

Loops, events og closure

Objekter, Arrays og DOM-manipulation

Agenda

- Objekter og arrays
- Fetch og JSON
- Loops
- Events og closure
- DOM-manipulation
- Specialisering af output

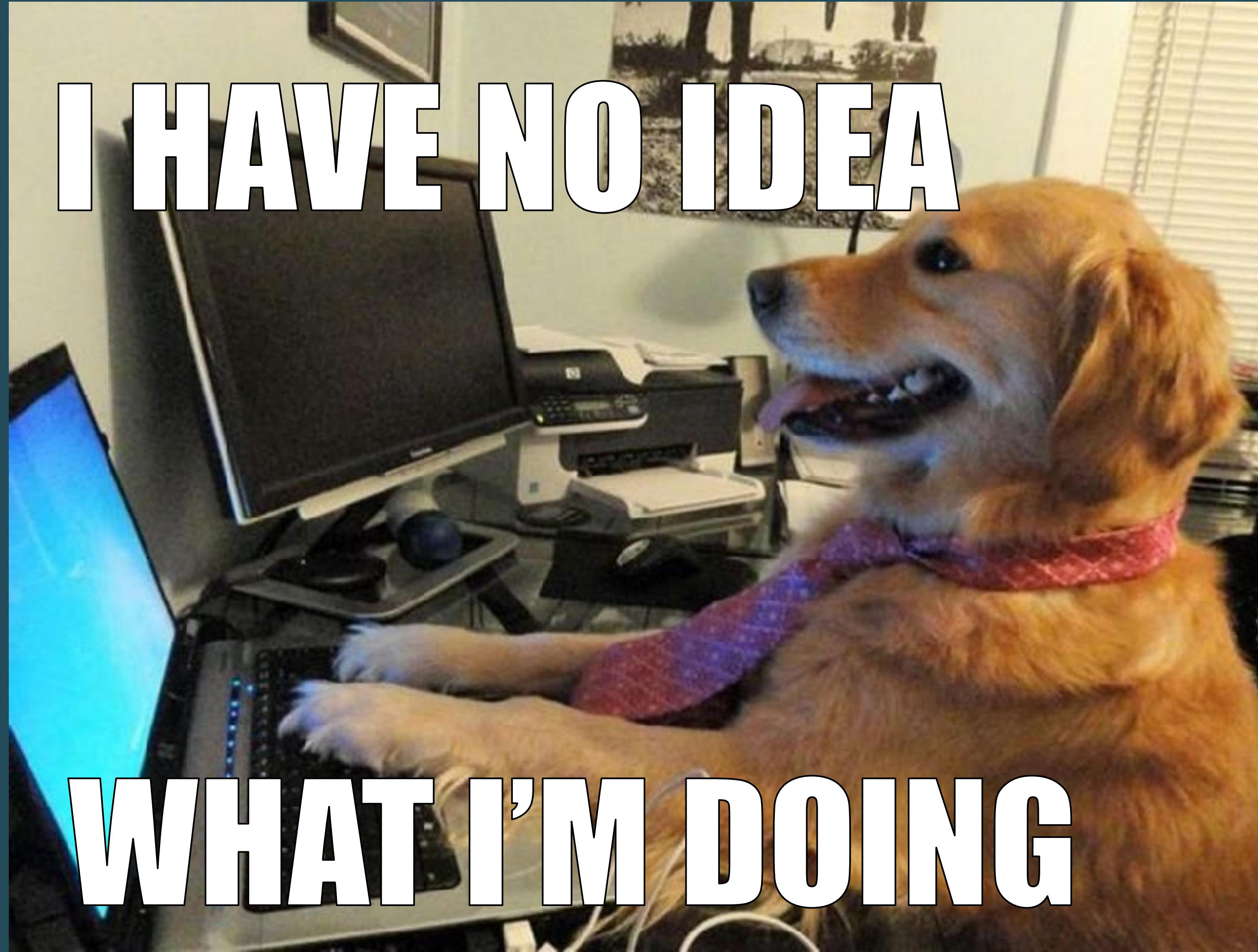
The image shows a screenshot of a code editor with a dark theme. At the top, there's a menu bar with options like View, Selection, Find, Debugger, Packages, Window, Help, and More. Below the menu, there's a toolbar with icons for selection, find, debugger, packages, window, help, and more. The main area of the editor displays a file named 'main.js' with the following content:

```
1 "use strict";
2
3 fetch("http://headlesscms.cederdorff.com/wp-json/wp/v2/posts?_embed")
4   .then(function(response) {
5     return response.json();
6   })
7   .then(function(json) {
8     appendPosts(json);
9   });
10
11 function appendPosts(posts) {
12   for (let post of posts) {
13     console.log(post);
14     document.querySelector("#grid-posts").innerHTML += `
15       <article>
16         <h3>${post.title.rendered}</h3>
17         <p>Email: <a href="mailto:${post.acf.email}">${post.acf.email}</a></p>
18         <p>Phone: ${post.acf.phone}</p>
19       </article>
20     `;
21   }
22 }
```

The code uses ES6 syntax, including arrow functions and template literals. It makes an API call to fetch posts from a WordPress site and then loops through the results to append them to a DOM element with the ID 'grid-posts'.

I HAVE NO IDEA

WHAT I'M DOING



Computer science student



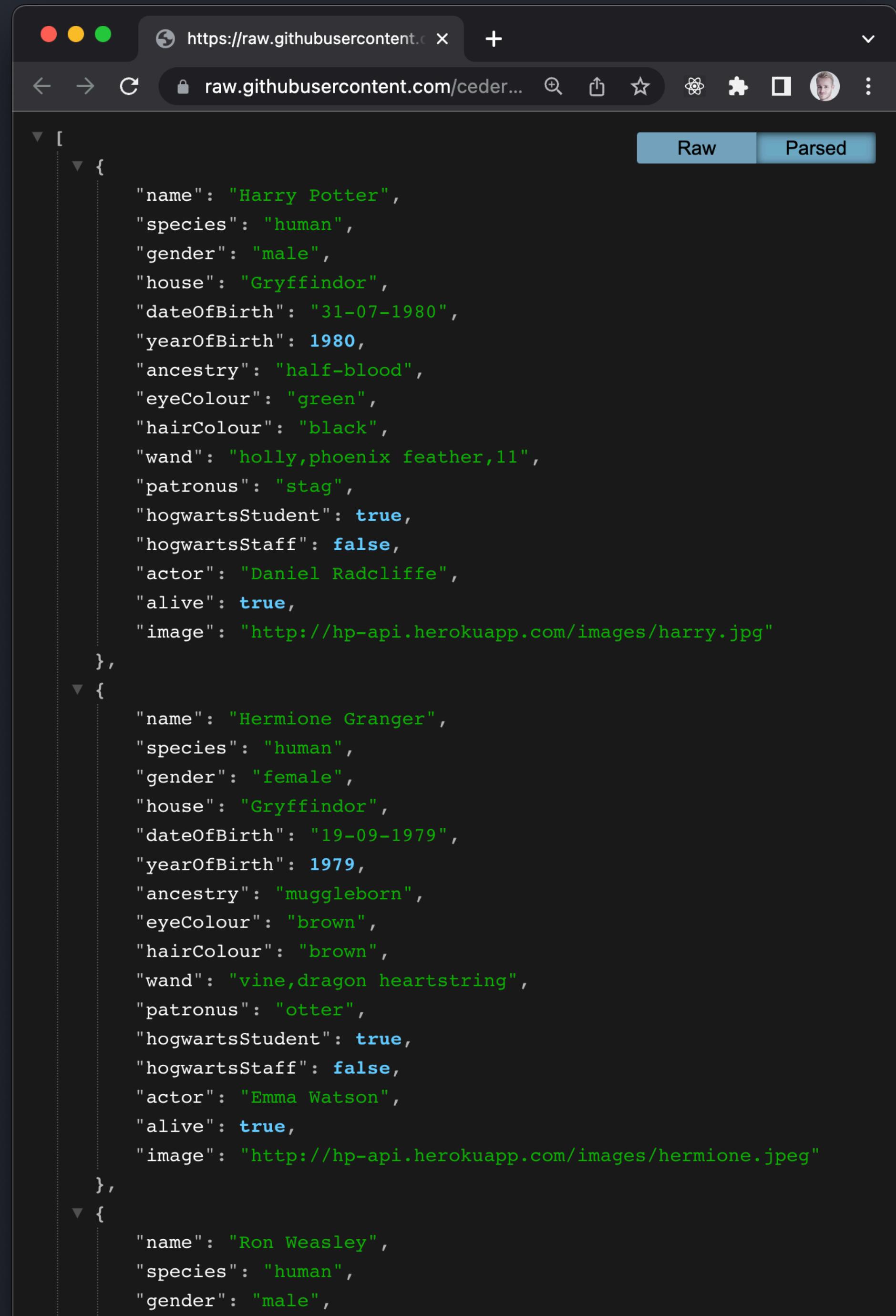
Senior developer, 10+ years experience



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

Data

JavaScript, lister,
objekter, JSON, fetch,
DOM-manipulation og
meget mere...

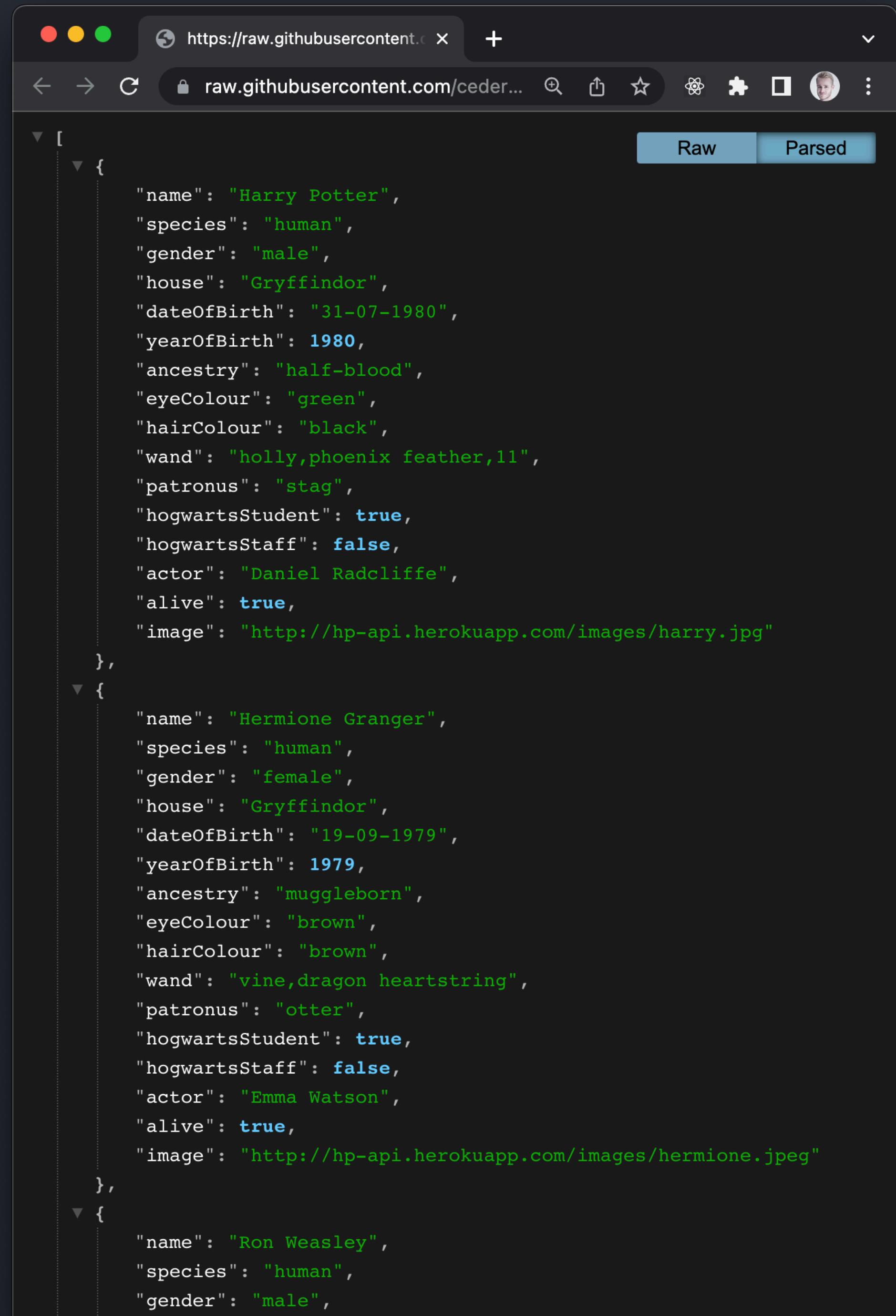


The screenshot shows a browser window displaying a JSON array of three objects representing characters from Harry Potter. The objects are labeled with their names and contain various properties such as species, gender, house, date of birth, year of birth, ancestry, eye colour, hair colour, wand material, patronus, and whether they are a Hogwarts student or staff member. The JSON is displayed in a dark-themed developer tools interface.

```
[{"name": "Harry Potter", "species": "human", "gender": "male", "house": "Gryffindor", "dateOfBirth": "31-07-1980", "yearOfBirth": 1980, "ancestry": "half-blood", "eyeColour": "green", "hairColour": "black", "wand": "holly,phoenix feather,11", "patronus": "stag", "hogwartsStudent": true, "hogwartsStaff": false, "actor": "Daniel Radcliffe", "alive": true, "image": "http://hp-api.herokuapp.com/images/harry.jpg"}, {"name": "Hermione Granger", "species": "human", "gender": "female", "house": "Gryffindor", "dateOfBirth": "19-09-1979", "yearOfBirth": 1979, "ancestry": "muggleborn", "eyeColour": "brown", "hairColour": "brown", "wand": "vine,dragon heartstring", "patronus": "otter", "hogwartsStudent": true, "hogwartsStaff": false, "actor": "Emma Watson", "alive": true, "image": "http://hp-api.herokuapp.com/images/hermione.jpeg"}, {"name": "Ron Weasley", "species": "human", "gender": "male", "house": "Gryffindor", "dateOfBirth": "11-08-1980", "yearOfBirth": 1980, "ancestry": "pureblood", "eyeColour": "brown", "hairColour": "red", "wand": "willow,mongoose hair,12", "patronus": "ewe", "hogwartsStudent": true, "hogwartsStaff": false, "actor": "Rupert Grint", "alive": true, "image": "http://hp-api.herokuapp.com/images/ron.jpg"}]
```

Hvordan viser vi denne JSON-data, så det giver mening for brugerne?

- Denne rette visualisering
- Generere HTML
- DOM-manipulation



The screenshot shows a browser window displaying JSON data for three characters: Harry Potter, Hermione Granger, and Ron Weasley. The data is presented in a hierarchical tree view, where each character is represented by a brace {}, and each character's properties are listed inside. The properties include name, species, gender, house, date of birth, year of birth, ancestry, eye colour, hair colour, wand material, patronus, Hogwarts status, staff status, actor, alive status, and image URL. The JSON is color-coded for readability, with green for strings and blue for booleans.

