WEB SERVER)

Application Programming Interface



Application Programming Interface

Communication between two systems

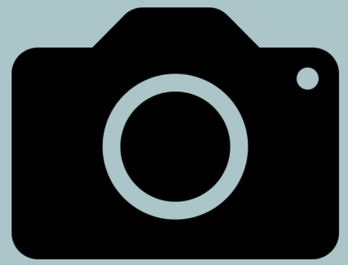


Application Programming Interface

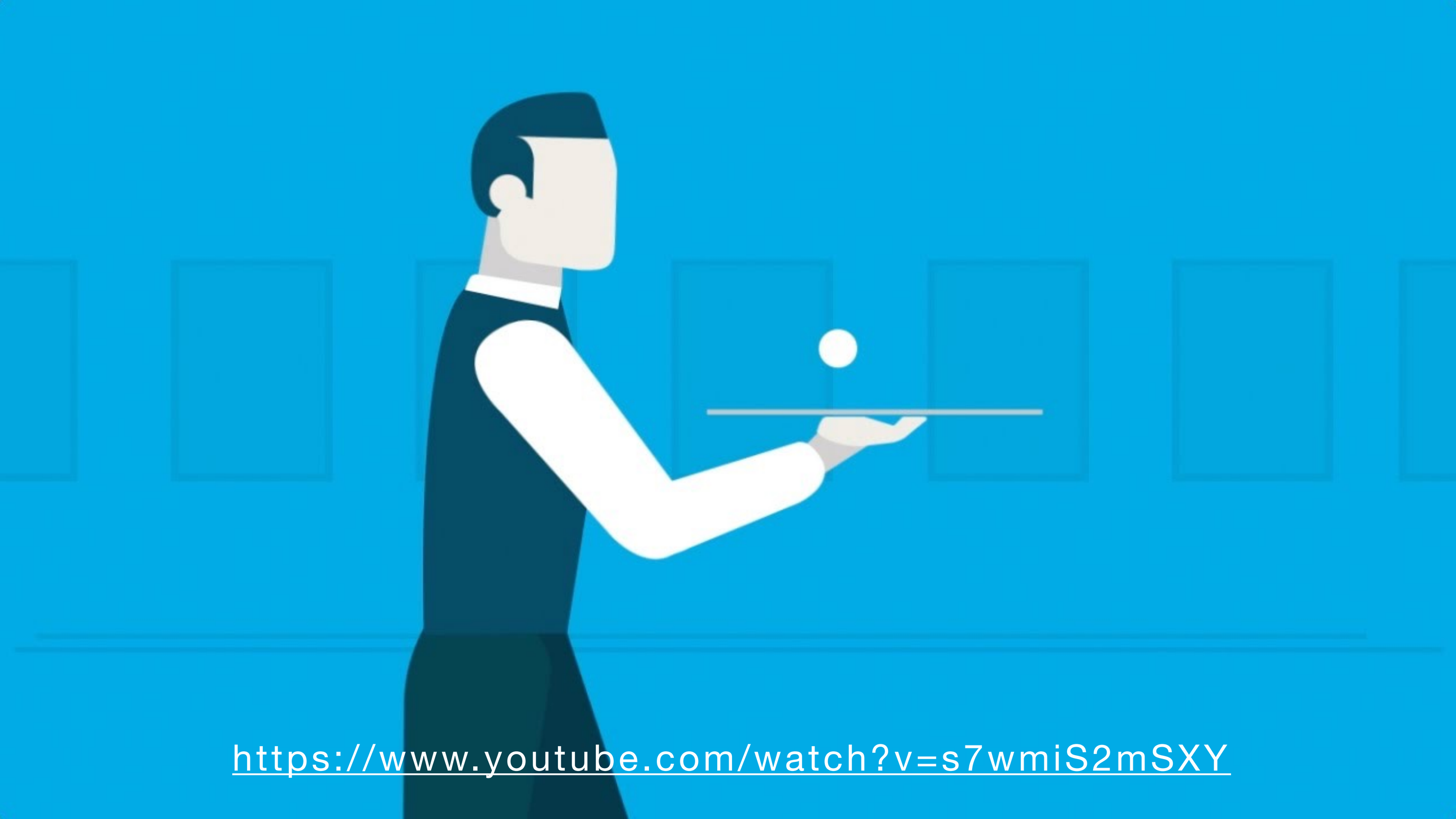
Communication between two systems

Web App

Camera API



Camera



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

CLIENT
(WEB APP)
API
DATA SOURCE
RECEIVER
WAITER
(MESSENGER)
KITCHEN
(SOURCE)

JSON

Java**S**cript **O**bject **N**otation

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

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

JSON (& API) is the glue

WEB APP

API

DATA SOURCE

FETCH(...)

REQUEST

JSON DATA

RESPONSE

JSON

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

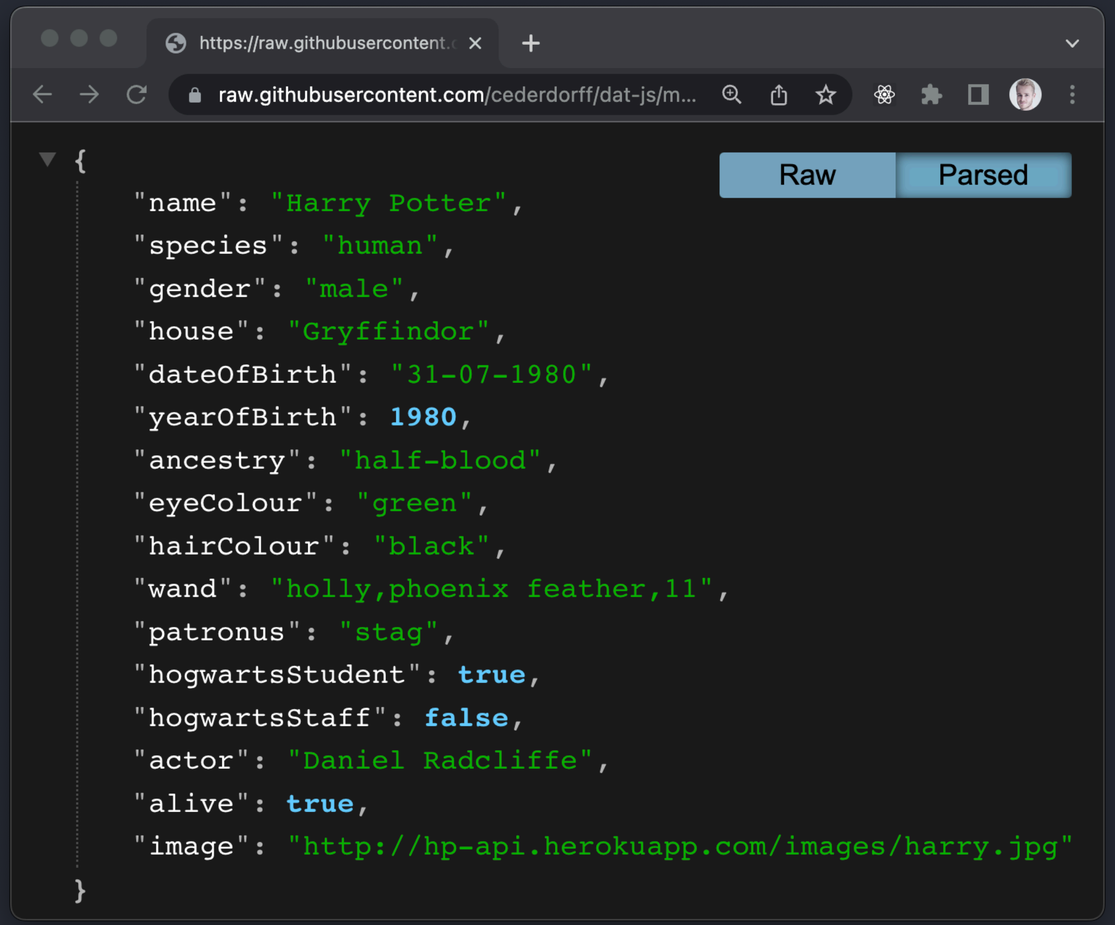
```
{
  "name": "Alicia",
  "age": 6
}
```

