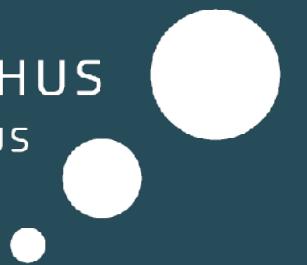


<https://www.instagram.com/p/CVqbCzgsZUF/>

DATA & ASYNC JAVASCRIPT

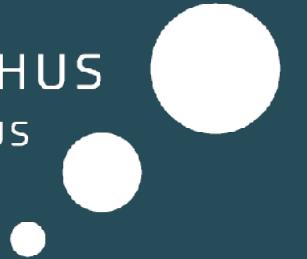
FRONTEND DEVELOPER

ERHVERVSAKADEMI AARHUS
BUSINESS ACADEMY AARHUS



PURPOSE

Gain advanced knowledge of fetch and data structures. Furthermore, to develop a CRUD SPA with [JSONBIN.io](#) and/or PHP Backend Service.



JSON & DATA STRUCTURES

WRAP UP: CANVAS USERS SPA

ASYNC JS: FETCH, HEADERS, HTTP METHODS, BODY & API

CASE: DATING SPA

10.07: SPA & ROUTER

11.21: FILTER, SORT & SEARCH

12.30: 2ND SEMESTER &
ELECTIVES

A screenshot of a MacBook Air screen. The top half shows a terminal window with a file tree on the left and some command-line output on the right. The file tree includes 'index.html', 'fetch.js', 'fetch_json_data', 'fetch_persons_img', 'fetch_weather_data', 'fetch_wp', 'main.css', 'main.js', 'index.html', 'fetch_wp_act', 'fetch_wp_act_products', 'main.css', 'main.js', 'index.html', 'fetch_wp_cederdorff.com', 'main.css', 'fonts', and 'img'. The bottom half shows a code editor with a large amount of JavaScript code. The code is a single file named 'fetch.js' located in the 'main.js' folder. It contains functions for loading JSON data from various sources and appending it to the DOM. The code uses fetch(), then(), and catch() methods, as well as template literals and document.querySelector(). The code editor has a dark theme with syntax highlighting.

```
trap_template
materializecss_fir
materializecss_fir
materializecss_sti
ch
ch_ajax
css
img
js
json
index.html
fetch.js
fetch_json_data
fetch_persons_img
fetch_weather_data
fetch_wp
css
in
main.js
index.html
fetch_wp_act
fetch_wp_act_products
main.css
in
main.js
index.html
fetch_wp_cederdorff.com
main.css
fonts
img
fetch_js/main.js ① 0 ▲ 0 ① 0 1:1
```

```
39 <a href='mailto:${person.mail}'>
40 </article>
41 `;
42 }
43 // appends htmlTemplate depending on given type
44 if (type === "familyMembers") {
45   document.querySelector("#family-members").innerHTML = htmlTemplate;
46 } else if (type === "teachers") {
47   document.querySelector("#teachers").innerHTML = htmlTemplate;
48 }
49 }
50 }
51 /*
52  * loads and appends json post data to the DOM
53 */
54 */
55 function loadPosts() {
56   fetch('http://jsonplaceholder.typicode.com/posts')
57     .then(function(response) {
58       return response.json();
59     })
60     .then(function(posts) {
61       console.log(posts);
62       let htmlTemplate = "";
63       for (let post of posts) {
64         htmlTemplate += `
65           <article>
66             <h4>${post.title}</h4>
67             <p>${post.body}</p>
68           </article>
69         `;
70       }
71       document.querySelector("#posts").innerHTML = htmlTemplate;
72     });
73 }
74 }
```

AGENDA

MacBook Air

Uge 44				
	Mandag d. 01 - 11	Tirsdag d. 02 - 11	Onsdag d. 03 - 11	Torsdag d. 04 - 11
08:30 - 10:00	Frontend programming RACE eaa-R104-1.16	Backend programming KATO eaa-R104-S.30		Interaction and experience design SBJ eaa-R104-S.30
10:30 - 12:00	Frontend programming RACE eaa-R104-1.16	Backend programming KATO eaa-R104-S.30		Interaction and experience design SBJ eaa-R104-S.30
12:30 - 14:00	Frontend programming RACE eaa-R104-1.16	Backend programming KATO eaa-R104-S.30		Interaction and experience design SBJ eaa-R104-S.30
14:30 - 16:00				

Uge 45				
	Mandag d. 08 - 11	Tirsdag d. 09 - 11	Onsdag d. 10 - 11	Torsdag d. 11 - 11
08:30 - 10:00	Frontend programming RACE eaa-R104-1.16	Backend programming KATO eaa-R104-S.30		Interaction and experience design SBJ eaa-R104-S.30
10:30 - 12:00	Frontend programming RACE eaa-R104-1.16	Backend programming KATO eaa-R104-S.30		Interaction and experience design SBJ eaa-R104-S.30
12:30 - 14:00	Frontend programming RACE eaa-R104-1.16	Backend programming KATO eaa-R104-S.30		Interaction and experience design SBJ eaa-R104-S.30
14:30 - 16:00				