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

Fetch Persons

127.0.0.1:5503/fetch-persons-grid/index.html

Apps KEA DAT DAT.js IT Arkitektur EAAA BOOKS

Fetch Persons

 <p>Birgitte Kirk Iversen Senior Lecturer hki@mail.dk</p>	 <p>Martin Aagaard Nøhr Lecturer mnr@mail.dk</p>	 <p>Rasmus Cederdorff Senior Lecturer rnc@mail.dk</p>
 <p>Dan Okkels Brendstrup Lecturer dob@mail.dk</p>	 <p>Line Skjødt Senior Lecturer & Internship Coordinator lskj@mail.dk</p>	 <p>Kasper Fischer Topp Lecturer keto@mail.dk</p>
 <p>Anne Kirketerp Head of Department anki@mail.dk</p>	 <p>Maria Louise Bendixen Senior Lecturer mlbe@basa.dk</p>	 <p>Marlene Ahlgreen Jensen Senior Lecturer maj@basa.dk</p>

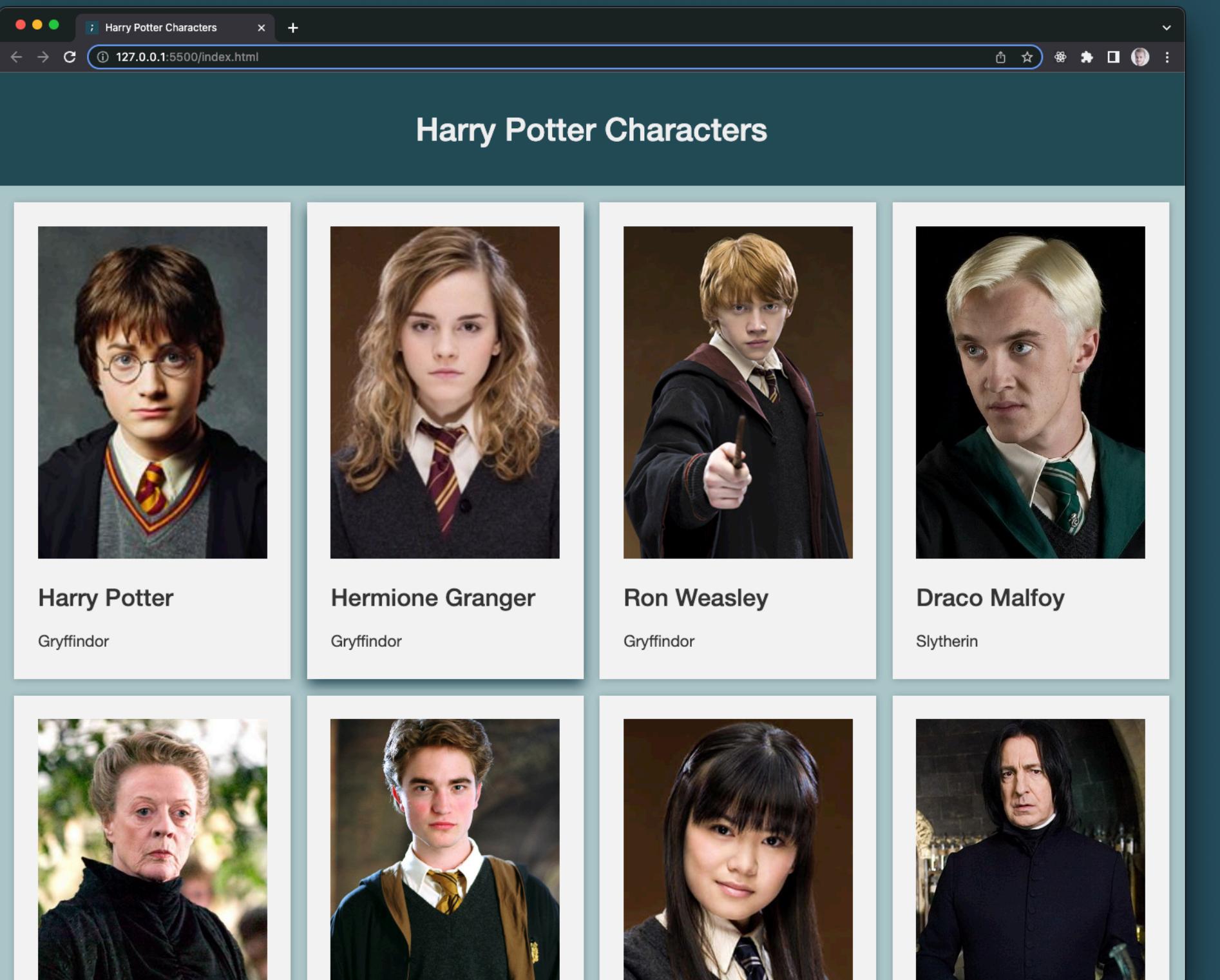
Forståelse for objekter og arrays

Programming knowledge

Backend & Frontend



Frontend (client)



<https://cederdorff.github.io/potter-app/>

JSON Data Source (Server)

The screenshot shows a browser window displaying a JSON object representing Harry Potter characters. The JSON is structured as follows:

```
[{"name": "Harry Potter", "species": "human", "gender": "male", "house": "Gryffindor", "dateOfBirth": "31-07-1980", "yearOfBirth": 1980, "ancestry": "half-blood", "eyeColour": "green", "hairColour": "black", "wand": "holly,phoenix feather,11", "patronus": "stag", "hogwartsStudent": true, "hogwartsStaff": false, "actor": "Daniel Radcliffe", "alive": true, "image": "http://hp-api.herokuapp.com/images/harry.jpg"}, {"name": "Hermione Granger", "species": "human", "gender": "female", "house": "Gryffindor", "dateOfBirth": "19-09-1979", "yearOfBirth": 1979, "ancestry": "muggleborn", "eyeColour": "brown", "hairColour": "brown", "wand": "vine,dragon heartstring", "patronus": "otter", "hogwartsStudent": true, "hogwartsStaff": false, "actor": "Emma Watson", "alive": true, "image": "http://hp-api.herokuapp.com/images/hermione.jpeg"}, {"name": "Ron Weasley", "species": "human", "gender": "male", "house": "Gryffindor", "dateOfBirth": "01-03-1980", "yearOfBirth": 1980, "ancestry": "pure-blood", "eyeColour": "blue", "hairColour": "red", "wand": "willow,unicorn tail-hair,14", "patronus": "Jack Russell terrier", "hogwartsStudent": true, "hogwartsStaff": false, "actor": "Rupert Grint", "alive": true, "image": "http://hp-api.herokuapp.com/images/ron.jpg"}, {"name": "Draco Malfoy", "species": "human", "gender": "male", "house": "Slytherin", "dateOfBirth": "16-07-1980", "yearOfBirth": 1980, "ancestry": "pure-blood", "eyeColour": "grey", "hairColour": "blonde", "wand": "rowan,serpent hair,12", "patronus": "dementor", "hogwartsStudent": false, "hogwartsStaff": false, "actor": "Tom Felton", "alive": true, "image": "http://hp-api.herokuapp.com/images/draco.jpg"}]
```

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

Objects

A set of named values

Objects are used to store keyed
collections of various data



Containers for named values
called properties. A property
is a “key: value” pair

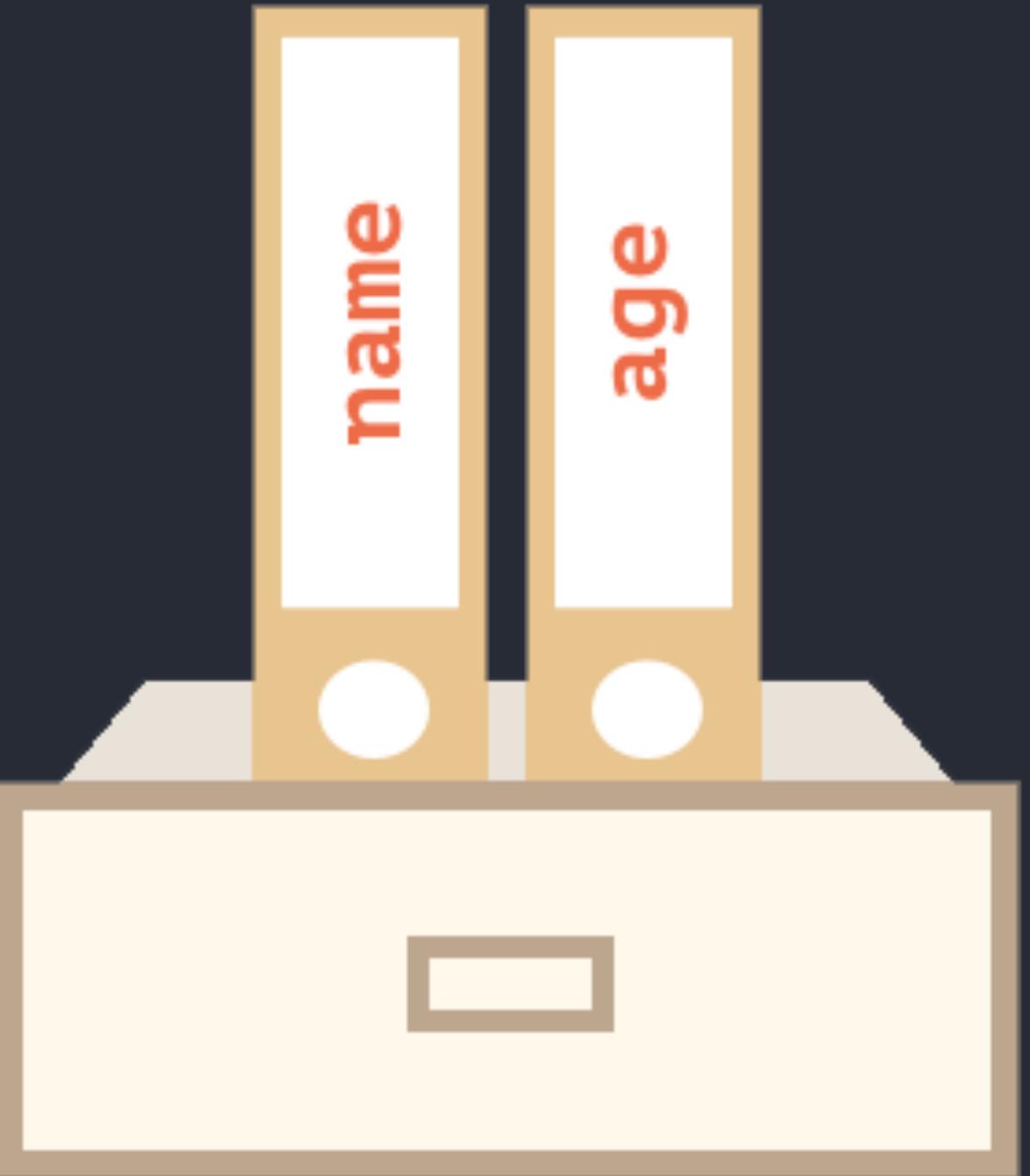
Objects

A set of named values

```
let user = {  
    name: 'Alicia',  
    age: 6  
};
```

```
console.log(user.name +  
    " is " + user.age +  
    " years old.");
```

user



Alicia is 6 years old.

main.js:11

Hvad ser du?

The screenshot shows the KEA website's search interface. At the top, there is a navigation bar with links for 'For studerende', 'For ansatte', 'For alumner', 'Blog', 'Bibliotek', 'Job', and 'Kontakt'. Below the navigation bar, there is a search bar with the placeholder 'DEL AF NAVN, E-MAIL ELLER TELEFON' and a dropdown menu for 'AFDELING' set to 'Alle'. A search button labeled 'Søg...' is located below the search bar. To the right of the search bar, there is a circular profile picture of a man with a beard, identified as Rasmus Cederdorff. Below the profile picture, his name is listed as 'RASMUS CEDERDORFF' with the title 'Lektor'. His contact information is provided: 'E: RACE@kea.dk', 'KEA Digital', 'Guldbergsgade 29N', '2200 København N', and a link to 'LÆS MERE'. At the bottom of the page, there is a footer section with links for 'KEA - KØBENHAVNS ERHVERVSAKADEMI', 'OM OS', 'FOR STUDERENDE', and 'PRAKTISK INFO'.

This screenshot shows a detailed view of the search result for 'RASMUS CEDERDORFF'. The search bar at the top contains 'RACE'. The search result card for Rasmus Cederdorff is displayed, featuring his profile picture, name, title, and contact information. The card also includes a LinkedIn link (<https://linkedin.com/in/cederdorff>), an Instagram link (<https://instagram.com/cederdorff>), and a personal website link (<https://cederdorff.com>). The background of the page is dark, and the search result card has a light gray background. The footer at the bottom is identical to the one in the first screenshot.

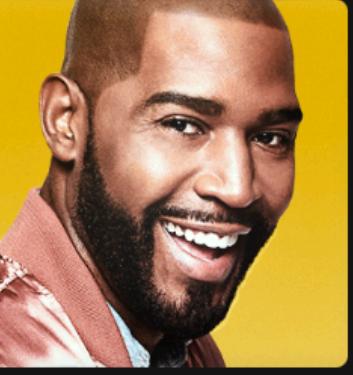
Hvad ser du?

The screenshot shows a web browser window with the URL kea.dk/find-medarbejder#qx-kea-phonebook-7910. The page title is "Find en medarbejder - Københavns Erhvervsakademi". The header includes links for "For studerende", "For ansatte", "For alumner", "Blog", "Bibliotek", "Job", and "Kontakt". A language switch to "EN" and a search icon are also present. The main content area has a breadcrumb trail "FORSIDEN / FIND EN MEDARBEJDER" and a large heading "FIND EN MEDARBEJDER". Below this is a search form with fields for "DEL AF NAVN, E-MAIL ELLER TELEFON" (containing "RACE") and "AFDELING" (containing "Alle"). A "Søg..." button is part of the form. To the right, there is a circular profile picture of a man with a beard, identified as "RASMUS CEDERDORFF", a "Lektor", with contact information: "E: RACE@kea.dk", "KEA Digital", "Guldbergsgade 29N", "2200 København N", and a "LÆS MERE" link. At the bottom of the page, there are footer sections for "KEA - KØBENHAVNS ERHVERVSAKADEMI", "OM OS", "FOR STUDERENDE", and "PRAKTISK INFO".

```
const race = {  
    name: "Rasmus Cederdorff",  
    position: "Lektor",  
    mail: "race@kea.dk",  
    department: "KEA Digital",  
    address: "Guldbergsgade 29N, 2200 København N",  
    image: "https://kea.dk/slir/w180-c1x1/images/us";  
};
```

NETFLIX

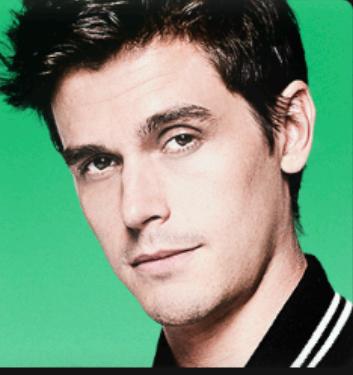
Hvem ser?



Personen der
rent faktisk
betaler for
profilen



Nasser 1



Nasser 2



Nasser 3



Nasser 4 Khader

Administrer profiler

● ○ ● | □ | < > ⌂ +

netflix.com

NETFLIX Start Serier Film Nyt og populært Min liste

N SERIE

TOO HOT TO HANDLE

TOP 10 Nr. 4 i Danmark i dag

På paradisets kyst mødes de lækkere singler og mingler. Men der er et tvist. For at vinde den attraktive pengepræmie, må de give afkald på at have sex.

Afspil Mere info

13+

Kun på Netflix

TOO HOT TO HANDLE NYE EPISODER

EMILY IN PARIS

QUEER EYE more than a makeover

The Woman in the House Across the Street From the Girl in the Window

BRIDGERTON NYE EPISODER

Se videre med profilen Nasser 1

the office

TIGER KING

Don't Look UP

JEFFREY EPSTEIN: FILTHY RICH

THE MIND explained

Frost II (2019) - IMDb

imdb.com/title/tt4520988/

IMDb Menu All Search IMDb

Frost II

Original title: Frozen II
2019 · 7 · 1h 43m

IMDb RATING YOUR RATING POPULARITY

★ 6.8/10 160K ★ Rate 896 ▲ 102

Cast & crew · User reviews · Trivia · IMDbPro 🔍 All topics | [Share](#)

+ Play trailer 0:16

55 VIDEOS

99+ PHOTOS

Animation Adventure Comedy

+ Add to Watchlist

Anna, Elsa, Kristoff, Olaf and Sven leave Arendelle to travel to an ancient, autumn-bound forest of an enchanted land. They set out to find the origin of Elsa's powers in order to save their kingdom.