JSON OBJECT

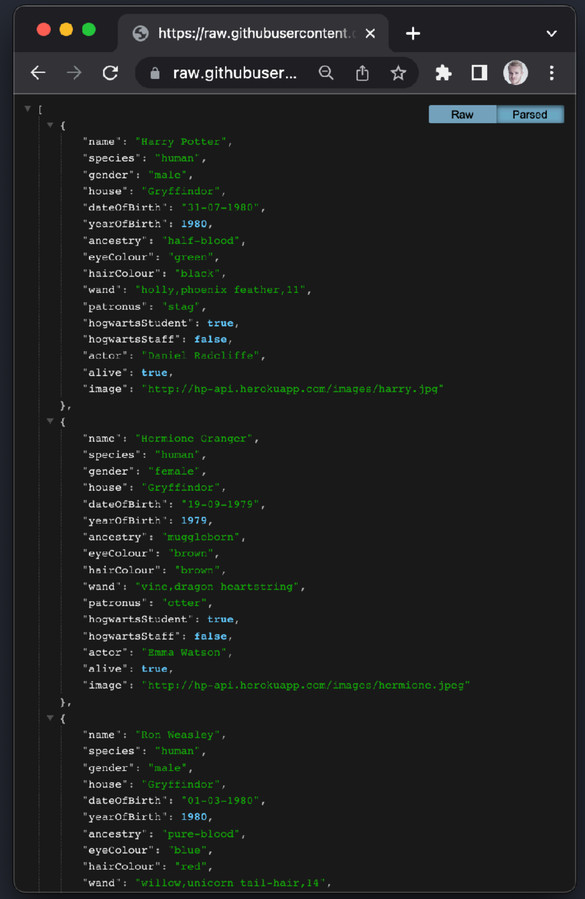
```
[{
  "name": "Alicia",
  "age": 6
}, {
  "name": "Peter",
  "age": 22
}]
```