Uge 46				
	Mandag d. 15 - 11	Tirsdag d. 16 - 11	Onsdag d. 17 - 11	Torsdag d. 18 - 11
08:30 - 10:00	Frontend programming RACE eaa-R104-1.16	Backend programming KATO eaa-R104-S.30		Interaction and experience design SBJ eaa-R104-S.30
10:30 - 12:00	Frontend programming RACE eaa-R104-1.16	Backend programming KATO eaa-R104-S.30		Interaction and experience design SBJ eaa-R104-S.30
12:30 - 14:00	Frontend programming RACE eaa-R104-1.16	Backend programming KATO eaa-R104-S.30		Interaction and experience design SBJ eaa-R104-S.30
14:30 - 16:00				

Uge 47				
	Mandag d. 22 - 11	Tirsdag d. 23 - 11	Onsdag d. 24 - 11	Torsdag d. 25 - 11
08:30 - 10:00	Frontend programming RACE eaa-R104-1.16	Backend programming KATO eaa-R104-S.30		Interaction and experience design SBJ eaa-R104-S.30
10:30 - 12:00	Frontend programming RACE eaa-R104-1.16	Backend programming KATO eaa-R104-S.30		Interaction and experience design SBJ eaa-R104-S.30
12:30 - 14:00	Frontend programming RACE eaa-R104-1.16	Backend programming KATO eaa-R104-S.30		Interaction and experience design SBJ eaa-R104-S.30
14:30 - 16:00				Interdisciplinary project RACE SBJ eaa-R104-S.30