1.4K User reviews 289 Critic reviews 64 Metascore

Directors Chris Buck · Jennifer Lee

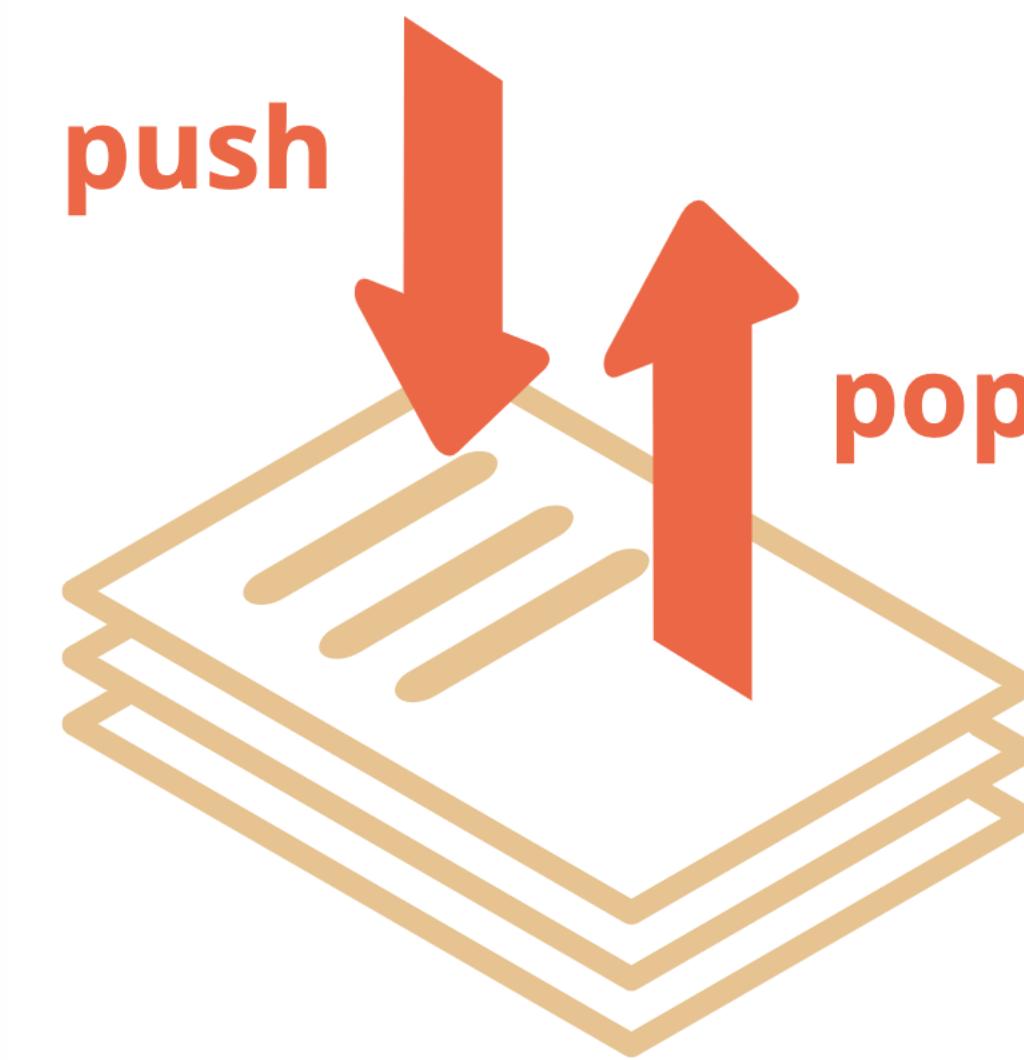
Writers

```
let movie = {  
  title: "Frozen 2",  
  description: "Elsa the Snow Queen has a",  
  trailer: "https://www.youtube.com/embed",  
  length: "1h 43m",  
  year: "2019"  
}
```

Arrays

Collections

Ordered collection of values or
objects



An array is a way to hold more than one value at a time we have a 1st, a 2nd, a 3rd, a 4th element and so on.

Hvad ser du?

The screenshot shows a web browser window for the KEA website. The URL is kea.dk/find-medarbejder#qx-kea-phonebook-7910. The page title is "Find en medarbejder". The header includes the KEA logo and navigation links for students, staff, alumni, blog, library, jobs, and contact. A search bar at the top has fields for "DEL AF NAVN, E-MAIL ELLER TELEFON" and "AFDELING" (set to "KEA Digital"), with a "Søg..." button. Below the search bar, there are five circular profile pictures. The first three profiles have associated data:

Name	Position	Mail	Department	Address	Image
LARS BOGETOFT	Uddannelseschef	T: 51850497 M: 51850497 E: larb@kea.dk	KEA Digital	Guldbergsgade 29N 2200 København N	https://share.cederdorff.com/images/lars-bogetoft.jpg
MAGDALENA MARIA OTAP	Adjunktvikar	E: mago@kea.dk	KEA Digital	Guldbergsgade 29N 2200 København N	https://share.cederdorff.com/images/magdalena-maria-otap.jpg
OSKAR TUSKA	Adjunkt	E: ostu@kea.dk	KEA Digital	Guldbergsgade 29N 2200 København N	https://share.cederdorff.com/images/oskar-tuska.jpg

The last two profiles are placeholder icons.

```
const lecturers = [
  {
    name: "Peter Lind",
    position: "Lektor",
    mail: "petl@kea.dk",
    department: "KEA Digital",
    address: "Guldbergsgade 29N, 2200 København N",
    image: "https://share.cederdorff.com/images/peter-lind.jpg"
  },
  {
    name: "Rasmus Cederdorff",
    position: "Lektor",
    mail: "race@kea.dk",
    department: "KEA Digital",
    address: "Guldbergsgade 29N, 2200 København N",
    image: "https://kea.dk/slir/w180-c1x1/images/rasmus-cederdorff.jpg"
  }
];
```

CRUD App

Lars Bogetoft
Head of Education
lrb@kea.dk

Peter Lind
Senior Lecturer
pet@kea.dk

Magdalena "Lenka" Otap
Lecturer
mago@kea.dk

Rasmus Cederdorff
Senior Lecturer
race@kea.dk

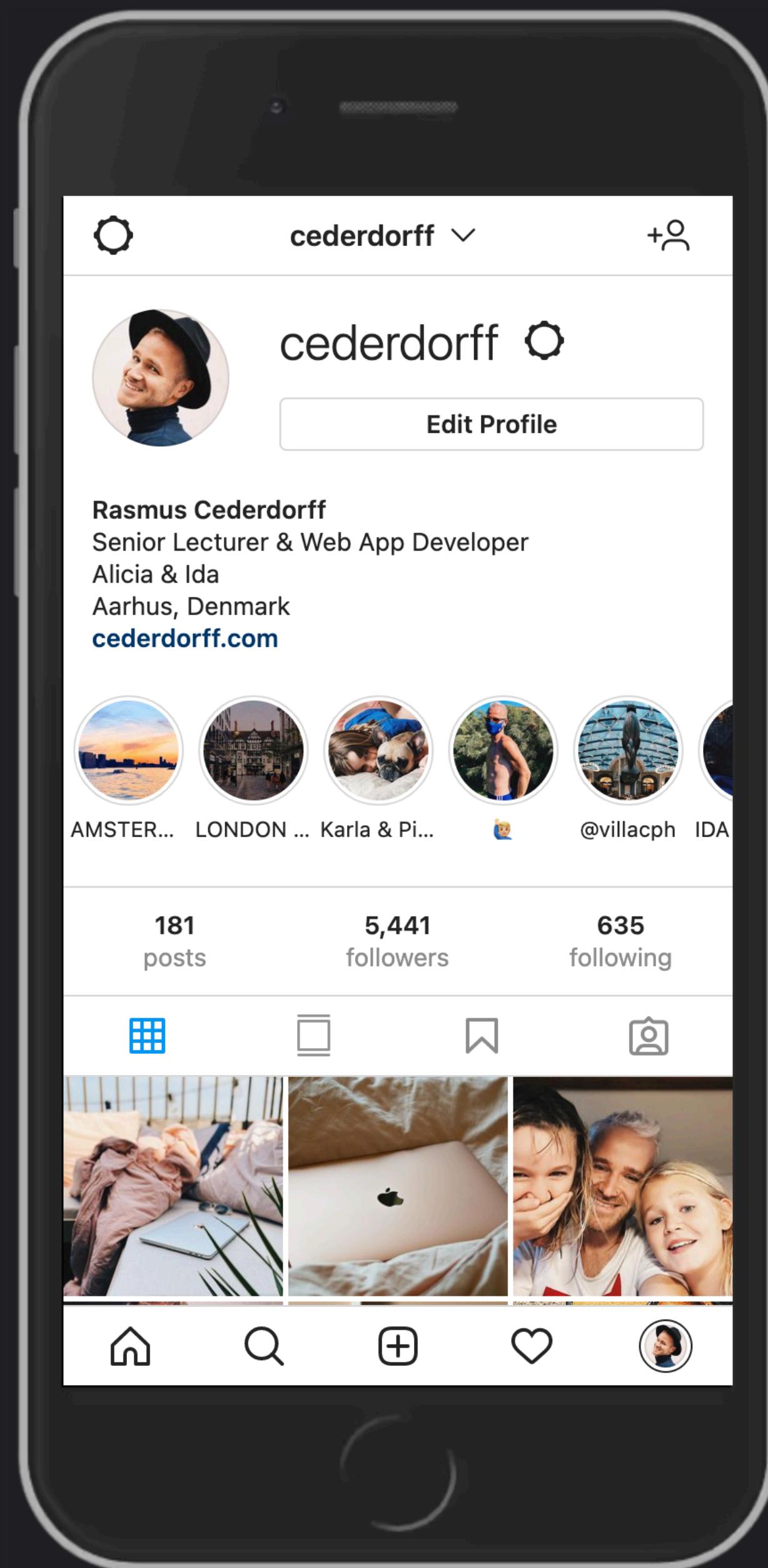
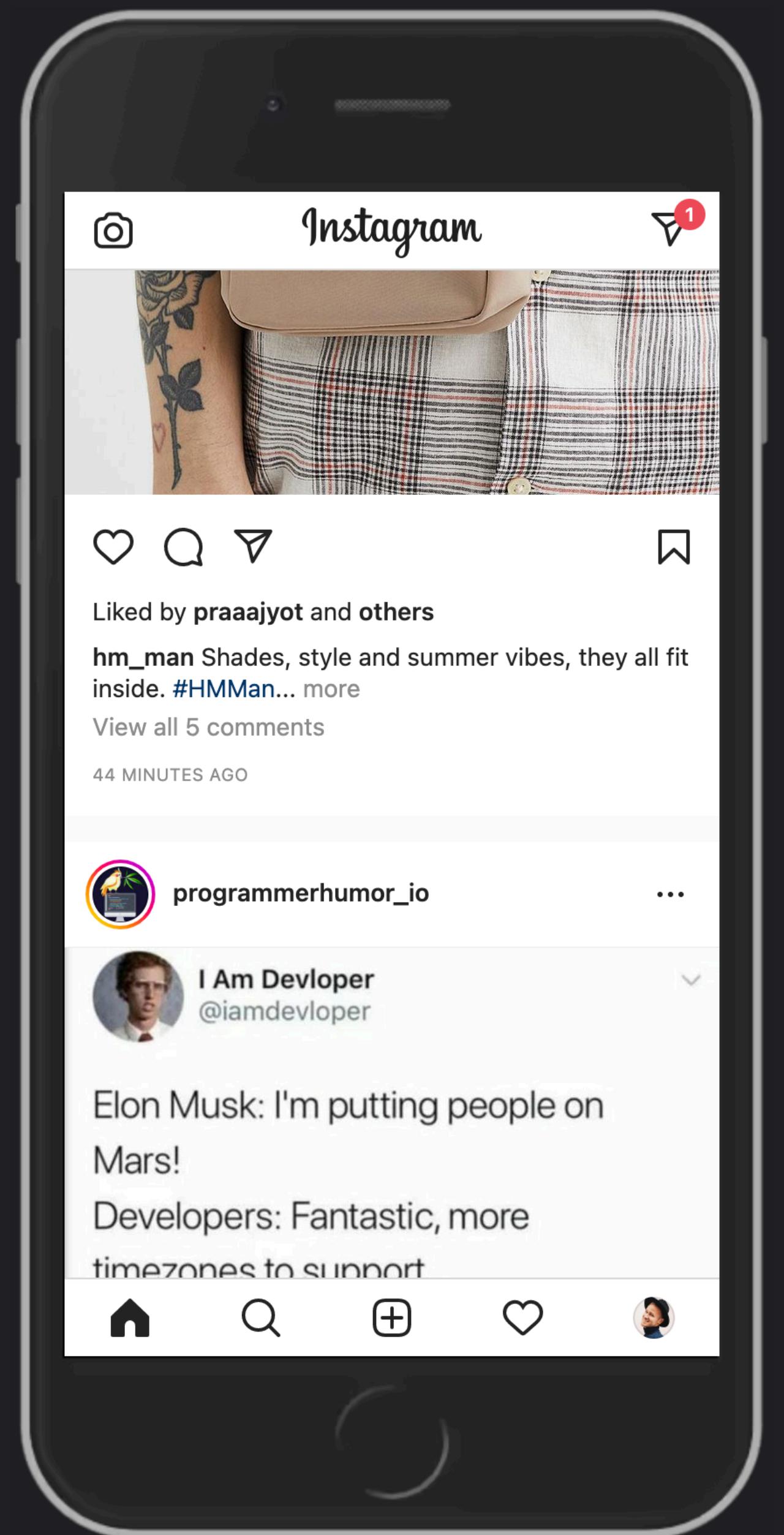
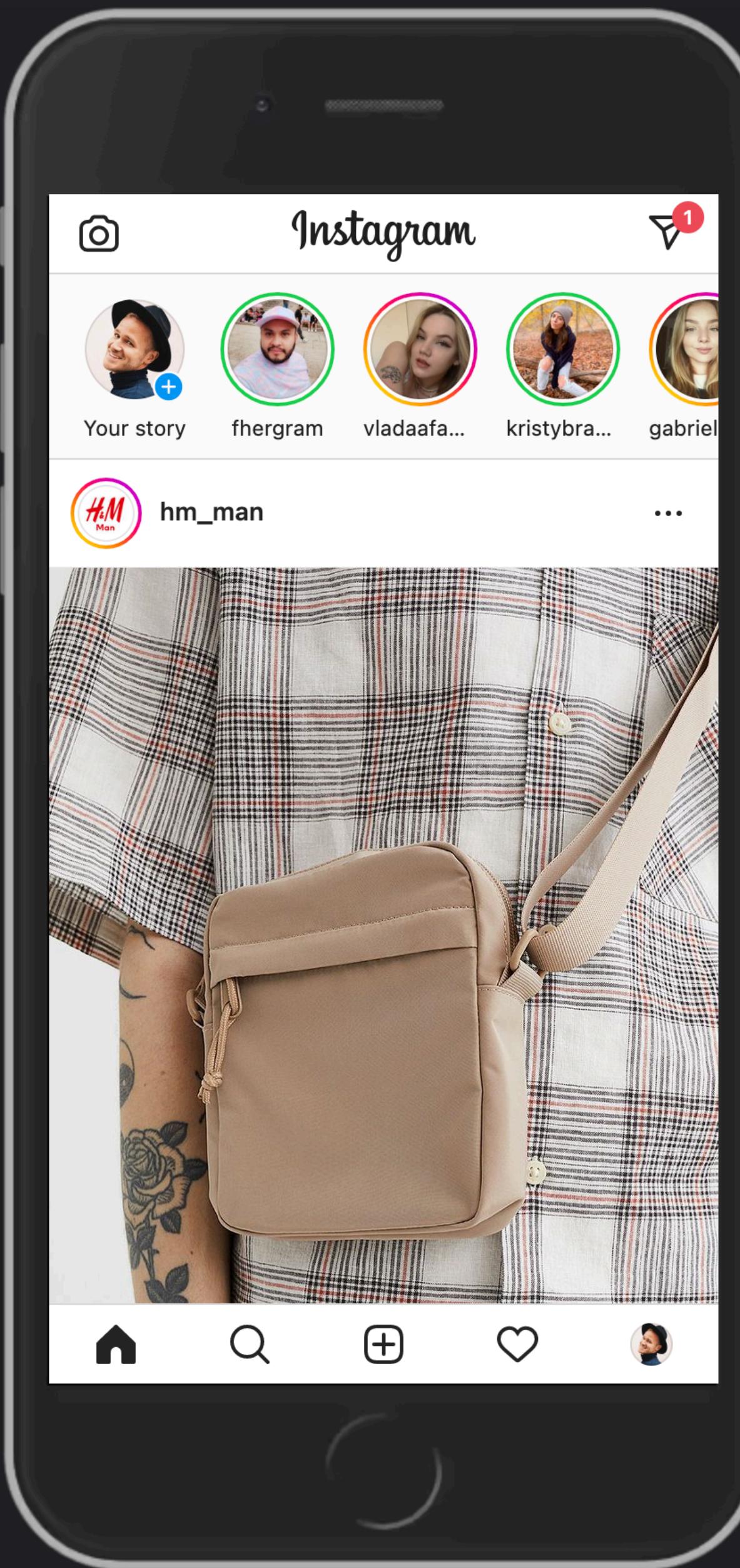
Create a new User

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

CREATE USER

Update User

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



Objects? Arrays?