LIST OF JSON OBJECTS

JSON

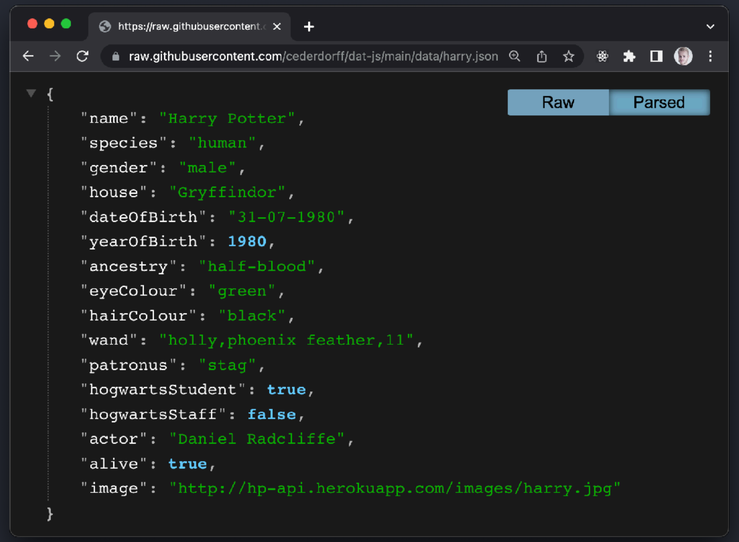


JSON OBJECT

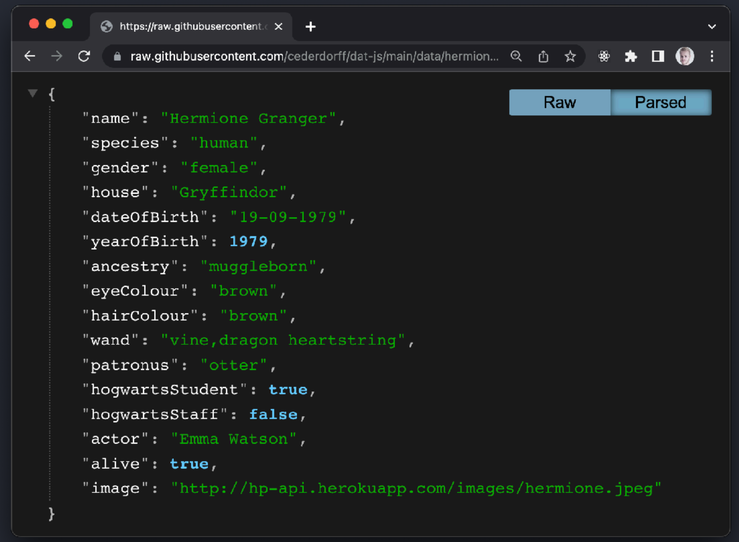


LIST OF JSON OBJECTS

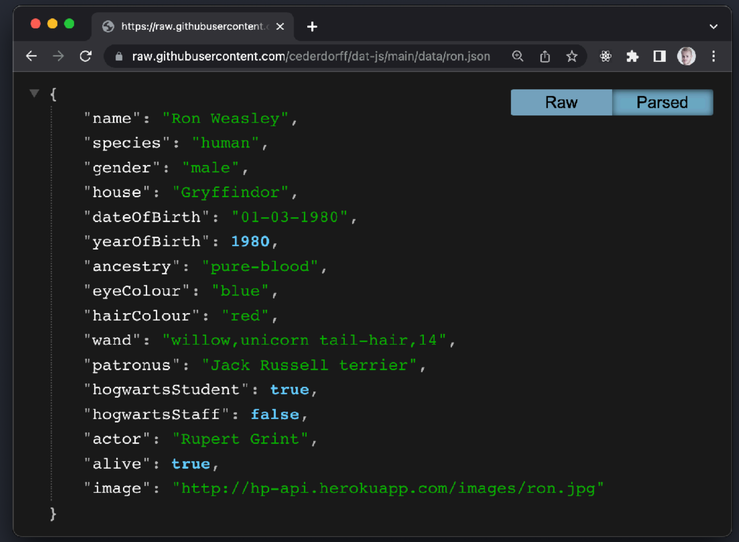
JSON Objects



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

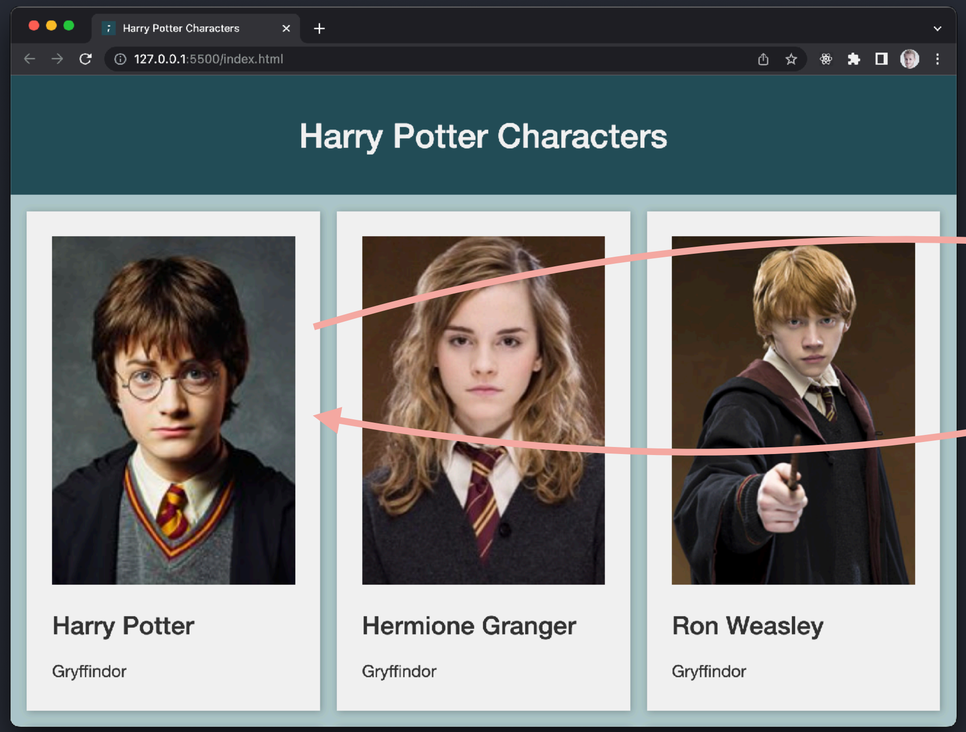


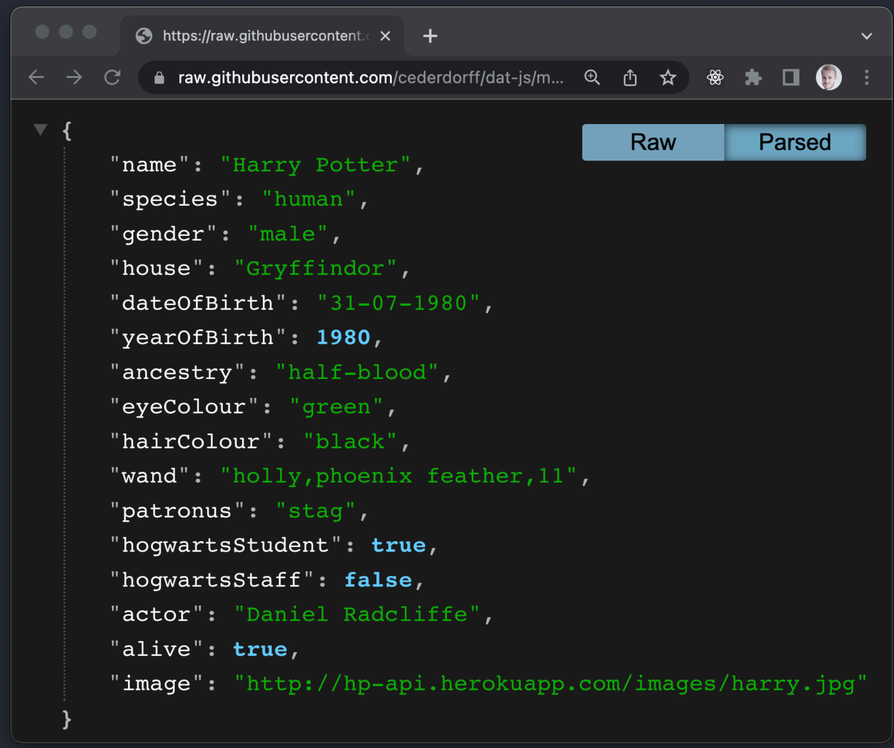
https://raw.githubusercontent.com/cederdorff/dat-js/main/data/hermione.json



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

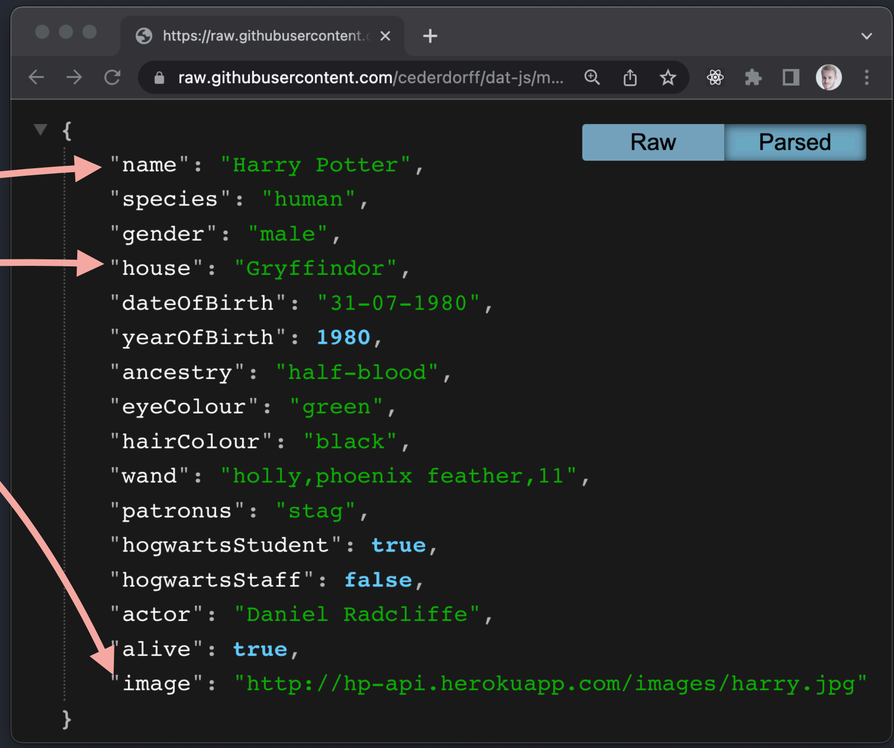
JSON Object





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

JSON Object



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",
```

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.

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