BACK

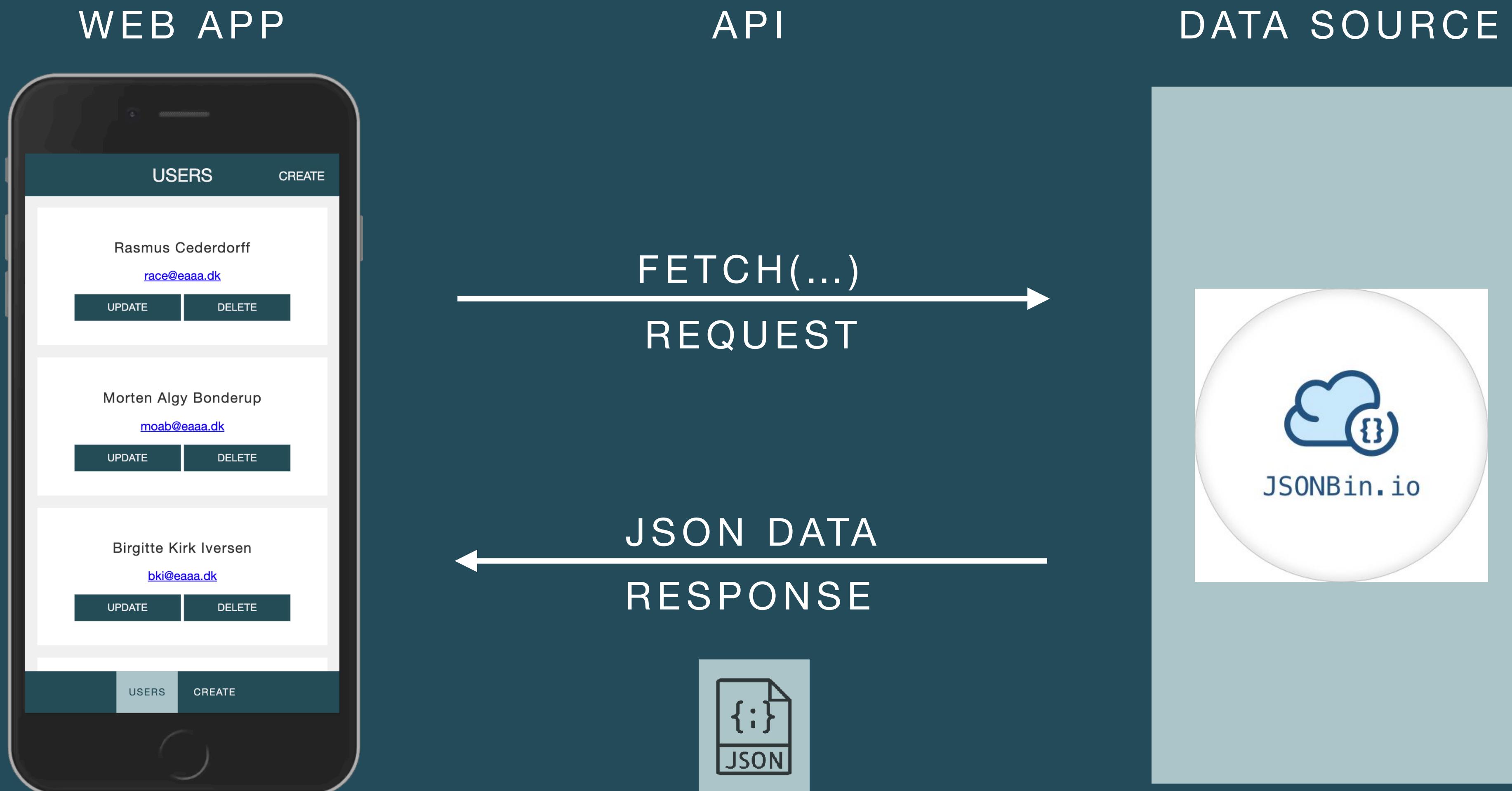
JSON & DATA STRUCTURES

JSON

JAVASCRIPT OBJECT NOTATION

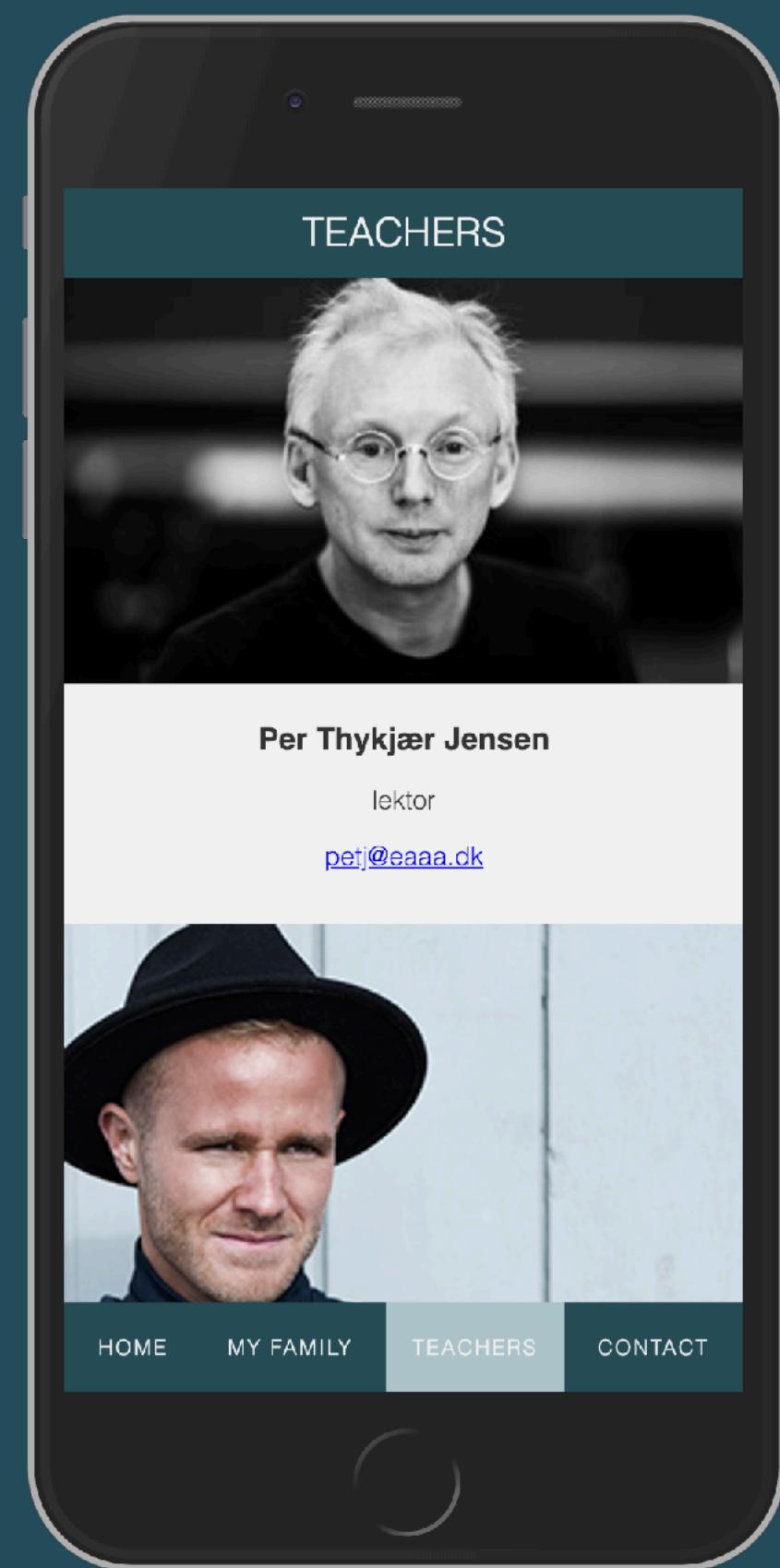
... A SYNTAX FOR STORING &
EXCHANGING DATA OVER THE WEB