The screenshot shows the homepage of DR Nyheder. At the top, there are navigation links for NYHEDER, DRTV, and DR LYD. Below the navigation, there are six thumbnail cards for TV shows: DR1: Løvens Hule, DR3: Nationens stærkeste, P1: LSD kælderen, DR LYD: Annas Margrethe, DR3: Du fucker med de forkerte, and A Very British Scandal. Under these, a section titled "Seneste nyt" (Latest news) displays three news items: "EU klager over Kinas hårde kurs over for Litauen" (5 MIN. SIDEN), "Børn og skoleelever opfordres stadig til to ugentlige coronatest" (13 MIN. SIDEN), and "England skrætter størstedelen af coronarestriktionerne fra i dag" (25 MIN. SIDEN). The main content area features a large image of medical supplies (a mask, a thermometer, a syringe, and a bottle of hand sanitizer) against a blue background, with the text "15 lande bakker Danmark op: Danske soldater skal blive i Mali" overlaid. At the bottom, a red banner reads "Regeringen har meldt genåbning - men ikke".

The screenshot shows the "ALLE ERHVERVSAKADEMI-UDDANNELSER" (All Business Academy Programs) page. At the top, there are two navigation links: "ALLE UDDANNELSER" and "UDDANNELSER UD FRA INTERESSE". Below this, a grid of 12 program cards, each featuring a student's face and the program name. The programs are: AUTOMATIONSTEKNOLOG (Automation Technologist), BYGGEKOORDINATOR (Construction Coordinator), BYGGETEKNIKER (Building Technician), DATAMATIKER (Computer Science), DESIGNTEKNOLOG (Design Technologist), ENTREPRENØRSKAB OG DESIGN (Entrepreneurship and Design), EL-INSTALLATOR (Electrical Installer), ENERGITEKNOLOG (Energy Technologist), IT-TEKNOLOG (IT Technologist), KORT- OG LANDMÅLING (Surveying and Land Measurement), MULTIMEDIEDESIGNER (Multimedia Designer), and PRODUKTIONSTEKNOLOG (Production Technologist). Each card has a small arrow icon pointing to the right.

It's all objects &
arrays!

Data Types & Data Structures

Objects & Arrays

fetch(...)

HTTP Requests in
JavaScript.

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

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

fetch(...)

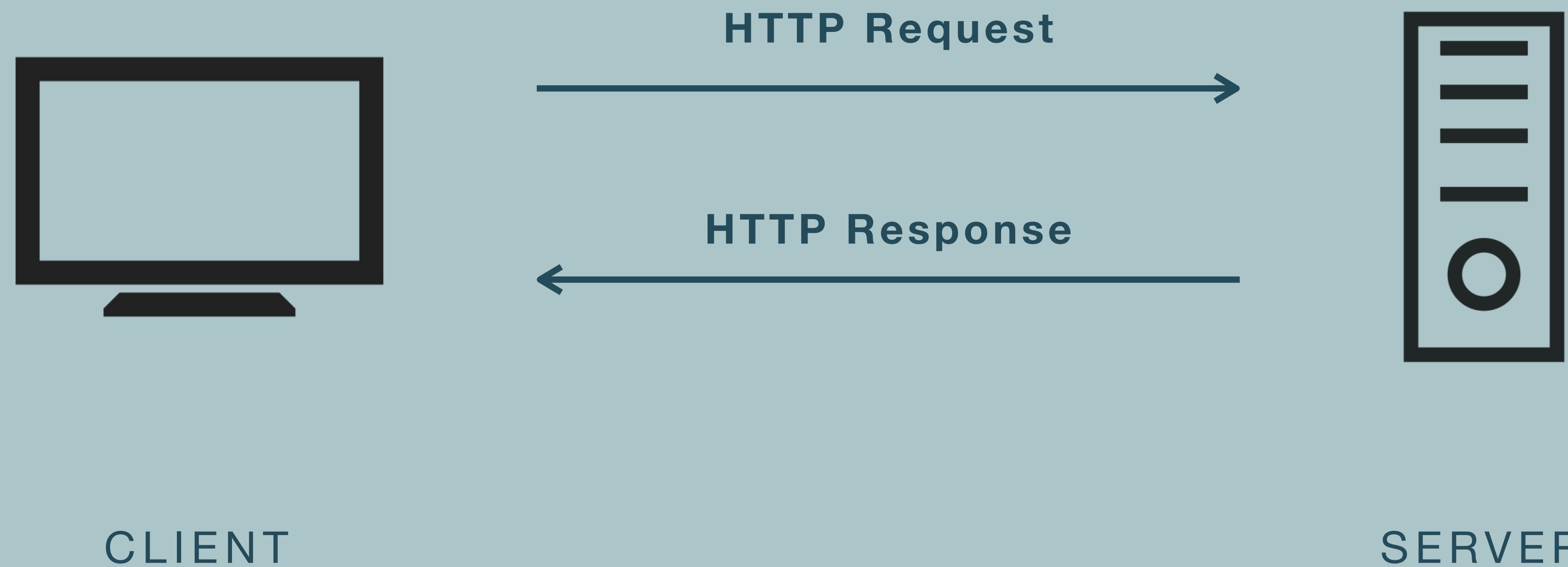
HTTP Requests in
JavaScript.

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

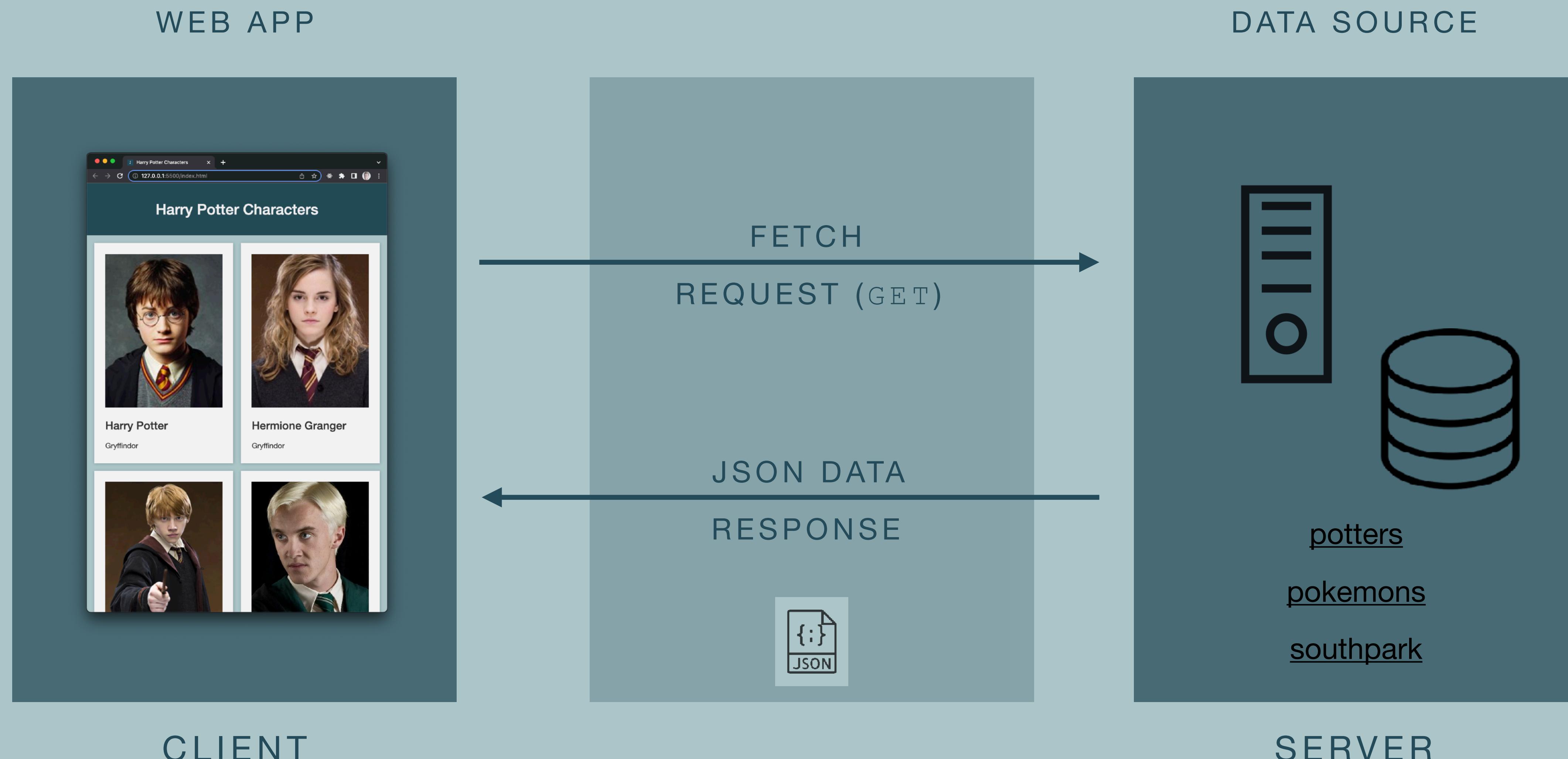
```
async function getCharacters() {  
  const response = await fetch(url);  
  const data = await response.json();  
  console.log(data);  
  return data;  
}
```

Client-Server Model

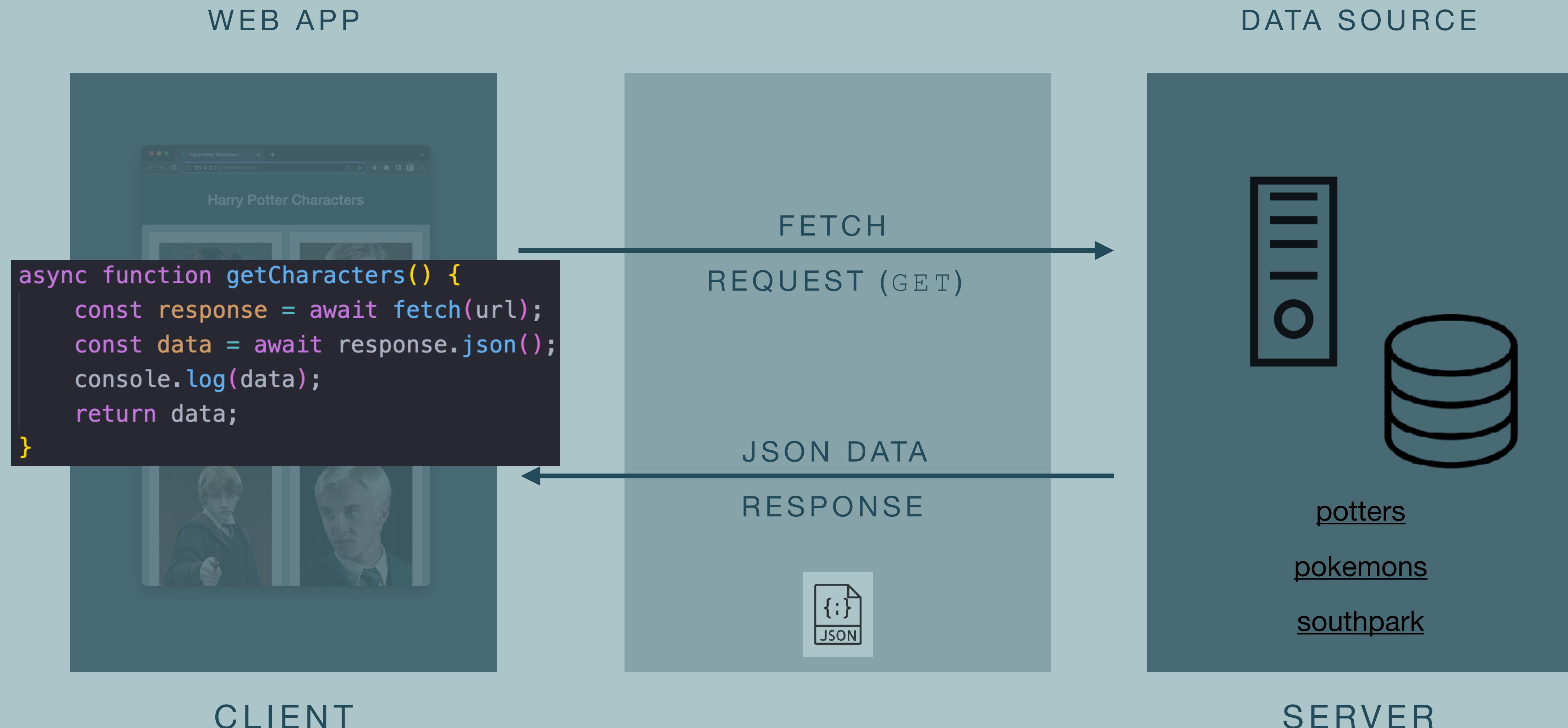
Communication between web **clients** and web **servers**.