```
// 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);
```

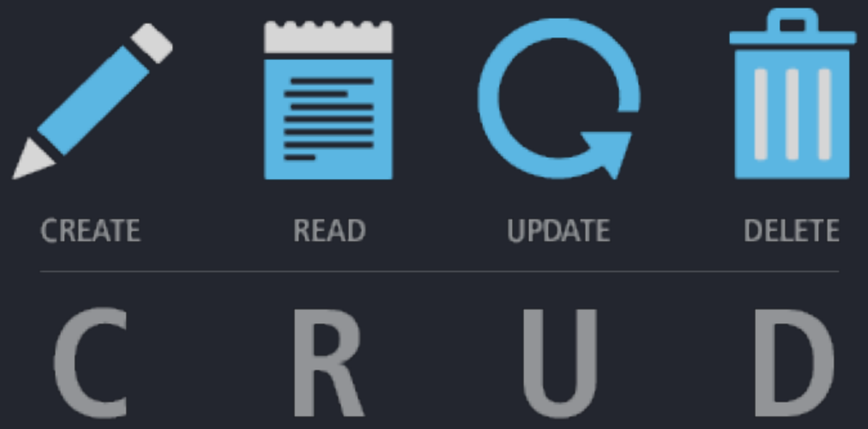
https://www.w3schools.com/js/js_json_stringify.asp

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



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.

codecademy.com/article/what-is-crud

GET

POST

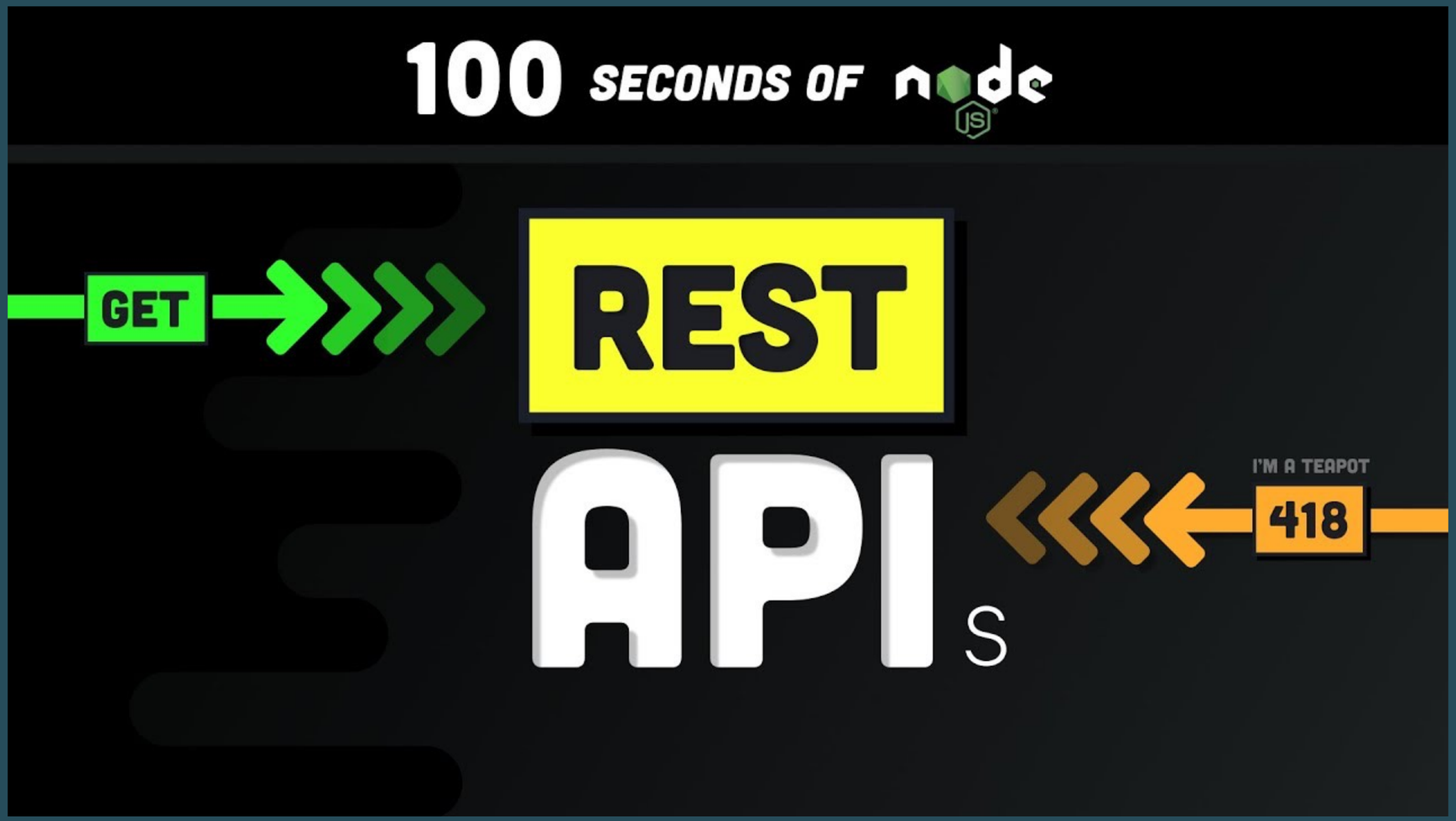
PUT

DELETE

What’s REST?

- 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.

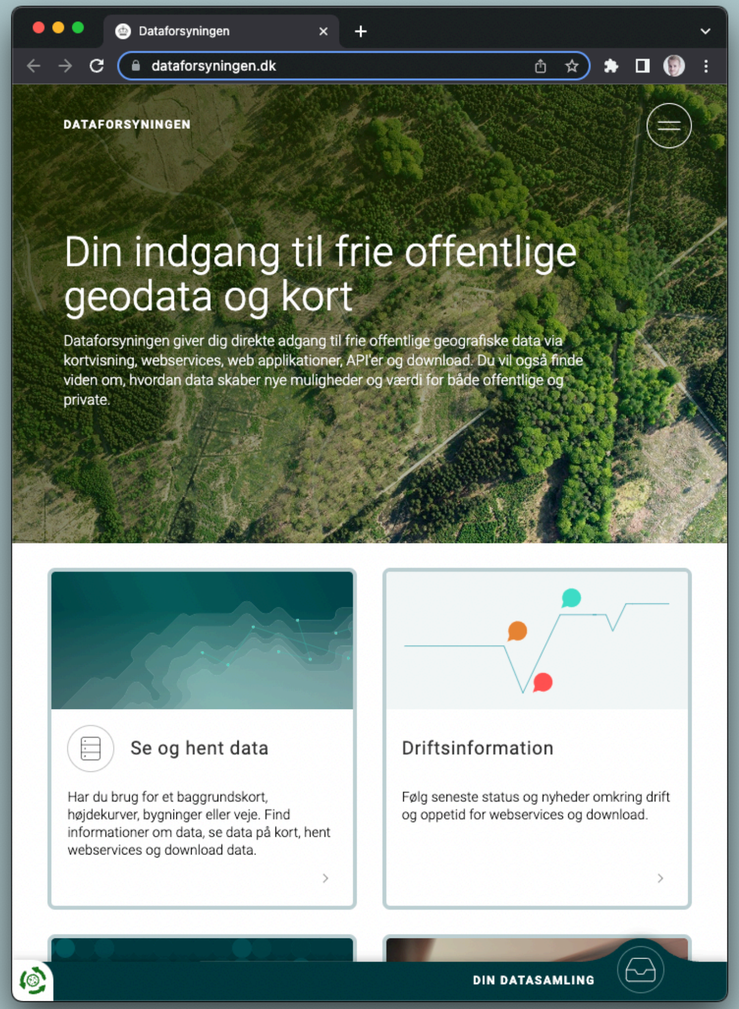
codecademy.com/article/what-is-rest



https://youtu.be/-MTSQjw5DrM

Dataforsyningen REST API

... an example of a REST API