FETCH, HTTP REQUEST & RESPONSE



THE GLUE: JSON & API

FRONTEND



API

FETCH(...)
REQUEST

BACKEND



JSON

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

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

JSON OBJECT

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

LIST OF JSON OBJECTS

JAVASCRIPT OBJECT NOTATION

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

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

JSON METHODS

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

FETCH

... CONVERTED JS OBJECT INTO JSON STRING

```
const user = {  
    name: "John",  
    age: 31  
};  
  
const response = await fetch('/article/fetch/post/user', {  
    method: 'POST',  
    headers: {  
        'Content-Type': 'application/json; charset=utf-8'  
    },  
    body: JSON.stringify(user) ←  
});  
  
const result = await response.json();
```

CANVAS USERS SPA

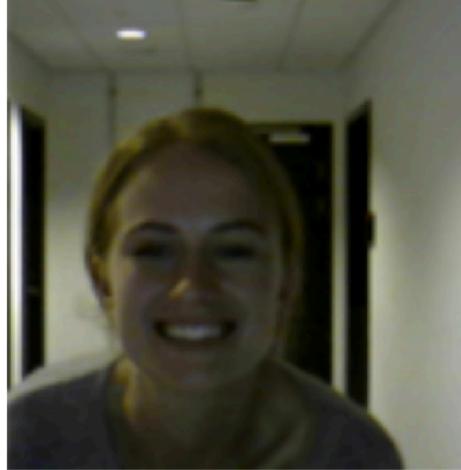
BACK

SINGLE PAGE WEB APP TEMP X +

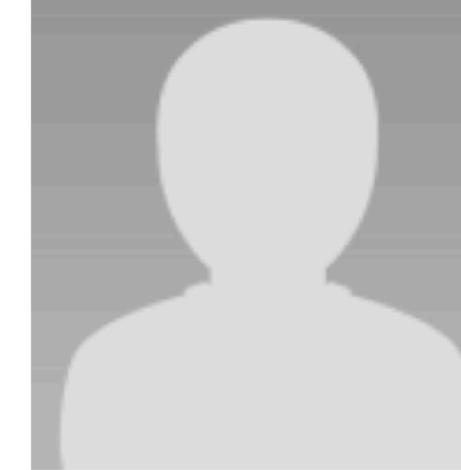
cederdorff.github.io/mdu-frontend/spa-canvas-users/#users Incognito :

ALL USERS

Order by: Choose here ▾ Filter by: All Courses: All Search



Clara Juul Birk
eaclbi@students.eaaa.dk
Student
Course: MDU-E20FRONT1
[UPDATE](#) [DELETE](#)



Mikkel Due Hørup Bjørnsholm
mikkel.13@hotmail.com
Student
Course: MDU-E20FRONT1
[UPDATE](#) [DELETE](#)



Morten Algy Bonderup
moab@eaaa.dk
Teacher
Course: admin
[UPDATE](#) [DELETE](#)



USERS CREATE





WRAP UP IN PAIRS

1. Present your solution and tell your struggles.
2. With code examples, discuss and explain the use of:
 1. DOM Manipulation
 2. Fetch, JSON, async & await
 3. Search, filter & sort
 4. CRUD (create, read, update & delete)
 5. Functions and events (eventlisteners, onclick, etc.)
 6. Global versus local variables
3. Compare your solutions and note your differences. What did you learn from each other?
4. Help each other explain the architecture of SPA and the role of a router.
5. Consider what you need a recap on and add themes, terms, questions and comments to the Padlet: https://eaaa.padlet.org/race/frontend_programming

IMPROVEMENTS?

JS, DOM, ID'S & FUNCTIONS

ID'S & FUNCTIONS: UPDATE

calling `selectUser(...)` with id of user

```
function appendUsers(users) {
  let htmlTemplate = "";
  for (const user of users) {
    htmlTemplate += /*html*/ `
      <article>
        <h3>${user.name}</h3>
        <p><a href="mailto:${user.mail}">${user.mail}</a></p>
        <button onclick="selectUser(${user.id})">Update</button>
        <button onclick="deleteUser(${user.id})">Delete</button>
      </article>
    `;
  }
  document.querySelector("#grid-users").innerHTML = htmlTemplate;
  showLoader(false);
}
```

```
function selectUser(id) {
  _selectedUserId = id;
  // find user by given user id
  const user = _users.find(user => user.id == _selectedUserId);
  // references to the input fields
  let nameInput = document.querySelector('#name-update');
  let mailInput = document.querySelector('#mail-update');
  // set inout values with selected user values
  nameInput.value = user.name;
  mailInput.value = user.mail;
  navigateTo("#/update");
}
```

```
function appendUsers(users) {
  let htmlTemplate = "";
  for (const user of users) {
    htmlTemplate += /*html*/
      <article>
        <h3>${user.name}</h3>
        <p><a href="mailto:${user.mail}">${user.mail}</a></p>
        <button onclick="selectUser(${user.id})">Update</button>
        <button onclick="deleteUser(${user.id})">Delete</button>
      </article>
  }
  document.querySelector("#grid-users").innerHTML = htmlTemplate;
  showLoader(false);
}
```