Fetch, HTTP Request & Response



Fetch, HTTP Request & Response

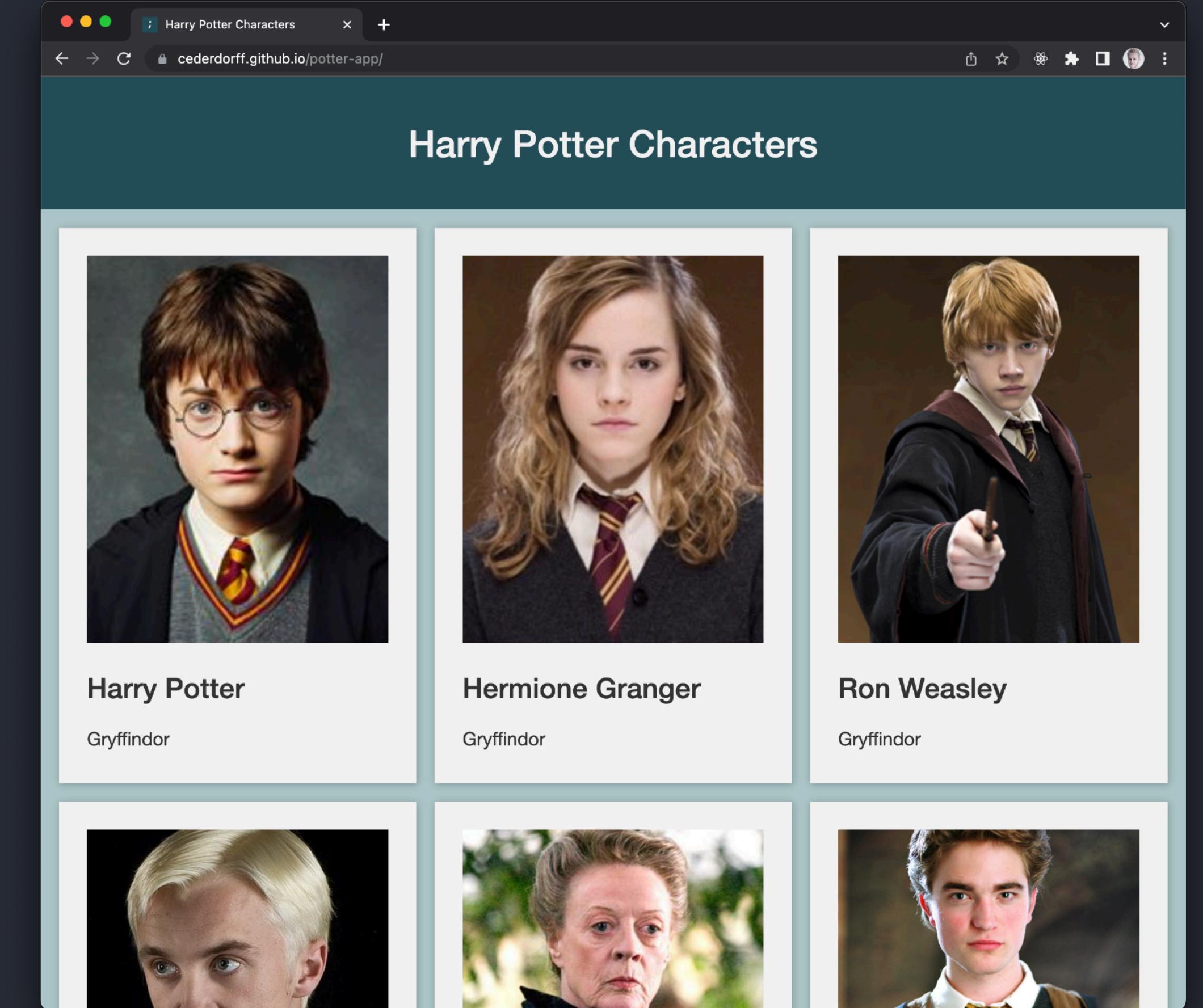


fetch(...)

Og Potter App

Løsning: <https://cederdorff.github.io/potter-app/>

Kode: <https://github.com/cederdorff/potter-app/tree/grid>



fetch(...) og “call stack”

1) initApp kaldes på load.

2) initApp kalder getCharacters og gemmer hvad der returneres i const'en characters.

3) getCharacters er en async funktion, da der anvendes await inde i funktionen. Funktionen fetcher data fra en given url. Data parses fra JSON til JavaScript-array, der gemmes i data-variablen. data returneres, så det kan læses hvor funktionen kaldes.

4) initApp kalder nu showCharacters med characters som argument (parameter). characters holder den data, som returneres fra getCharacters i 3).

5) showCharacters har characterList som parameter. characterList holder den data som blev sendt med fra initApp, dvs at det er samme data som lå i characters i initApp og tidligere i data i getCharacters.

showCharacters loop'er igennem characterList. For hver character (objekt) i characterList (array) kaldes showCharacter.

6) showCharacter tager en character (et objekt) som parameter og genererer HTML med værdier fra character-objektets properties. HTML'en tilføjes med insertAdjacentHTML.

```
app.js — potter-app
JS app.js
JS app.js > ...
4
5 window.addEventListener("load", initApp); 1)
6
7 async function initApp() {
8     const characters = await getCharacters(); 2)
9     showCharacters(characters); 4)
10 }
11
12 // ===== READ ===== //
13 // Read (GET) characters from json file located on GitHub
14 async function getCharacters() {
15     const response = await fetch("https://raw.githubusercontent.com/cederdorff/..."); 3)
16     const data = await response.json();
17     console.log(data);
18     return data;
19 }
20
21 // Create HTML and display all users from given list
22 function showCharacters(characterList) {
23     //loop through all users and create an article with content for each
24     for (const character of characterList) { 5)
25         showCharacter(character);
26     }
27 }
28
29 function showCharacter(character) {
30     document.querySelector("#characters").insertAdjacentHTML("beforeend", 6)
31     /*html*/
32     <article class="grid-item">
33         
34         <h2>${character.name}</h2>
35         <p>${character.house}</p>
36     </article>
37 }
```

Ln 1, Col 14 Spaces: 4 UTF-8 LF {} JavaScript ⚡ Port : 5500 ✓ Prettier ⚡ 🔔

fetch(...) og “call stack”

7) showCharacter tilføjer click-event til senest tilføjede article. Derfor article:last-child. Dette matcher med, at der i 6) anvendes “beforeend” som position til at indsætte.

8) Click-eventet er characterClicked, der er en nested function i showCharacter. Det er en nested function, da vi har brug for at kunne læse character og sende det med som argument (parameter) til showCharacterModal. På denne måde svarer character altid den karakter, der bliver klikket på.

9) showCharacterModal tager character som parameter. character er netop den vi klikker på. showCharacterModal bruger værdierne i character objektet til at foretage DOM-manipulationen og opdatere de eksisterende HTML-elementer.

10) I showCharacterModal kaldes generateDescription, der er en hjælpe funktion.

11) generateDescription genererer en beskrivelse på baggrund af character objektets properties hogwartsStaff, hogwartsStudent, alive og name. Beskrivelsen returneres tilbage til showCharacterModal og vises.

12) Når alle informationer i dialog er sat med DOM-manipulation og .textContent, vises dialog'en med metoden .showModal().

```
app.js — potter-app
JS app.js x
JS app.js > ...
29  function showCharacter(character) {
30      document.querySelector("#characters").insertAdjacentHTML(
31          "beforeend",
32          /*html*/
33          `

34              ![Character image](${character.image})
35              

## ${character.name}


36              

${character.house}


37

`
38      );
39  }
40
41  document.querySelector("#characters article:last-child").addEventListener("click", characterClicked);
42
43  function characterClicked() {
44      showCharacterModal(character);
45  }
46
47
48  function showCharacterModal(character) { 9)
49      console.log(character);
50      document.querySelector("#dialog-image").src = character.image;
51      document.querySelector("#dialog-title").textContent = character.name;
52      document.querySelector("#dialog-house").textContent = character.house;
53
54      // description
55      let description = generateDescription(character); 10)
56      document.querySelector("#dialog-character-description").textContent = description;
57
58      document.querySelector("#dialog-gender").textContent = character.gender;
59      document.querySelector("#dialog-birth-date").textContent = character.dateOfBirth;
60      document.querySelector("#dialog-eye-color").textContent = character.eyeColour;
61      document.querySelector("#dialog-hair-color").textContent = character.hairColour;
62      document.querySelector("#dialog-ancestry").textContent = character.ancestry;
63      document.querySelector("#dialog-species").textContent = character.species;
64
65      document.querySelector("#dialog-name").textContent = character.name;
66      document.querySelector("#dialog-actor-name").textContent = character.actor;
67
68      // show dialog
69      document.querySelector("#dialog-character").showModal(); 12
70  }
71
72  function generateDescription(character) { 11)
73      let description = "";
74      if (character.hogwartsStaff && character.alive) {
75          description = `${character.name} is employed at Hogwarts.`;
76      } else if (character.hogwartsStaff && !character.alive) {
77          description = `${character.name} was employed at Hogwarts but is no longer alive.`;
78      } else if (character.hogwartsStudent && character.alive) {
79          description = `${character.name} is a student at Hogwarts.`;
80      } else {
81          description = `${character.name} is not associated with Hogwarts.`;
82      }
83  }
```

JSON

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

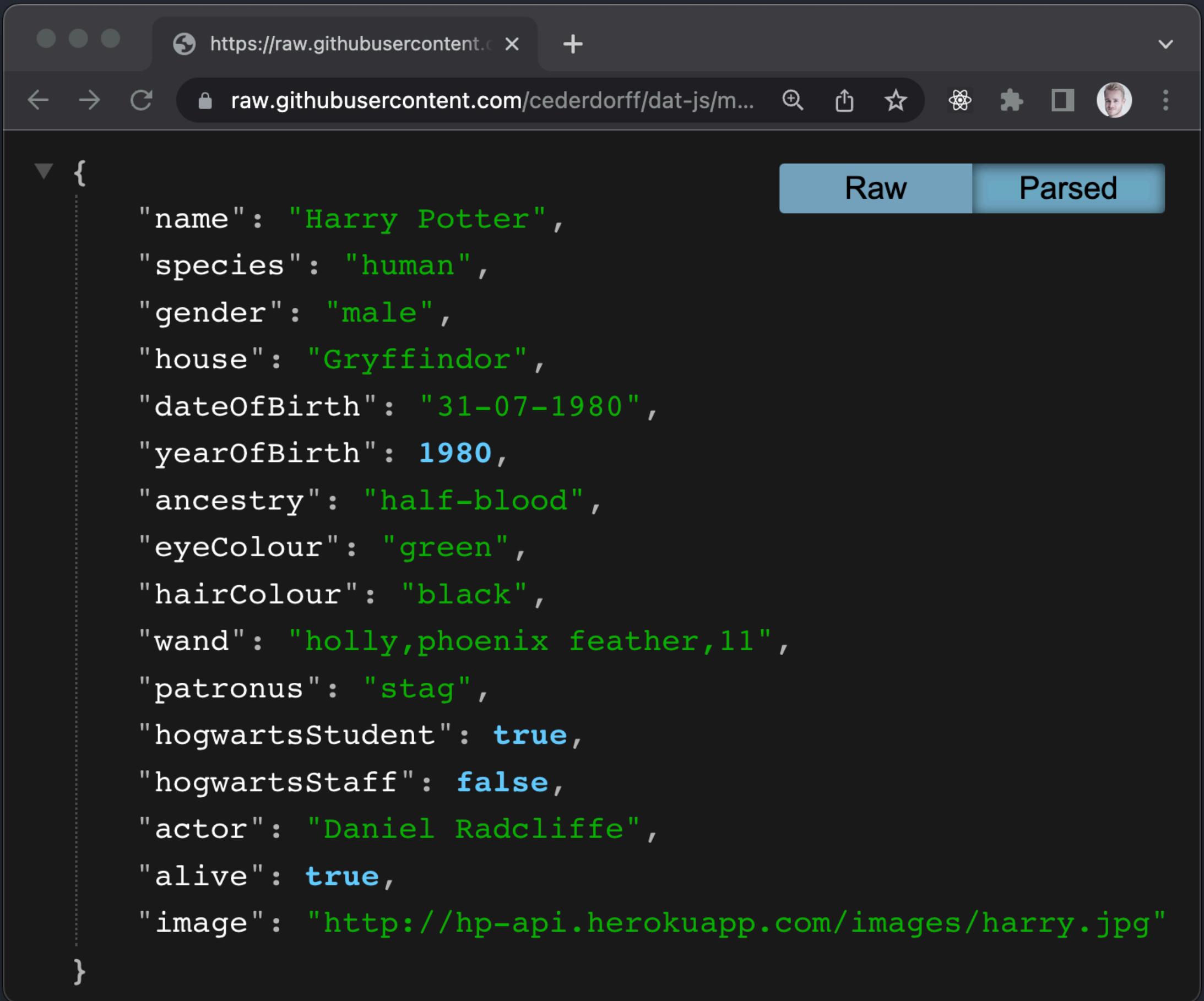
```
{  
  "name": "Alicia",  
  "age": 6  
}
```

JSON OBJECT

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

LIST OF JSON OBJECTS

JSON

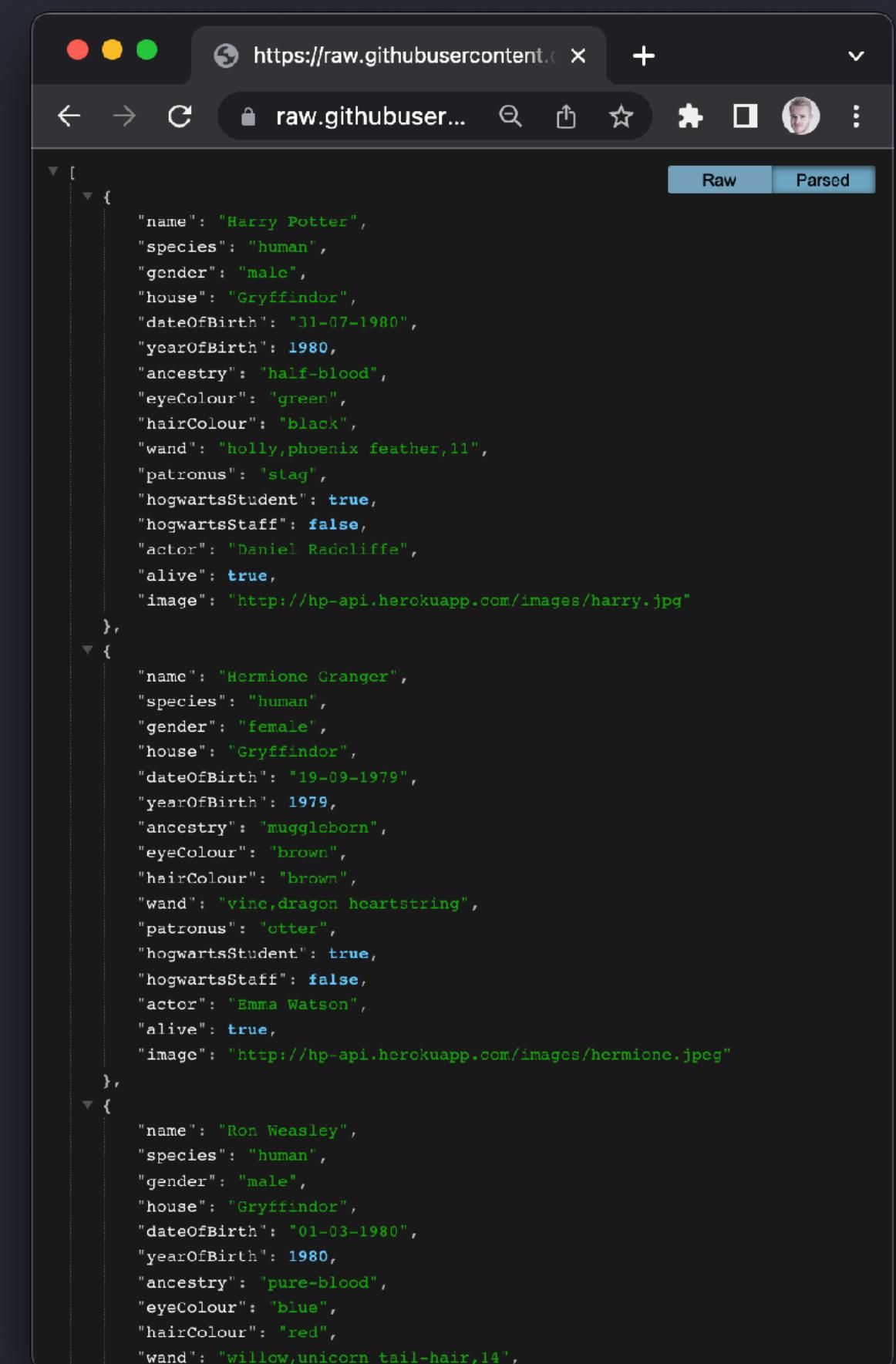


A screenshot of a web browser window displaying a single JSON object. The URL is <https://raw.githubusercontent.com/cederdorff/dat-js/master/data/harry.json>. The JSON structure is as follows:

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

The JSON is displayed in two tabs: "Raw" and "Parsed". The "Parsed" tab shows the JSON structure with color-coded keys and values.

JSON OBJECT



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

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

The JSON is displayed in two tabs: "Raw" and "Parsed". The "Parsed" tab shows the JSON structure with color-coded keys and values.