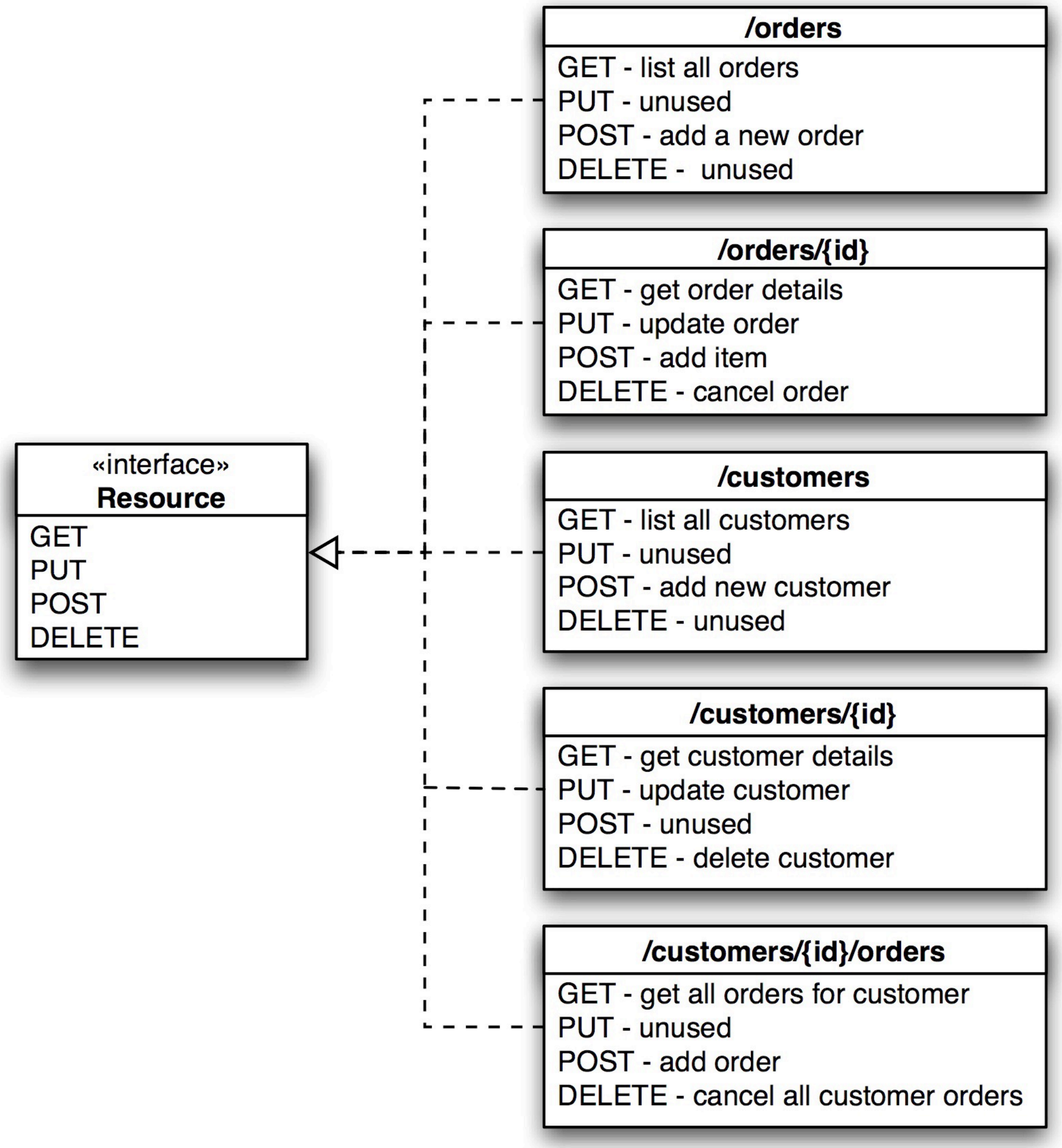


https://dataforsyningen.dk/

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

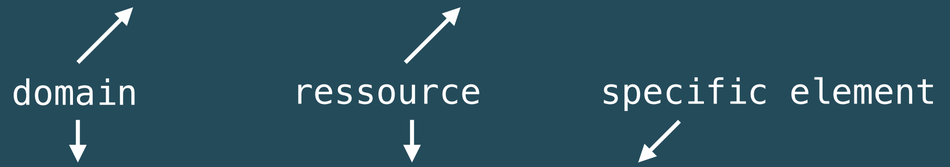


Principles of REST

- 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

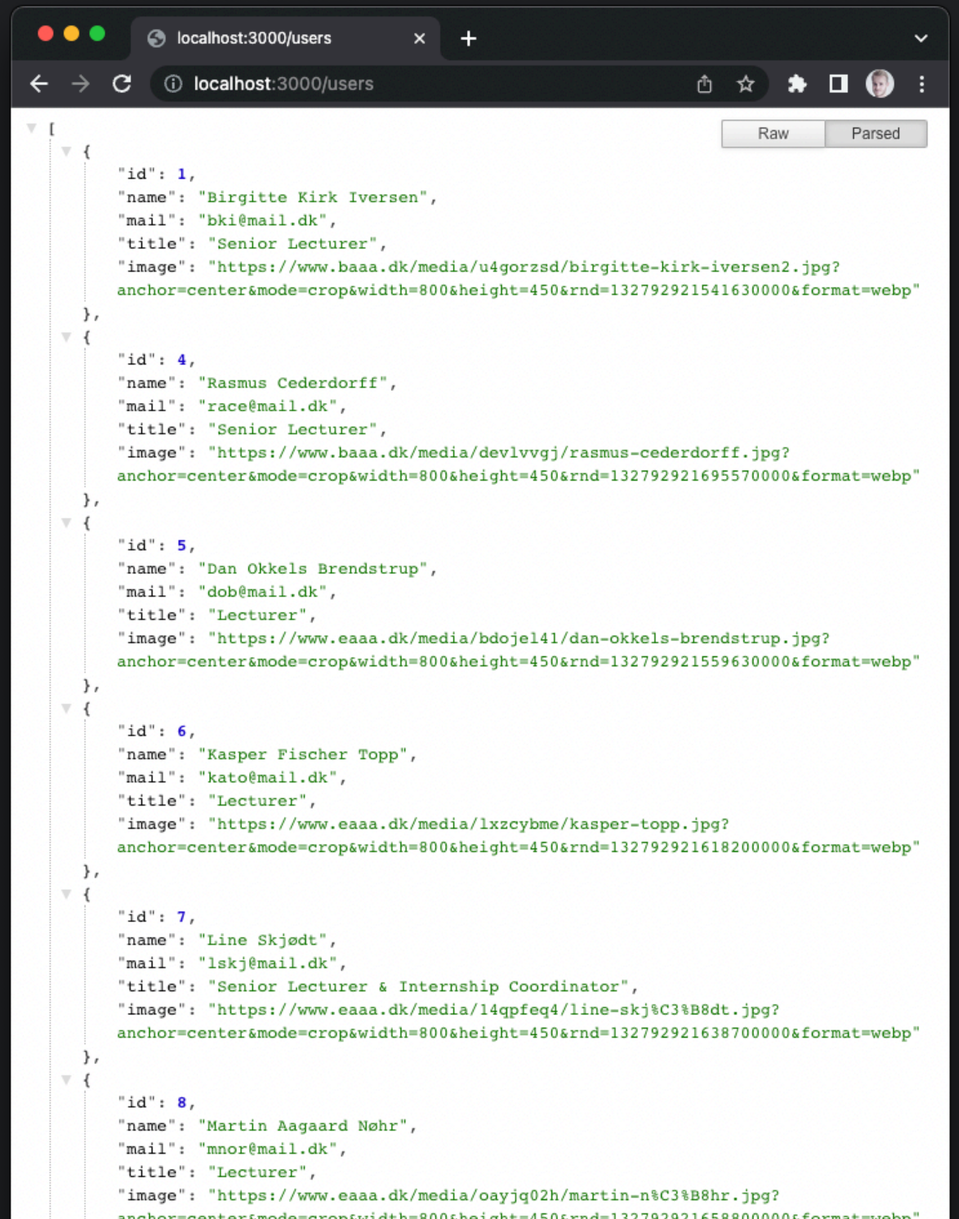


- http://some-url.com/products/product1

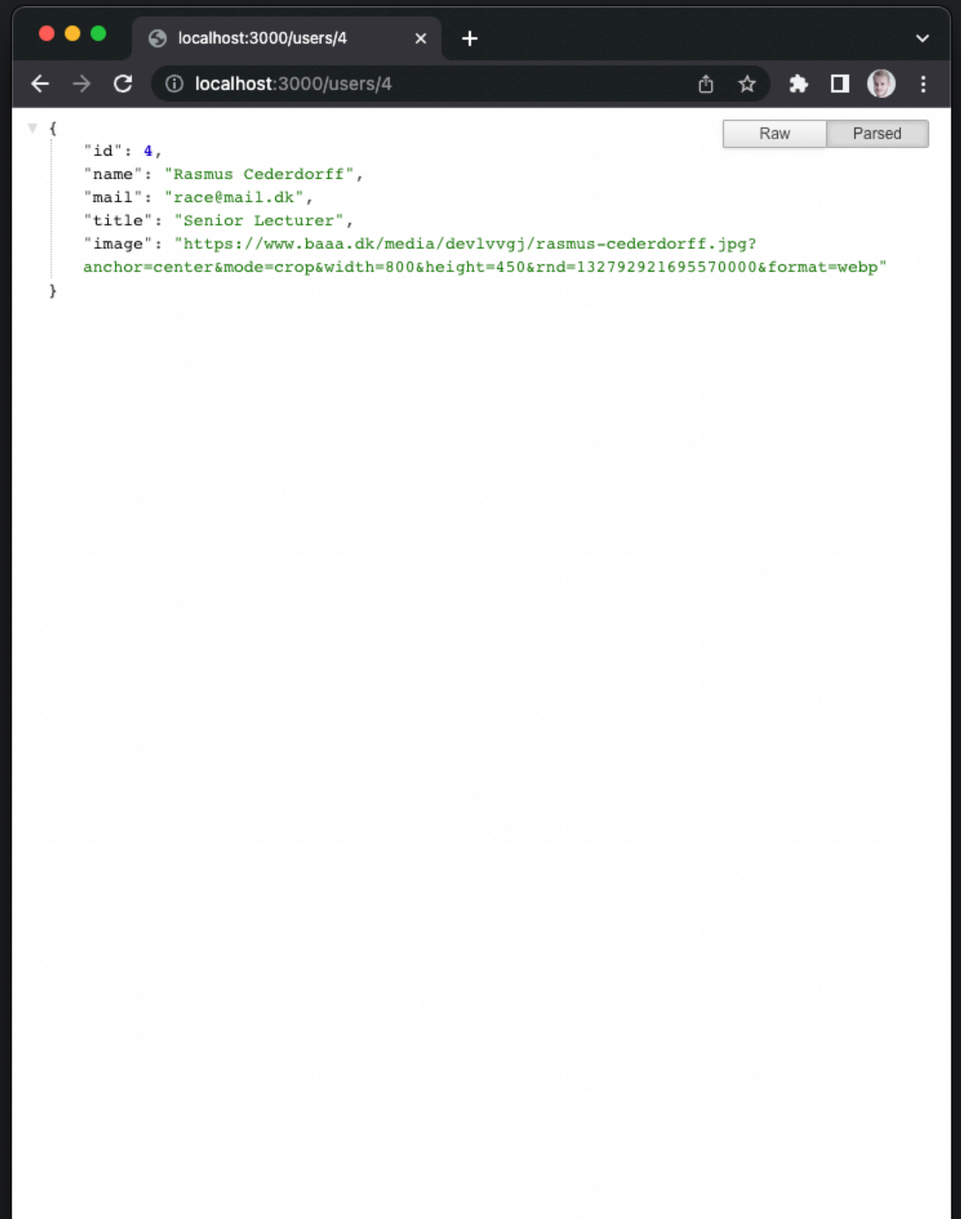
- Data type —> JSON

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

Collections (JSON Array)



Element (JSON Object)

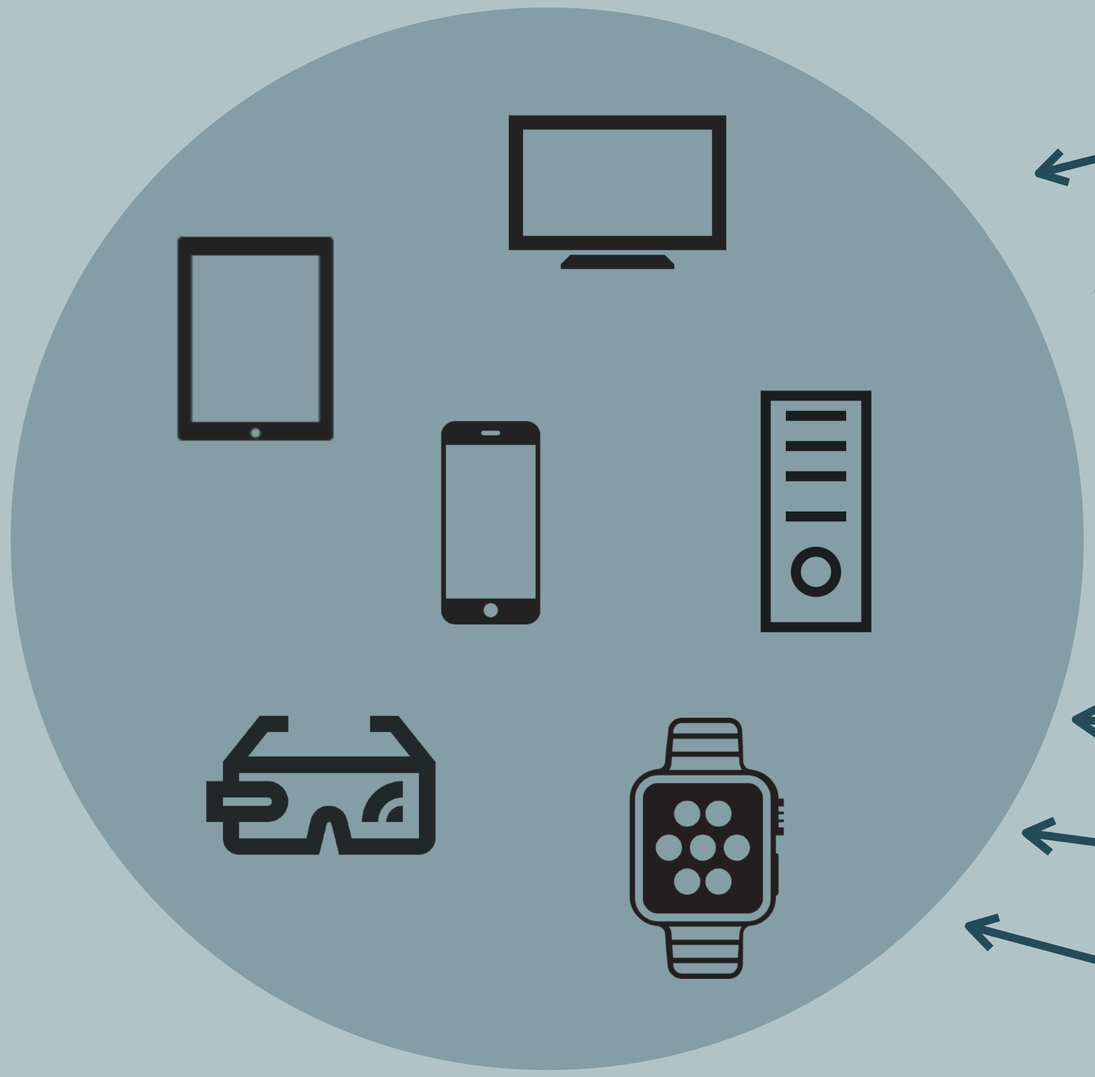


Advantages of REST

- 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