append to #grid-users

The screenshot shows a web application titled "User CRUD SPA" running at `127.0.0.1:5502/spa-user-crud-jsonbin/#/`. The application displays a grid of users with columns for name and email, and buttons for update and delete. The browser's developer tools are open, specifically the Elements tab of the DevTools. A blue arrow points from the explanatory text above to the highlighted element in the DOM tree: `<div id="grid-users" class="grid-container">`. The DOM tree also shows the generated HTML for each user, including the article elements and their contents.

Emil Gir' en Kramer
button 122.8x35 @hotmail.dk
UPDATE DELETE

Rasmus Cederdorff
race@aaaa.dk
UPDATE DELETE

Birgitte Kirk Iversen
bki@aaaa.dk
UPDATE DELETE

Morten Algy Bonderup
moab@aaaa.dk
UPDATE DELETE

Elements Console Sources > 1 | X

<div id="grid-users" class="grid-container">

<article>

<h3>Emil Gir' en Kramer </h3>

<p>...</p>

<button onclick="selectUser(1631775433990)">Update</button> == \$0

<button onclick="deleteUser(1631775433990)">Delete</button>

</article>

<article>

<h3>Rasmus Cederdorff</h3>

<p>...</p>

<button onclick="selectUser(1631792160482)">Update</button>

<button onclick="deleteUser(1631792160482)">Delete</button>

ID'S & FUNCTIONS: UPDATE

calling `selectUser(...)` with id of user

The screenshot shows the Chrome DevTools Elements tab. On the left, there's a sidebar with buttons for 'CREATE' and 'DELETE'. The main area displays a hierarchical tree of HTML elements. A specific button in the first article section is highlighted with a blue arrow pointing to it. The button has the following attributes: `onclick="selectUser(1631775433990)"`. The entire tree structure is as follows:

```
<div id="grid-users" class="grid-container">
  <article>
    <h3>Emil Gir' en Kramer </h3>
    <p>...</p>
    <button onclick="selectUser(1631775433990)">Update</button> == $0
    <button onclick="deleteUser(1631775433990)">Delete</button>
  </article>
  <article>
    <h3>Rasmus Cederdorff</h3>
    <p>...</p>
    <button onclick="selectUser(1631792160482)">Update</button>
    <button onclick="deleteUser(1631792160482)">Delete</button>
  </article>
  <article>
    <h3>Birgitte Kirk Iversen</h3>
  </article>
</div>
```

At the bottom of the DevTools interface, there are tabs for 'Styles', 'Computed', 'Layout', 'Event Listeners', and 'DOM Breakpoints'.

```
function selectUser(id) {
  _selectedUserId = id;
  // find user by given user id
  const user = _users.find(user => user.id === _selectedUserId);
  // references to the input fields
  let nameInput = document.querySelector('#name-update');
  let mailInput = document.querySelector('#mail-update');
  // set inout values with selected user values
  nameInput.value = user.name;
  mailInput.value = user.mail;
  navigateTo("#/update");
}
```

ID'S & FUNCTIONS: UPDATE

```
<section id="update" class="page">
  <header class="topbar">
    <a href="#" class="nav-link left">Back</a>
    <h2>Update user</h2>
    <a class="right" onclick="updateUser()">Update</a>
  </header>
  <form>
    <input type="text" id="name-update" placeholder="Type your name" required>
    <input type="email" id="mail-update" placeholder="Type your mail" required>
    <button type="button" name="button" onclick="updateUser()">Update User</button>
  </form>
</section>
```

```
async function updateUser() {
  showLoader(true);
  // references to input fields
  const nameInput = document.querySelector("#name-update");
  const mailInput = document.querySelector("#mail-update");
  // find user to update by given user id
  const userToUpdate = _users.find(user => user.id === _selectedUserId);
  // update values of user in array
  userToUpdate.name = nameInput.value;
  userToUpdate.mail = mailInput.value;
  // wait for update
  await updateJSONBIN(_users);
  // reset
  nameInput.value = "";
  mailInput.value = "";
  // navigating back
  navigateTo("/");
}
```

_selectedUserId is a global
variable saved in memory in
selectUser(...)