LIST OF JSON OBJECTS



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

JSON

The diagram illustrates the relationship between a piece of JavaScript code and its corresponding JSON data. On the left, a screenshot of a code editor shows a file named `app.js` with the following code:

```
28
29  function showCharacter(character) {
30      document.querySelector("#characters").insertAdjacentHTML(
31          "beforeend",
32          /*html*/
33          `

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


```

On the right, a screenshot of a web browser window displays a JSON object. A red arrow points from the `character.image` placeholder in the JavaScript code to the `"image": "http://hp-api.herokuapp.com/images/harry.jpg"` field in the JSON data. Another red arrow points from the `character.name` placeholder to the `"name": "Harry Potter"` field. The JSON data is as follows:

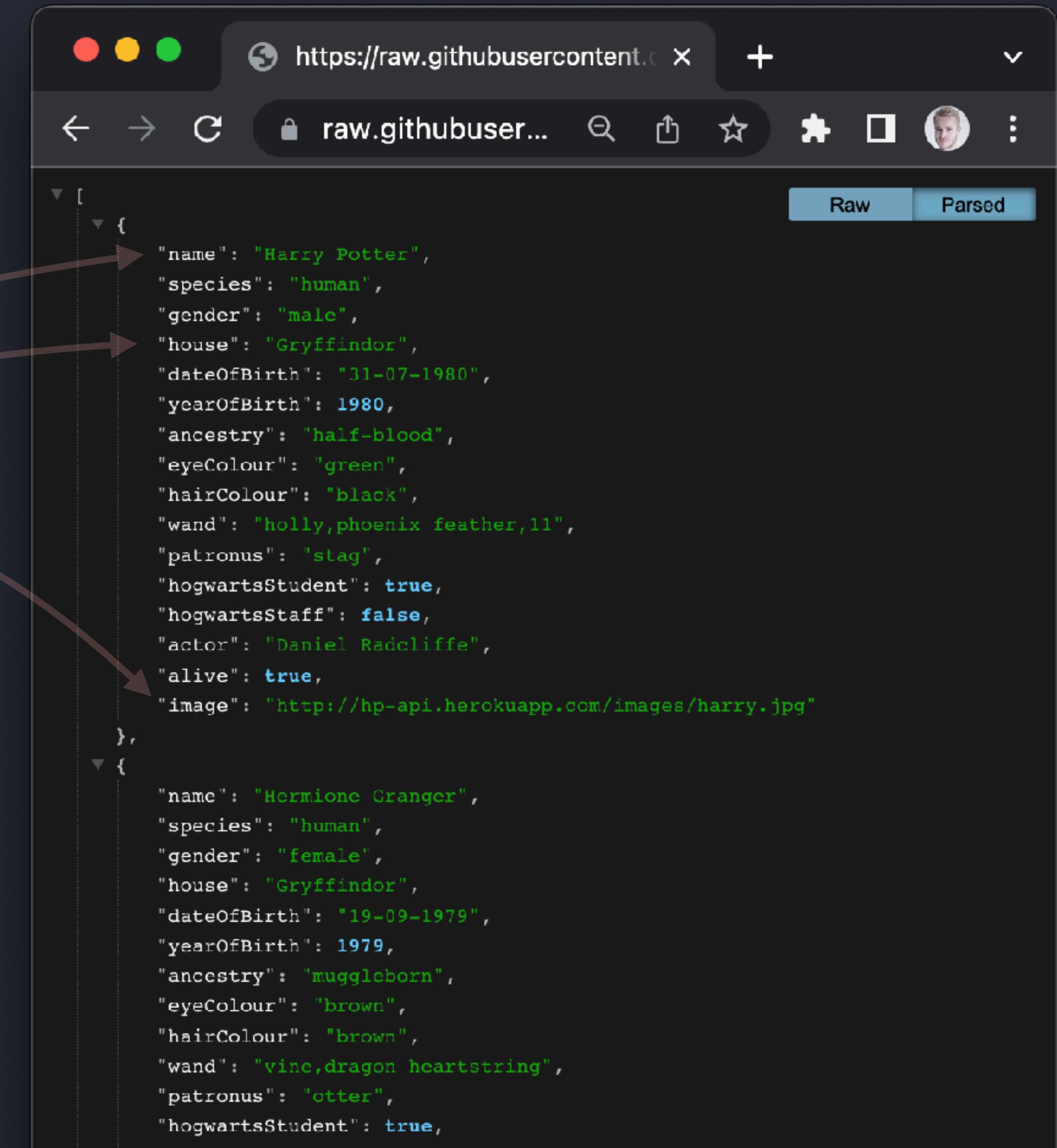
```
[{"name": "Harry Potter", "species": "human", "gender": "male", "house": "Gryffindor", "dateOfBirth": "31-07-1980", "yearOfBirth": 1980, "ancestry": "half-blood", "eyeColour": "green", "hairColour": "black", "wand": "holly,phoenix feather,11", "patronus": "stag", "hogwartsStudent": true, "hogwartsStaff":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

Review JSON-objekt

Det er vigtigt at vores properties matcher med det forventede! Det skal matche med den datamodel vi er blevet enige om: <https://cederdorff.github.io/dat-js/05-data/>

- Values - er der værdier i properties? Se hvilke [her](#).
 - Navngivning af properties - ja også store og små bogstaver  Se navngivning [her](#).
 - Er det den rigtige datatype? Brug dem vi er blevet enige om - se [her](#). Det må fx ikke være et array, hvis vi er blevet enige om en kommasepareret tekststreng.
 - Dobbeltjek properties og values matcher med: <https://cederdorff.github.io/dat-js/05-data/>
 - Din billed url. Er det et billede? Kontroller i browser. Du skal højeklikke på billede og vælge “Copy Image Address” eller “Copy Image Link” - ikke bare “Copy Link”.
 - Foretag tilpasninger af dit JSON-objekt (fra sidste uge), ikke noget med et array. Commit og push, så vi kan få de tilrettede objekter med.

- Values - er der værdier i properties? Se hvilke [her](#).
- Navngivning af properties - ja også store og små bogstaver. Se navngivning [her](#).
- Er det den rigtige datatype? Brug dem vi er blevet enige om - se [her](#). Det må fx ikke være et array, hvis vi er blevet enige om en kommaspareret tekststreng.
- Dobbeltjek properties og values matcher med: <https://cederdorff.github.io/dat-js/05-data/>
- Din billed url. Er det et billede? Kontroller i browser. Du skal højreklikke på billede og vælge "Copy Image Address" eller "Copy Image Link" - ikke bare "Copy Link"



Loops

Iterate over arrays or other iterable objects.

Ex loop through an array of objects.

```
// .forEach
characterList.forEach(showCharacter);

// for of loop
for (const character of characterList) {
    showCharacter(character);
}

// for loop
for (let index = 0; index < characterList.length; index++) {
    const character = characterList[index];
    showCharacter(character);
}
```

.forEach

an array method

```
const names = ["Peter", "Lenka", "Oskar", "Rasmus"];
names.forEach(showNames);

function showNames(name) {
  console.log(name);
}
```

You can use it on an array to do something for each element in the array.

forEach will call a function for each element in an array.

For of Loop

```
const names = ["Peter", "Lenka", "Oskar", "Rasmus"];
for (const name of names) {
  console.log(name);
}
```

You can use it to loop over an array to do something for every element.

for of loops through the values of an iterable object.

For of loop

iterate over arrays or other iterable objects

<https://scrimba.com/learn/introductiontojavascript/for-loops-cMMM8U9>

<https://scrimba.com/learn/introductiontojavascript/challenge-for-loops-cPkpJrcv>

For Loop

```
const names = ["Peter", "Lenka", "Oskar", "Rasmus"];

for (let index = 0; index < names.length; index++) {
  const name = names[index];
  console.log(name);
}
```

You can use it to loop over an array.

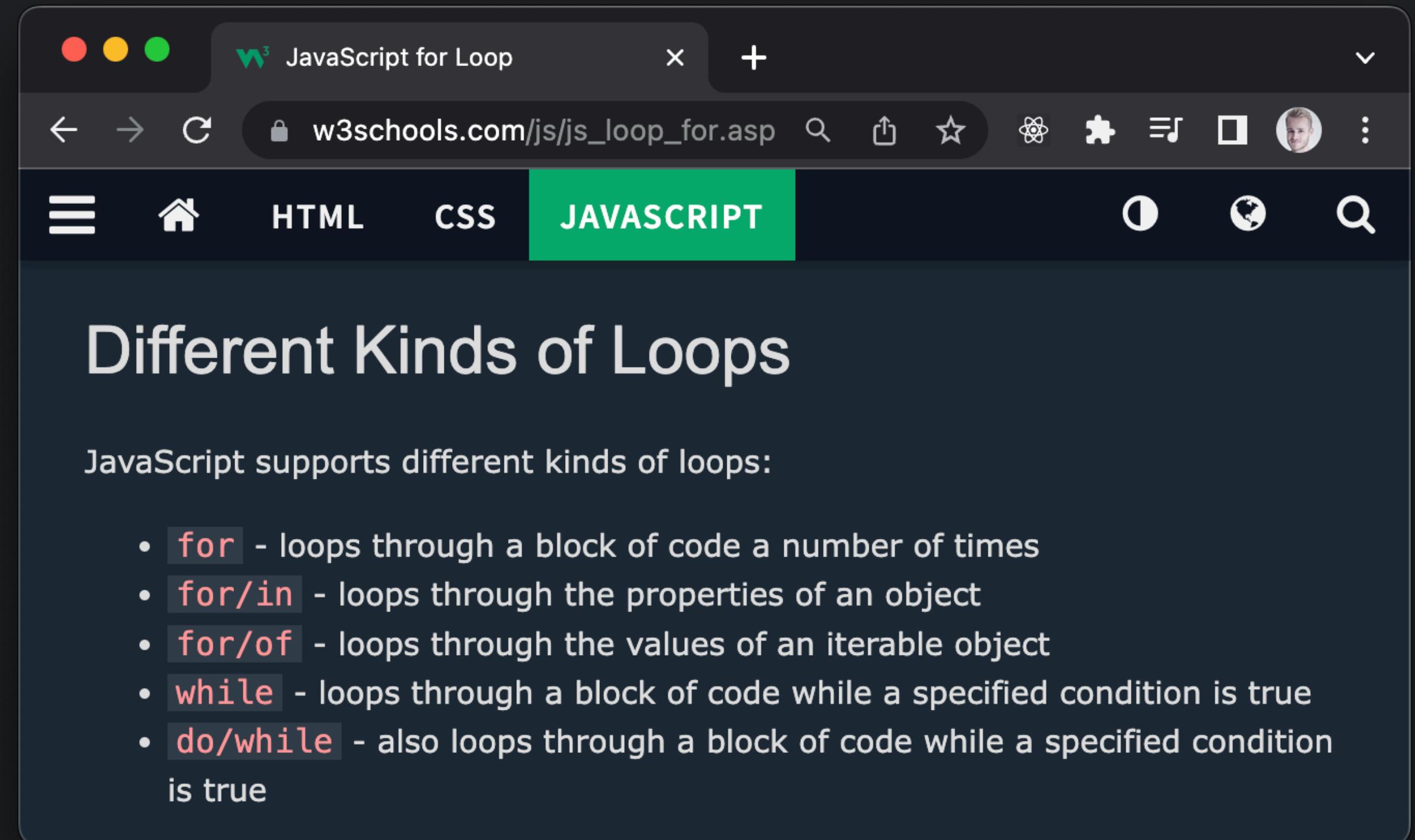
A for loop repeats until a specified condition is false.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#for_statement
<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for>
https://www.w3schools.com/js/js_loop_for.asp

Loops

Even more options 🎉

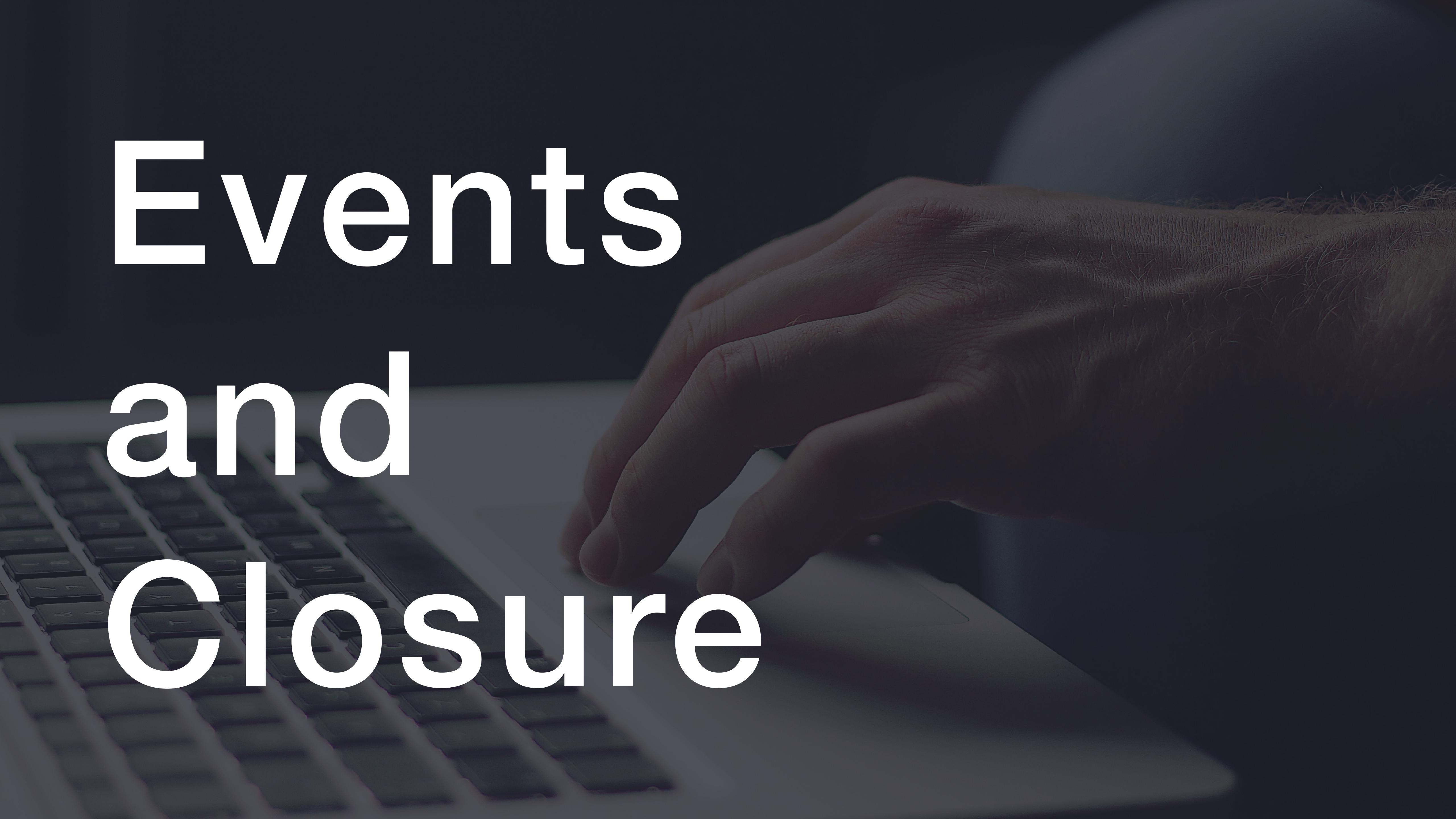
But use `forEach` or
`for of` for now.



JavaScript supports different kinds of loops:

- `for` - loops through a block of code a number of times
- `for/in` - loops through the properties of an object
- `for/of` - loops through the values of an iterable object
- `while` - loops through a block of code while a specified condition is true
- `do/while` - also loops through a block of code while a specified condition is true

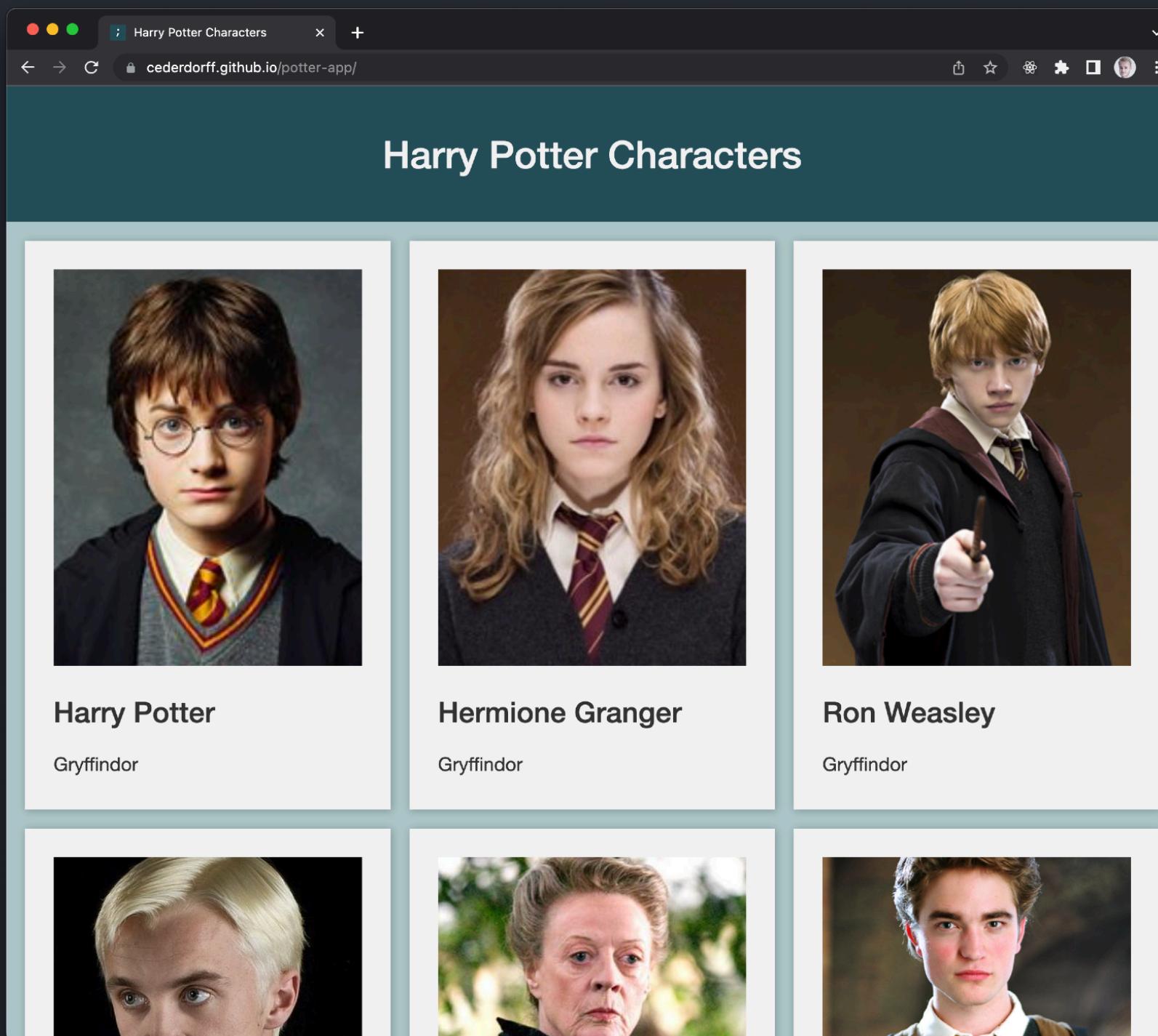
https://www.w3schools.com/js/js_loop_for.asp



Events
and
closure

EventListener

The eventListener attaches an event handler to the specified element

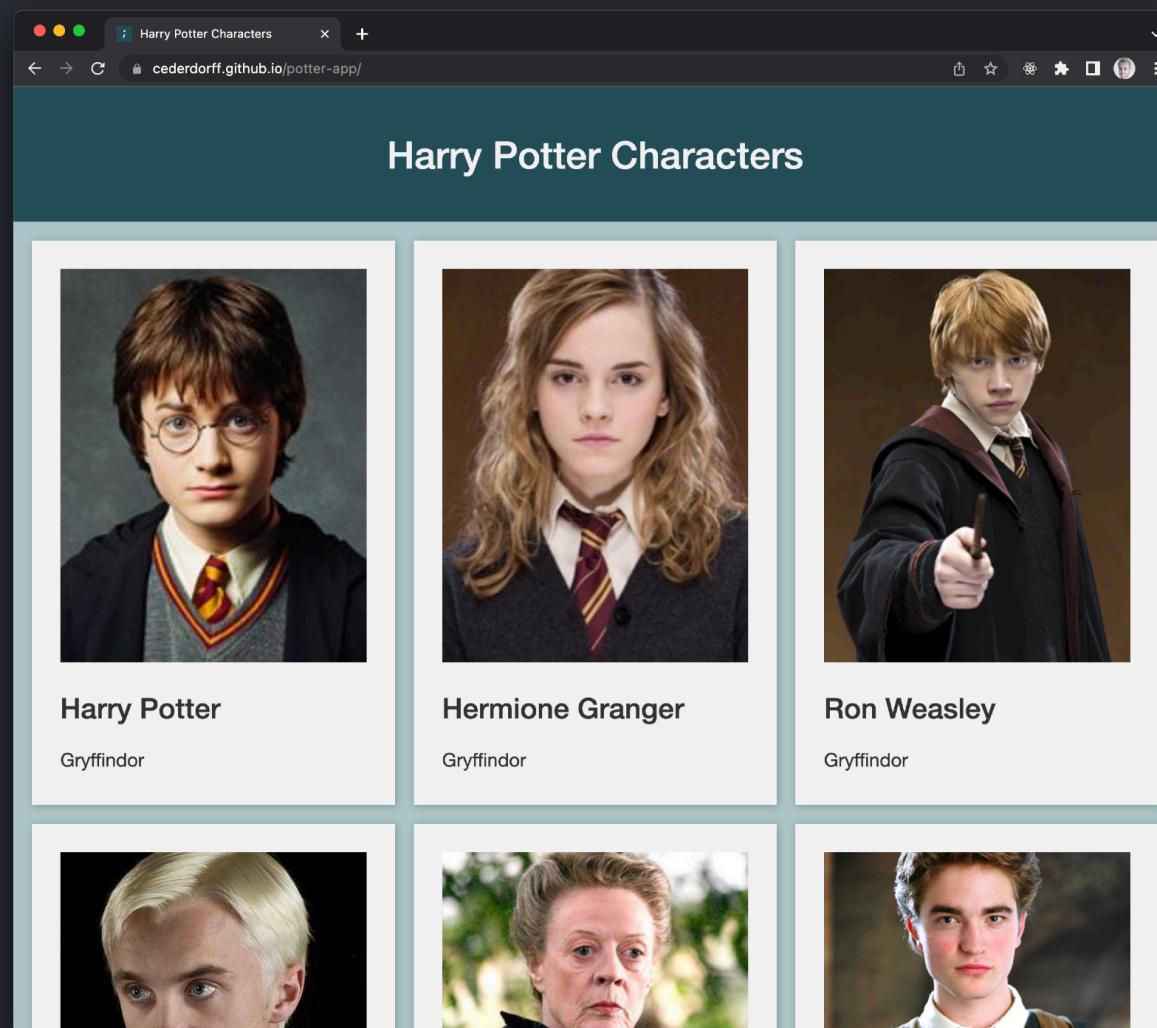