WEB APP



DATA SOURCES

API — WIKIPEDIA

API — GOOGLE MAPS

API — CVR API

API — SoMe

API — WEATHER API

API — HEADLESS CMS

API — Firebase REST

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

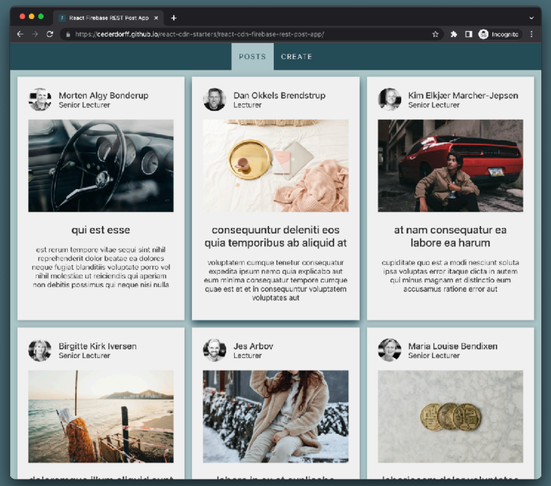
codecademy.com/article/what-is-rest

Fetch, HTTP Request & Response

WEB APP

FIREBASE REST API

DATA SOURCE



FETCH(GET, POST, PUT, DELETE)

REQUEST

JSON DATA

RESPONSE

JSON

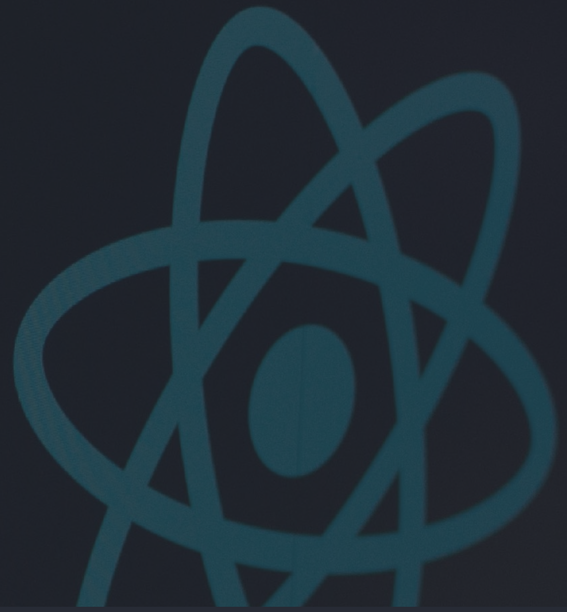
Firebase
Realtime Database

posts

users

Fetch and read posts

`GET` is the default request method for fetch.