ID'S & FUNCTIONS: DELETE

calling deleteUser (...) with id of user

The screenshot shows a user interface for managing users. On the left, there's a list of users with their names, emails, and two buttons: 'UPDATE' and 'DELETE'. The 'DELETE' button for the first user is highlighted with a blue arrow. In the center, there's a grid container with articles for each user. The first article has a 'Delete' button with the onclick event set to 'deleteUser(1631775433990)'. The browser's developer tools Elements tab is open, showing the DOM structure. The blue arrow points from the 'DELETE' button in the UI to the 'deleteUser' call in the browser's code.

```
async function deleteUser(id) {  
  showLoader(true);  
  _users = _users.filter(user => user.id !== id);  
  await updateJSONBIN(_users);  
}
```

ASYNC JS

JavaScript reads and runs the script from top to bottom.

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

... by default JavaScript is synchronous.

JS IS SYNCHRONOUS & SINGLE-THREADED

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

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

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

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

/* ----- */

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

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

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

initApp();
```

WITH CALLBACKS WE CAN MAKE JS ASYNCHRONOUS

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

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

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

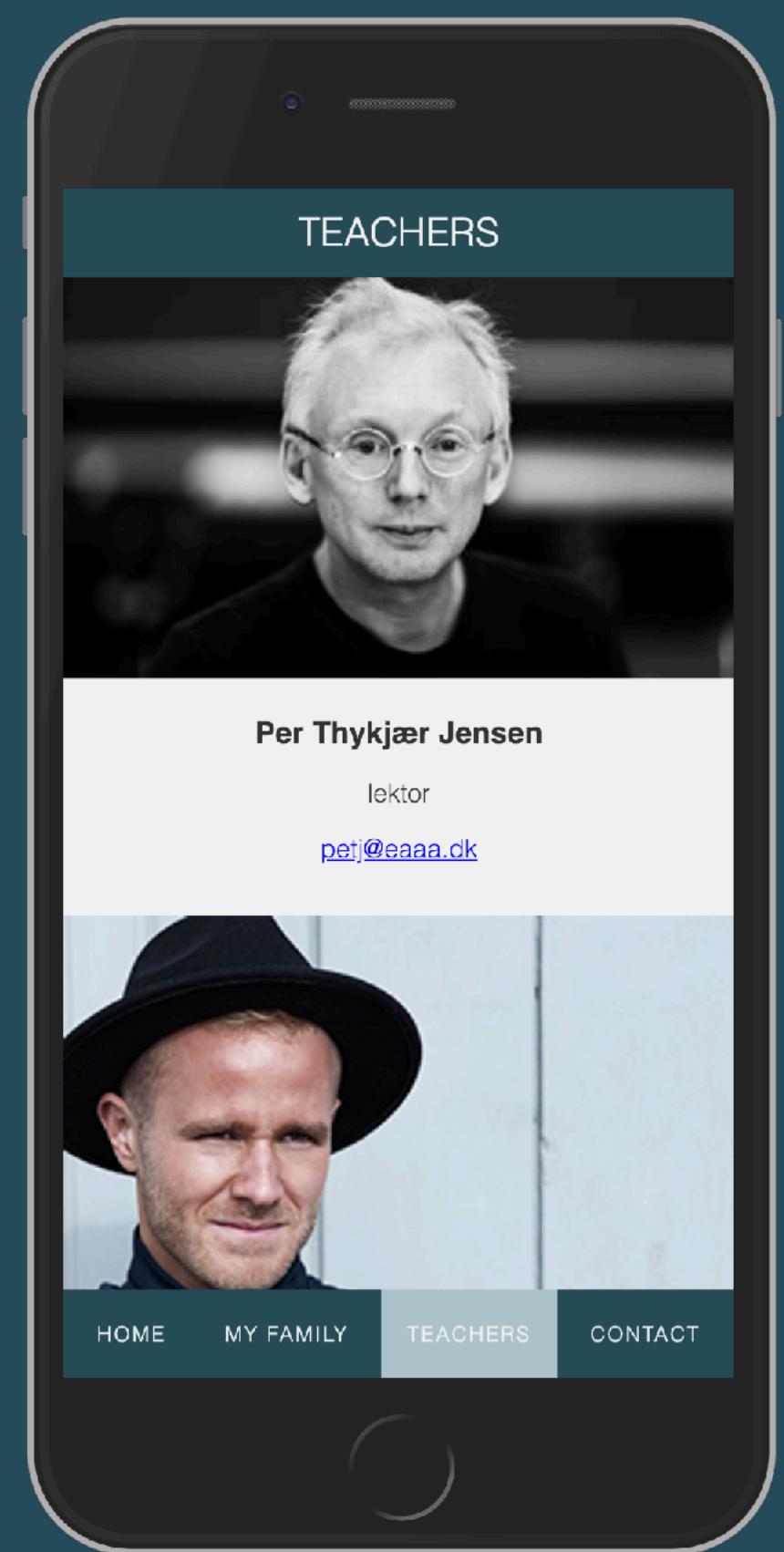
FETCH IS ASYNCHRONOUS

HTTP REQUEST IN JAVASCRIPT
...A WAY TO GET & POST DATA FROM & 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);
  });
// 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);
```