```
document.querySelector("#characters article:last-child")
    .addEventListener("click", characterClicked);
```

```
function characterClicked() {
    // do something
}
```

EventListener

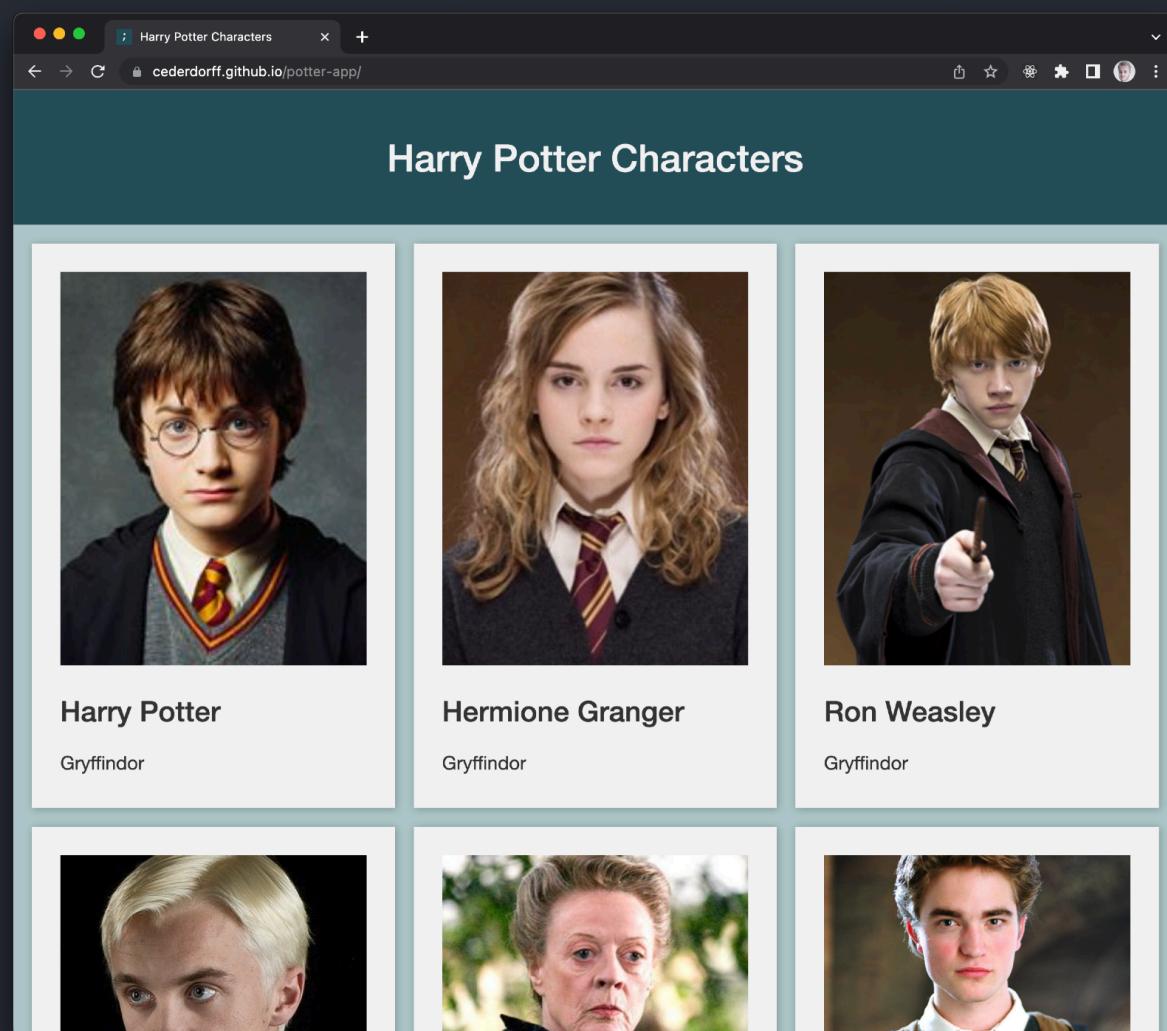
Vi tilføjer event for hver article som vi indsætter i DOM'en



```
function showCharacter(character) {  
    document.querySelector("#characters").insertAdjacentHTML(  
        "beforeend",  
        /*html*/ `'  
            <article class="grid-item">  
                  
                <h2>${character.name}</h2>  
                <p>${character.house}</p>  
            </article>  
        );  
  
    document.querySelector("#characters article:last-child").addEventListener("click", characterClicked);  
}
```

Nested functions & closure

Men vi skal kende character, for at kunne vise
lige præcis den character vi klikker på.



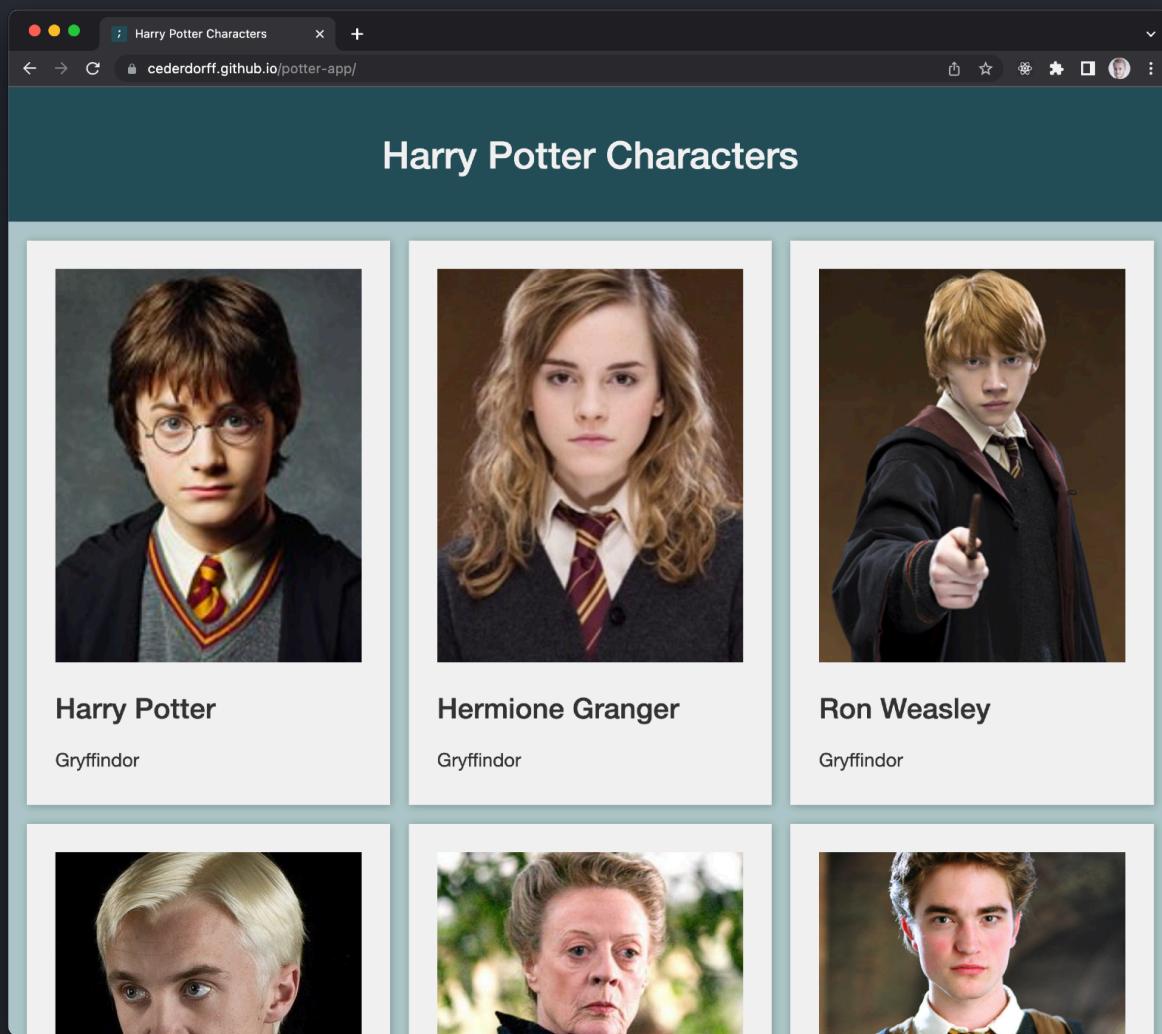
```
function showCharacter(character) {
  document.querySelector("#characters").insertAdjacentHTML(
    "beforeend",
    /*html*/
    `<article class="grid-item">
      