```
async function getPosts() {
    const url = "https://race-rest-default-rtdb.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`

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
    }
}
```

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

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

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-rtdb.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-rtdb.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=iosNuIdQoy8

Build better apps

Cloud Firestore
Store and sync app data at global scale

ML Kit BETA
Machine learning for mobile developers

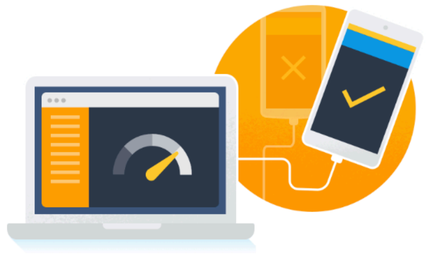
Cloud Functions
Run mobile backend code without managing servers

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



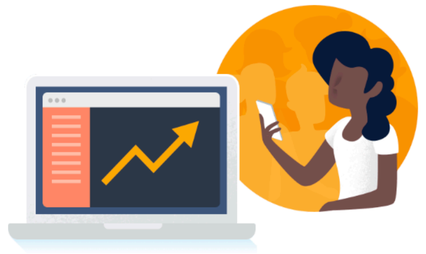
Improve app quality

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

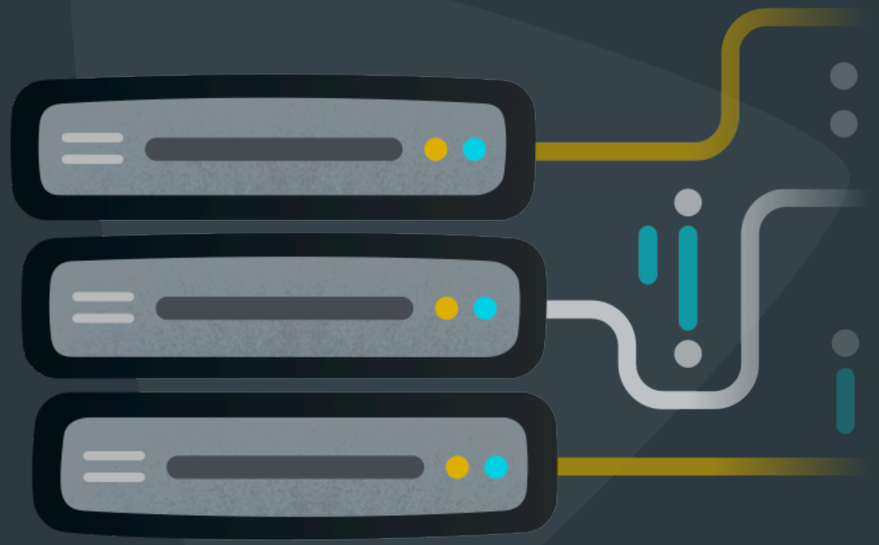
Predictions
Smart user segmentation based on predicted behavior

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



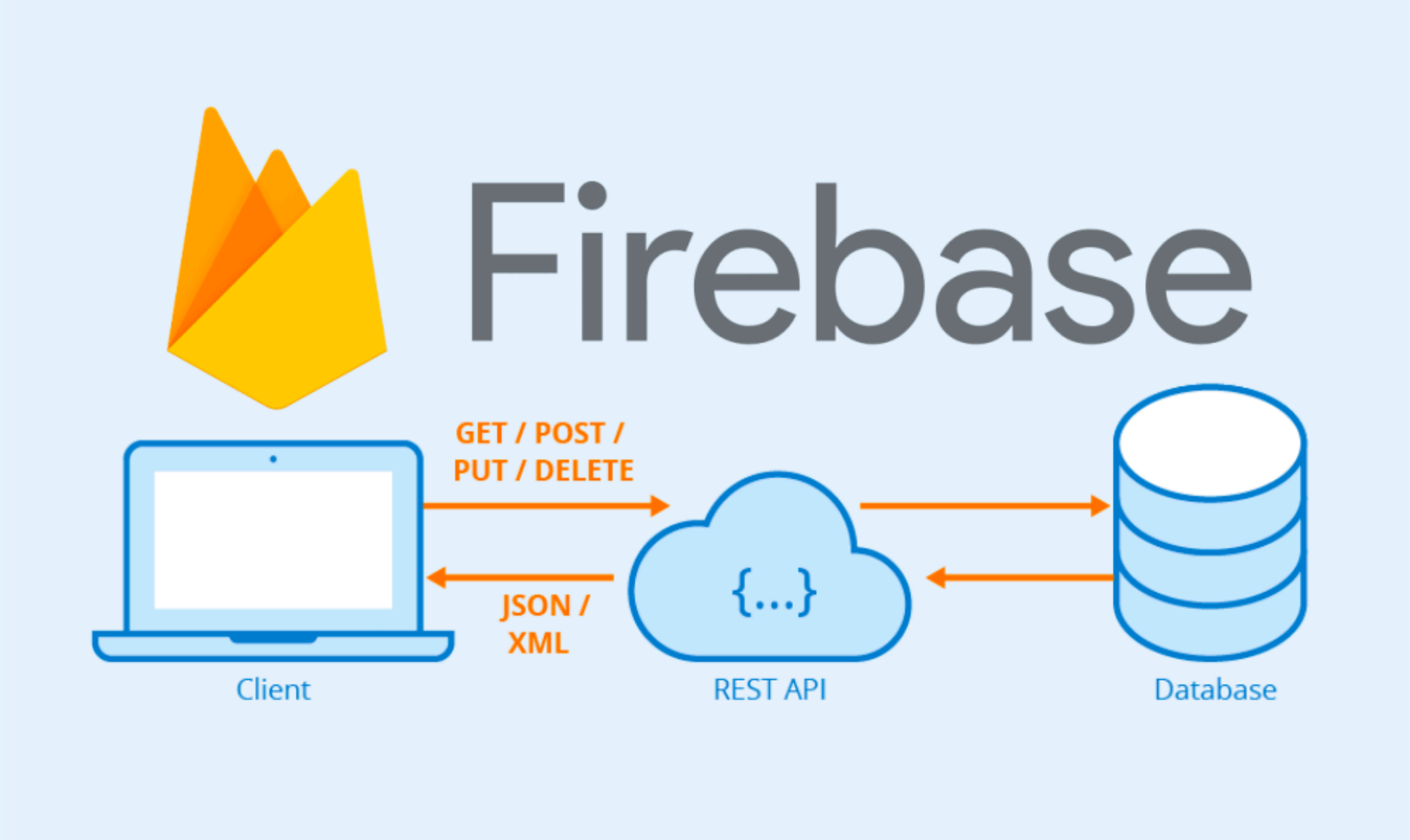
Realtime Database & REST API

Store and sync data in real time

REST API or SDK

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

Database



javascript.plainenglish.io/firebase-cloud-functions-tutorial-creating-a-rest-api-8cbc51479f80

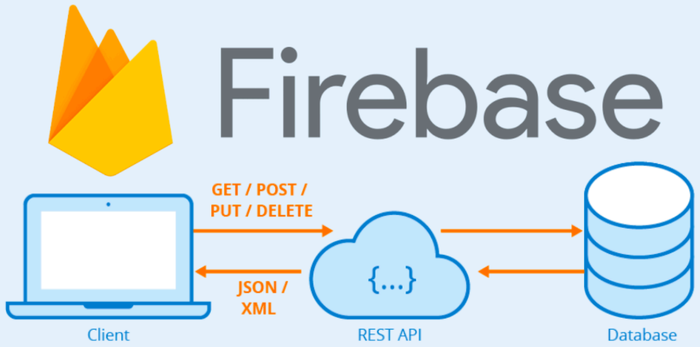
“

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.

”

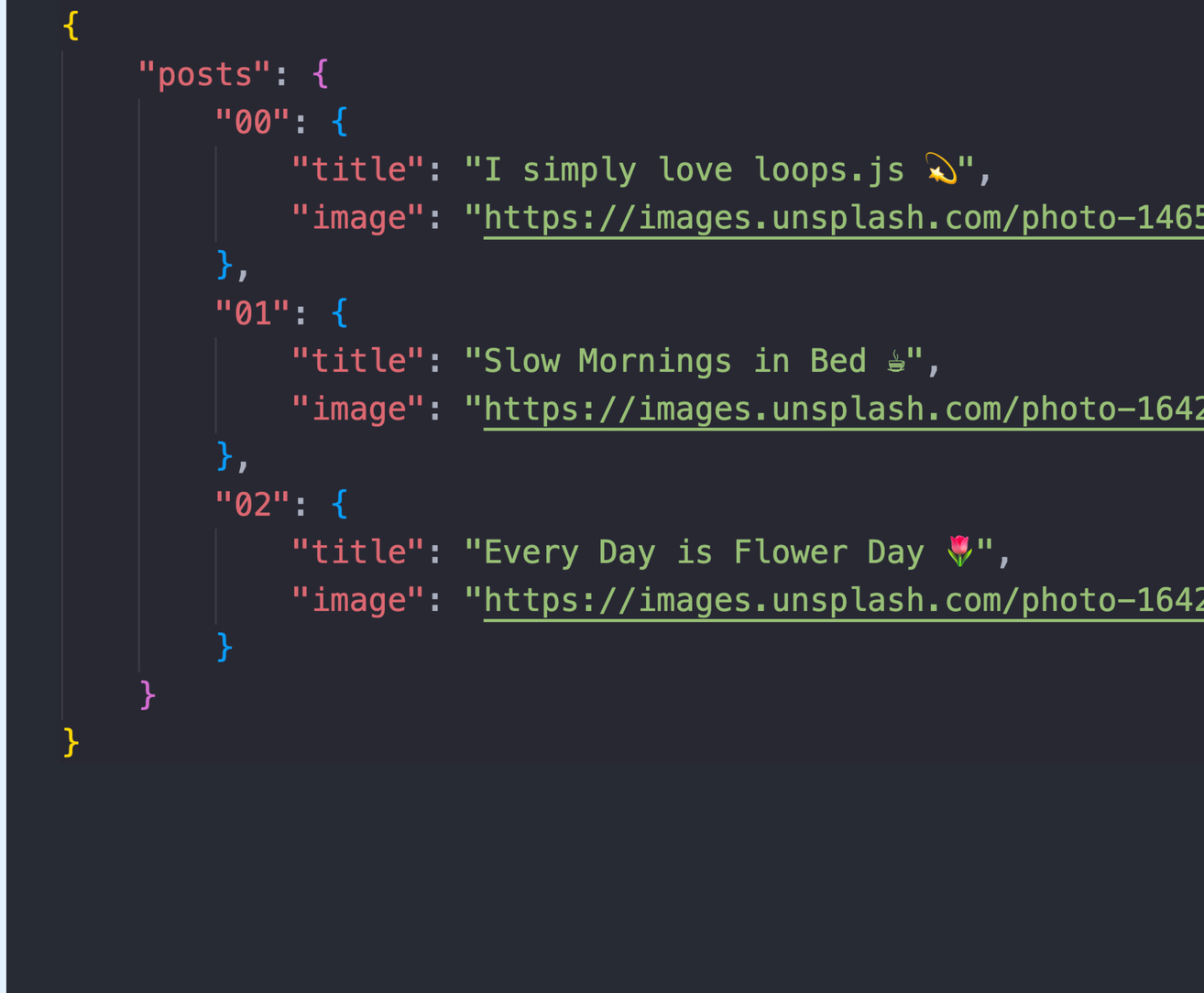
https://firebase.google.com/docs/database/rest/start



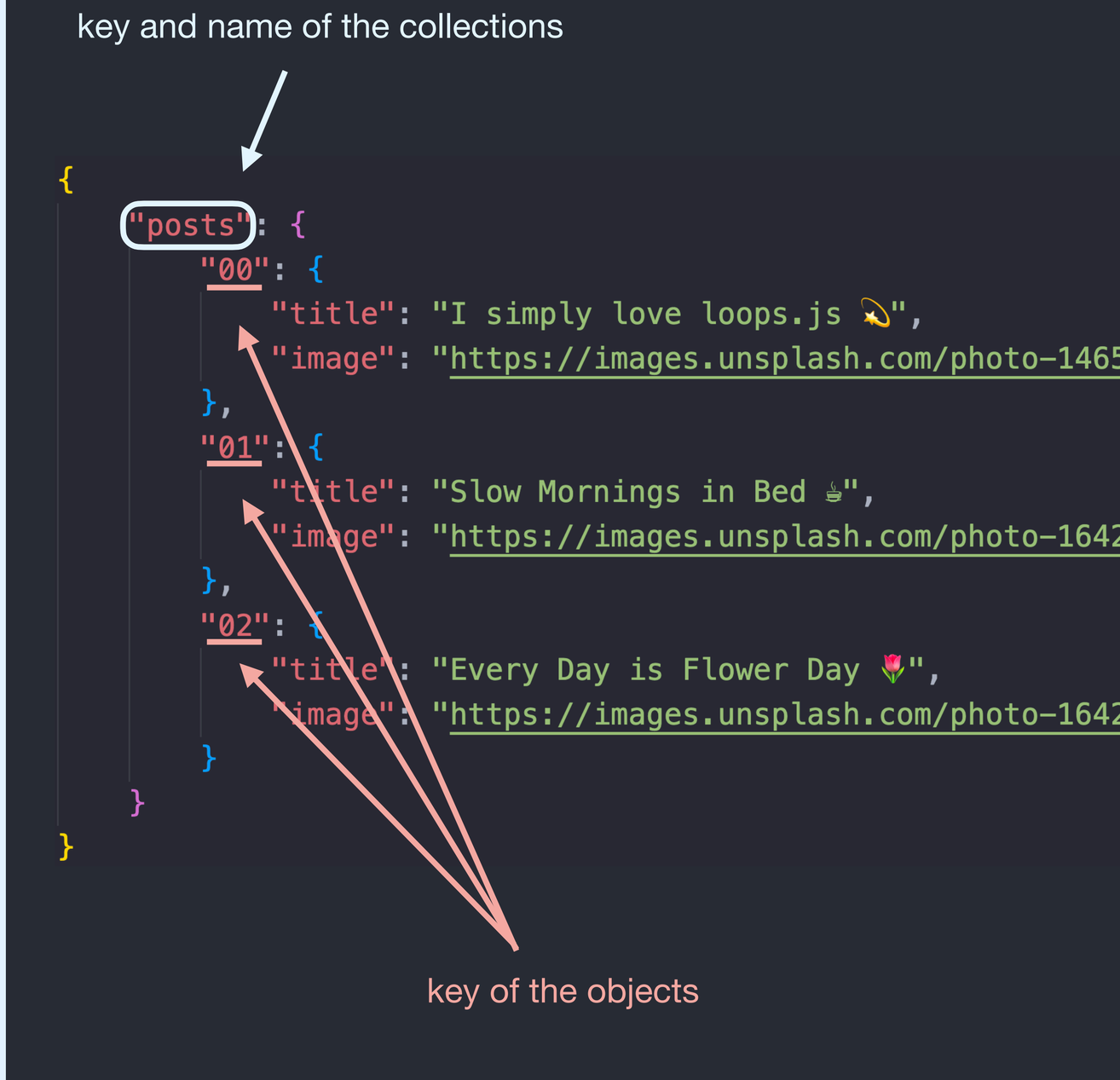
How data is structured