WEB DEVELOPMENT

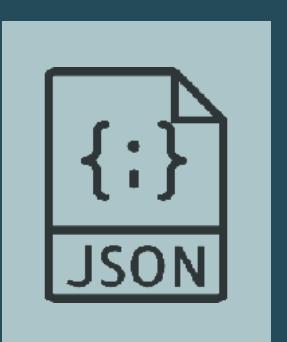
FRONTEND



(API)

FETCH(...)
REQUEST

JSON DATA
RESPONSE



BACKEND



"ASYNC AND AWAIT MAKE
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

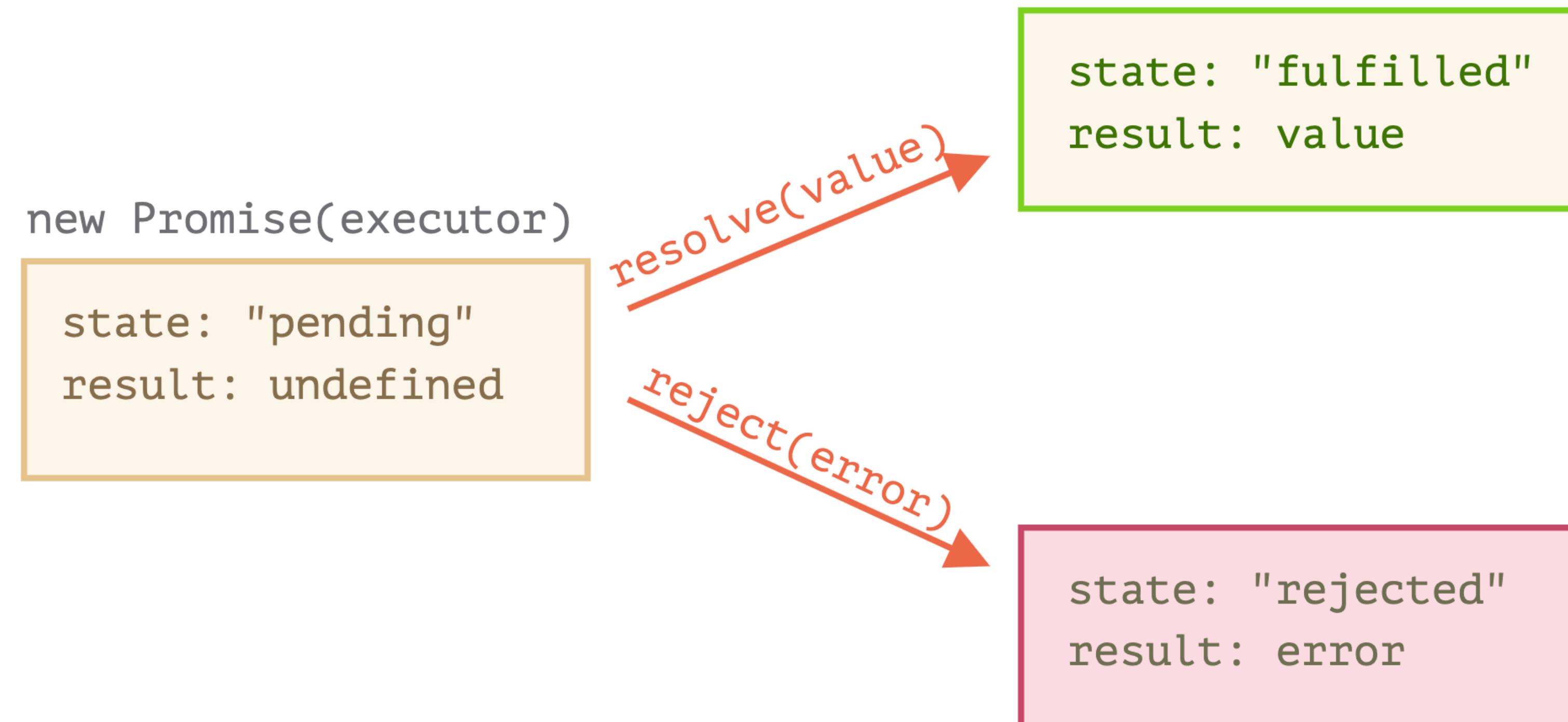
ASYNC MAKES A FUNCTION RETURN A PROMISE

AWAIT MAKES A FUNCTION WAIT FOR A PROMISE

WE CAN USE AWAIT TO WAIT FOR FETCH TO FINISH

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

THE TWO (OR THREE) STATES OF A PROMISE



FETCH: CALLBACK 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);
```

FETCH: HEADERS,
BODY & HTTP METHODS

FETCH WITHOUT HEADERS & BODY

THE DEFAULT HTTP METHODS IS “GET”

```
/**  
 * returns matches based on given userId  
 */  
async function getMatches(userId) {  
    const url = `${_baseUrl}?action=getMatches&userid=${userId}`;  
    const response = await fetch(url);  
    const data = await response.json();  
    return data;  
}
```

FETCH WITH HEADERS & BODY

... AND HTTP METHOD “POST”