      <h2>${character.name}</h2>
      <p>${character.house}</p>
    </article>
  );
}

document.querySelector("#characters article:last-child").addEventListener("click", characterClicked);

function characterClicked() {
  showCharacterModal(character);
}
```

Nested functions & closure

Derfor har vi en funktion i en funktion. Så vil characterClicked (nested function) have adgang til showCharacter's scope, og der ved kan vi læse character



```
function showCharacter(character) {
  document.querySelector("#characters").insertAdjacentHTML(
    "beforeend",
    /*html*/
    `


      <h2>${character.name}</h2>
      <p>${character.house}</p>
    </article>
  );
}

document.querySelector("#characters article:last-child").addEventListener("click", characterClicked);

function characterClicked() {
  showCharacterModal(character);
}

}


```

Nested function

A function inside a function.

All functions have access to the global scope.

Nested functions have access to the scope "above" them

Closure

A closure is a function that have access to the parent scope - also after the parent function has closed / has been executed

```
function showCharacter(character) {  
    document.querySelector("#characters").insertAdjacentHTML(  
        "beforeend",  
        /*html*/ `'  
            <article class="grid-item">  
                  
                <h2>${character.name}</h2>  
                <p>${character.house}</p>  
            </article>  
        );  
  
    document.querySelector("#characters article:last-child").addE  
};  
function characterClicked() {  
    showCharacterModal(character);  
}  
}
```

DOM Manipulation

“

When you use JavaScript to
add, remove, and modify
elements of a website

”

- document.querySelector(selector)
- document.querySelectorAll(name)
- document.createElement(name)
- parentNode.appendChild(node)
- element.append(node, node, ...)
- element.insertAdjacentHTML(position, html)
- element.innerHTML
- element.textContent
- element.style.xxx
- element.setAttribute()
- element.getAttribute()
- element.addEventListener()
- element.classList
- window.content
- Window.onload
- window.scrollTo()

Methods for DOM Manipulation

Common Methods

```
// .createElement and .appendChild
const newTitle = document.createElement("h1");
newTitle.textContent = "Hi!";
document.querySelector("body").appendChild(newTitle);

// backticks and .insertAdjacentHTML
const secondTitle = /*html*/ `<h1>Hi, again!</h1>`;
document.querySelector("body").insertAdjacentHTML(secondTitle);

// .textContent - must have defined an h1 element in your HTML
const thirdTitle = document.querySelector("h1");
thirdTitle.textContent = "Hi, again, again!";
```

Vi har anvendt
... i hvertfald for nu

DOM Manipulation

```
let fullName = "Peter Lind";
console.log(fullName);
document.querySelector("#fullName_container").textContent = fullName;
```

.textContent

Get or set the text content of a given (HTML) element

.textContent

Get or set the text content of a given (HTML) element. The element must be in the DOM.

```
let fullName = "Peter Lind";
console.log(fullName);
document.querySelector("#fullName_container").textContent = fullName;
```

```
<body>
  <header>
    <h1 id="fullName_container"></h1>
  </header>
  <script src="app.js"></script>
</body>
```



.textContent

Get or set the text content of a given (HTML) element. The element must be in the DOM.

The diagram illustrates the mapping between the code in `app.js` and the corresponding DOM structure in `index.html` using pink arrows. The code in `app.js` sets the `textContent` property of various elements to the values defined in the `character` object. The DOM structure in `index.html` contains these elements with their respective IDs and `textContent` properties.

```
JS app.js
47
48 function showCharacterModal(character) {
49     console.log(character);
50     document.querySelector("#dialog-image").src = character.image;
51     document.querySelector("#dialog-title").textContent = character.name;
52     document.querySelector("#dialog-house").textContent = character.house;
53
54     // description
55     let description = generateDescription(character);
56     document.querySelector("#dialog-character-description").textContent = description;
57
58     document.querySelector("#dialog-gender").textContent = character.gender;
59     document.querySelector("#dialog-birth-date").textContent = character.dateOfBirth;
60     document.querySelector("#dialog-eye-color").textContent = character.eyeColour;
61     document.querySelector("#dialog-hair-color").textContent = character.hairColour;
62     document.querySelector("#dialog-ancestry").textContent = character.ancestry;
63     document.querySelector("#dialog-species").textContent = character.species;
64
65     document.querySelector("#dialog-name").textContent = character.name;
66     document.querySelector("#dialog-actor-name").textContent = character.actor;
67
68     // show dialog
69     document.querySelector("#dialog-character").showModal();
70 }
```

```
index.html — potter-app
...
21 <dialog id="dialog-character">
22     <section class="dialog-grid">
23         <figure></h2>
27             <h3 id="dialog-house"></h3>
28             <p id="dialog-character-description"></p>
29
30             <section>
31                 <h4>Personal Info</h4>
32                 <ul>
33                     <li>Gender: <span id="dialog-gender"></span></li>
34                     <li>Birth Date: <span id="dialog-birth-date"></span></li>
35                     <li>Eye Color: <span id="dialog-eye-color"></span></li>
36                     <li>Hair Color: <span id="dialog-hair-color"></span></li>
37                     <li>Ancestry: <span id="dialog-ancestry"></span></li>
38                     <li>Species: <span id="dialog-species"></span></li>
39
40             </section>
41
42             <section>
43                 <h4>Actor</h4>
44                 <p><span id="dialog-name">Character</span> is played by <span id=
45             </section>
46
47         </article>
48     </section>
49 </dialog>
```

Arrows point from the `character` object properties in `app.js` to the corresponding `textContent` assignments in the `index.html` DOM. For example, the arrow from `character.name` points to the assignment in line 51 of `app.js`, which corresponds to the `<h3 id="dialog-house"></h3>` element in the `index.html` DOM. Similar mappings are shown for other properties like `image`, `house`, `gender`, `dateOfBirth`, etc.

document.createElement()

```
// create a new h1 element
const newTitle = document.createElement("h1");

// and give it some content
newTitle.textContent = "Hi there and greetings!";

// add the newly created element into the DOM
document.querySelector("body").appendChild(newTitle);
```

.createElement

Creates an HTML element specified by tagName.

.appendChild

Appends an element as last child of an element.

insertAdjacentHTML(position, html)

```
// 1

// create a new h1 with content using a backtick
const newTitle = /*html*/ `<h1>Hi there and greetings!</h1>`;

// add the newly created element into the DOM
document.querySelector("body").insertAdjacentHTML("beforeend", newTitle);

// 2

// variable holding username
const username = "RACE";

// create a new p with content using a backtick and username variable
const newParagraph = /*html*/ `<p>Welcome, ${username}!</p>`;

// add the newly created element into the DOM
document.querySelector("body").insertAdjacentHTML("beforeend", newParagraph);
```

Backtick string

Create multiline strings (templates) with embedded expressions and tags.

.insertAdjacentHTML

Inserts HTML into a specified position.

showCharacter(character)

```
function showCharacter(character) {  
    document.querySelector("#characters").insertAdjacentHTML(  
        "beforeend",  
        /*html*/ `   
            <article class="grid-item">  
                  
                <h2>${character.name}</h2>  
                <p>${character.house}</p>  
            </article>  
        ` );  
}
```

Backtick string

Create multiline strings (templates) with embedded expressions and tags.

.insertAdjacentHTML

Inserts HTML into a specified position.

Hvorfor ikke .innerHTML?

```
function showCharacter(character) {  
  document.querySelector("#characters").innerHTML = /*html*/`  
    <article class="grid-item">  
        
      <h2>${character.name}</h2>  
      <p>${character.house}</p>  
    </article>  
`;  
}
```

Backtick string

Create multiline strings (templates) with embedded expressions and tags.

.innerHTML

... sets or returns the HTML content (inner HTML) of an element.

“To insert the HTML into the document rather than replace the contents of an element, use the method `insertAdjacentHTML()`.”

.innerHTML & events

```
function showCharacter(character) {  
  document.querySelector("#characters").innerHTML = /*html*/ `   
    <article class="grid-item">  
        
      <h2>${character.name}</h2>  
      <p>${character.house}</p>  
    </article>  
  `;  
  
  document.querySelector("#characters article:last-child").addEventListener("click", characterClicked);  
  
  function characterClicked() {  
    showCharacterModal(character);  
  }  
}
```

Backtick string

Create multiline strings (templates) with embedded expressions and tags.

.innerHTML

“Please note that using `innerHTML` to append HTML elements [...] will result in the removal of any previously set event listeners. That is, after you append any HTML element that way you won't be able to listen to the previously set event listeners.”

Which one?

```
// backticks and .insertAdjacentHTML
const secondTitle = /*html*/ `<h1>Hi, again!</h1>`;
document.querySelector("body").insertAdjacentHTML(secondTitle);

// .textContent - must have defined an h1 element in your HTML
const thirdTitle = document.querySelector("h1");
thirdTitle.textContent = "Hi, again, again!";
```

.insertAdjacentHTML

Inserts HTML into a specified position.

.textContent

Get or set the text content of a given (HTML) element

Which one?

.insertAdjacentHTML

Inserts HTML into a specified position.

```
function showCharacter(character) {
  document.querySelector("#characters").insertAdjacentHTML(
    "beforeend",
    /*html*/
    `<article class="grid-item">
      
      <h2>${character.name}</h2>
      <p>${character.house}</p>
    </article>
  `;
}
//...
```

.textContent

Get or set the text content of a given (HTML) element

```
function showCharacterModal(character) {
  console.log(character);
  document.querySelector("#dialog-image").src = character.image;
  document.querySelector("#dialog-title").textContent = character.name;
  document.querySelector("#dialog-house").textContent = character.house;

  // description
  let description = generateDescription(character);
  document.querySelector("#dialog-character-description").textContent = description;

  document.querySelector("#dialog-gender").textContent = character.gender;
  document.querySelector("#dialog-birth-date").textContent = character.dateOfBirth;
  document.querySelector("#dialog-eye-color").textContent = character.eyeColour;
  document.querySelector("#dialog-hair-color").textContent = character.hairColour;
  document.querySelector("#dialog-ancestry").textContent = character.ancestry;
  document.querySelector("#dialog-species").textContent = character.species;

  document.querySelector("#dialog-name").textContent = character.name;
  document.querySelector("#dialog-actor-name").textContent = character.actor;

  // show dialog
  document.querySelector("#dialog-character").showModal();
}
```

Specialisering af output

“

Hmm, det handler vel om at vise
det “rigtige”, og det der giver
bedst mening for brugeren.

”

Hvis du nu ikke kendte til programmering?

A screenshot of a web browser window titled "Harry Potter Characters". The URL in the address bar is "127.0.0.1:5500/index.html". The main content is a modal dialog for "Hermione Granger". It features a large image of Emma Watson as Hermione. The title "Hermione Granger" is at the top, followed by her house "Gryffindor". A short bio states "Hermione Granger is a student at Hogwarts". The "Personal Info" section lists the following attributes:

- Gender: female
- Birth Date: 19-09-1979
- Eye Color: brown
- Hair Color: brown
- Ancestry: muggleborn
- Species: human

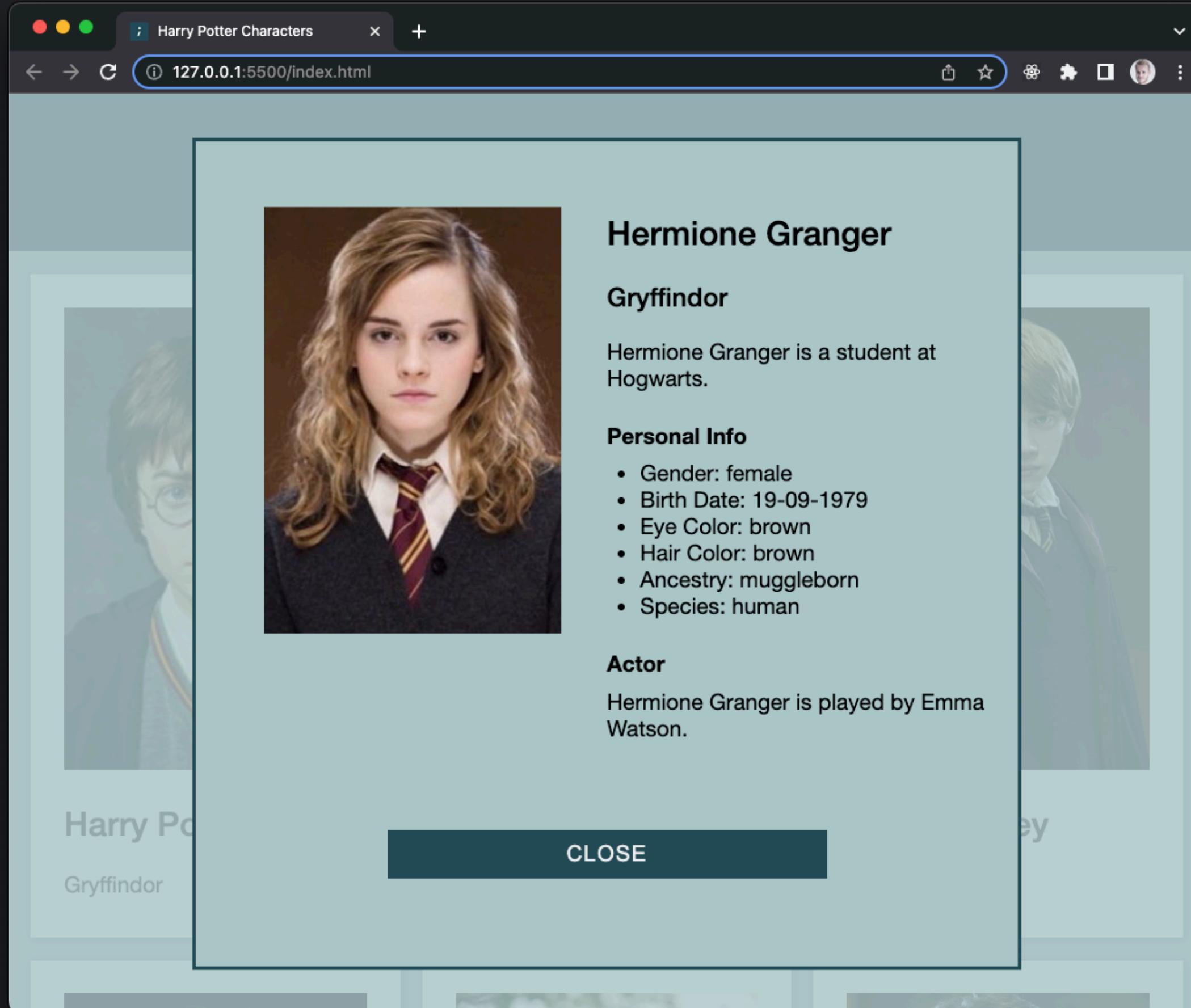
The "Actor" section notes that "Hermione Granger is played by Emma Watson". A "CLOSE" button is at the bottom of the modal.

A screenshot of a web browser window titled "Harry Potter Characters". The URL in the address bar is "127.0.0.1:5500/index.html". The main content is a modal dialog for "Hermione Granger". It features a large image of Emma Watson as Hermione. The title "Hermione Granger" is at the top, followed by her house "Gryffindor". A short bio states "Hermione Granger is a student at Hogwarts". The "Personal Info" section displays the following JSON data:

- Hogwarts Student: true
- Hogwarts Staff: false
- Is alive: true

The "Actor" section notes that "Hermione Granger is played by Emma Watson". A "CLOSE" button is at the bottom of the modal.

Forbedringer



- Er der noget, vi gerne vil vise på en særlig måde, illustrerer, tilpasse osv?

Specialisering af styling

Hvad nu, hvis du vil differentiere stylingen af dine grid-elementer eller dialog?



Code
Every
Day