```
{
    "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
        }
    }
}
```

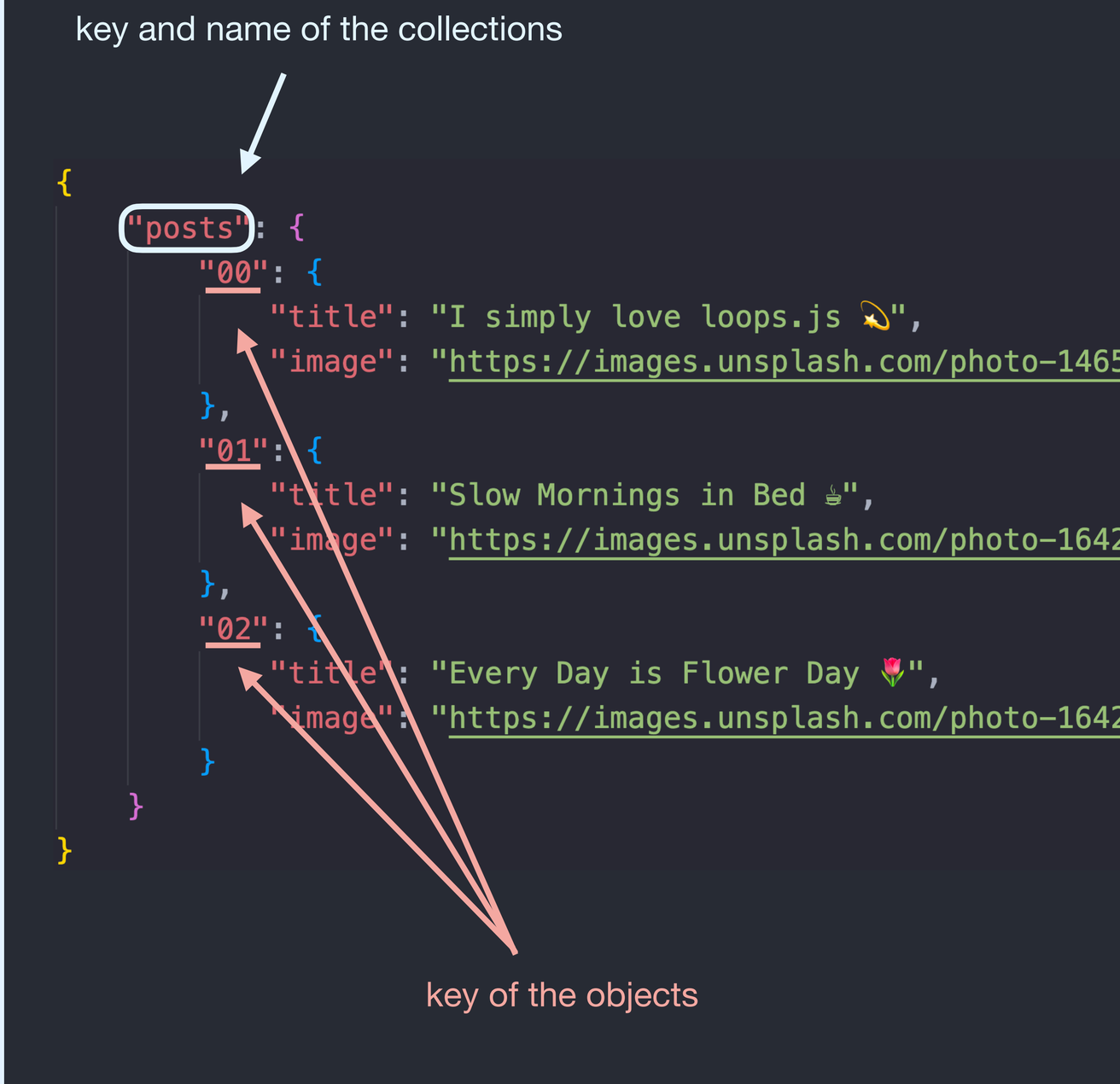
- One big object with objects.
- All objects have a unique key.
- “posts”, “00”, “01” and “02” are all keys.
- “posts” is the key of the collection of post objects (list of posts).
- “00”, “01” and “02” are keys of post objects.
- “title” and “image” are keys inside of every post object. “title” and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



- 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”]`



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

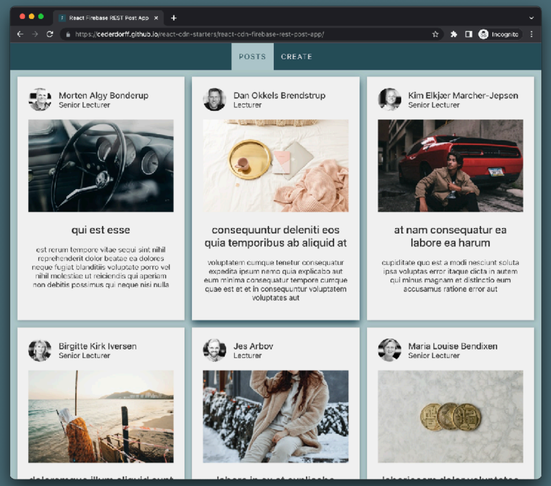


Fetch, HTTP Request & Response

WEB APP

FIREBASE REST API

DATA SOURCE



FETCH(GET, POST, PUT, DELETE)

REQUEST

JSON DATA

RESPONSE

JSON

Firebase
Realtime Database

posts

users

Realtime Database REST API

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-rtdb.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-rtdb.firebaseio.com/users.json");
        const data = await response.json();
        // mapp object into an array with objects
        const users = Object.keys(data).map(key => ({ id: key, ...data[key] }));
        return users;
    }
```


Code
Every
Day