```
// post new user to php userService using fetch(...)  
const response = await fetch(_baseUrl + "?action=createUser", {  
    method: "POST",  
    headers: { "Content-Type": "application/json; charset=utf-8" },  
    body: JSON.stringify(newUser) // parsing js object to json object  
});  
// waiting for the result  
const result = await response.json();  
console.log(result); // the result is the new updated users array
```

HTTP REQUEST METHODS

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>

REQUEST headers

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

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

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

https://developer.mozilla.org/en-US/docs/Glossary/Request_header

REQUEST body

When making HTTP request we sometimes need to send data. The data is wrapped inside of the request body.

The request body is one of the following:
a string (often JSON encoded string with data, object, arrays, etc.)
form data (form/multipart)
blob/ buffer source - binary data
URL search params (x-www-form-urlencoded)

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

FETCH WITH HEADERS & BODY

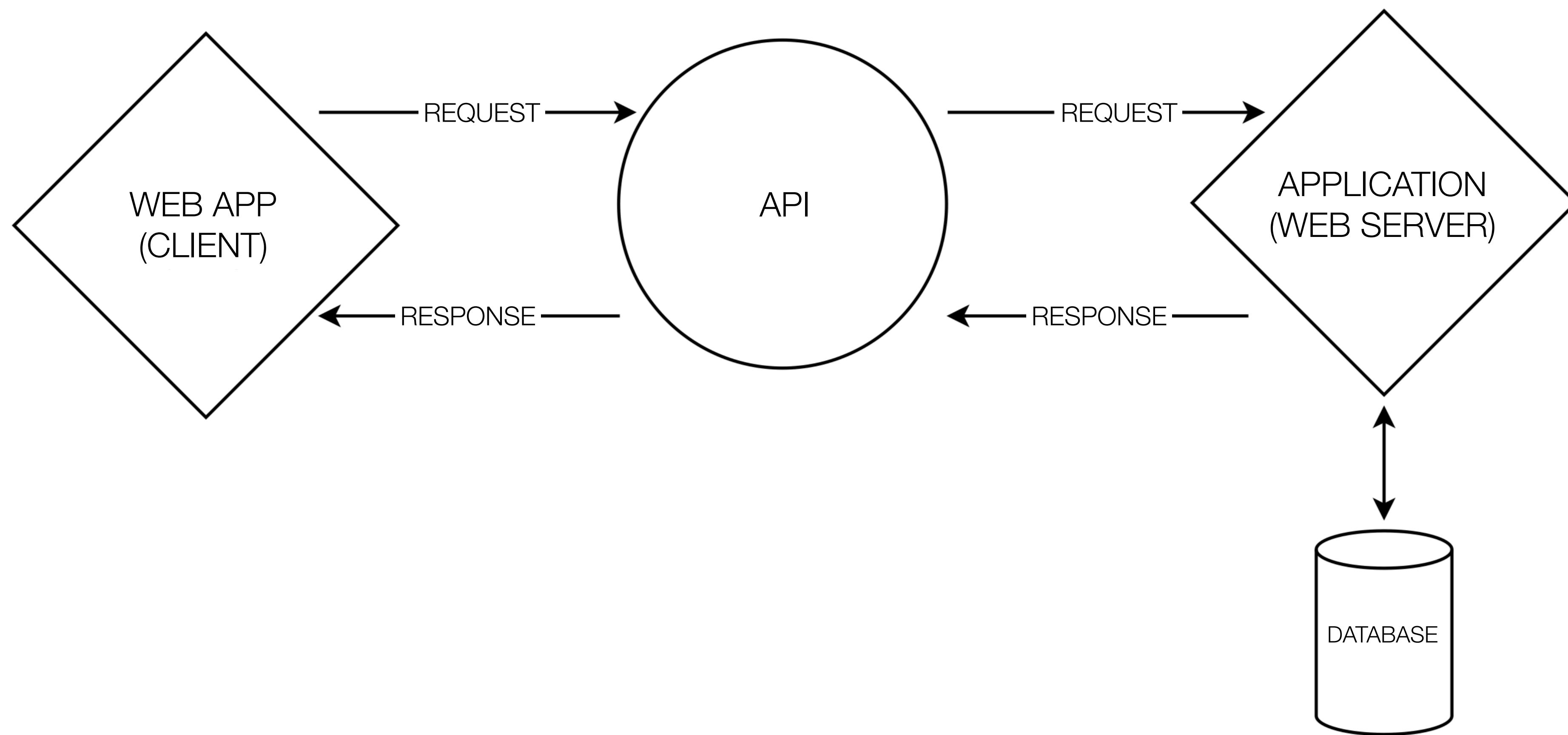
... AND HTTP METHOD “POST”

```
const user = {  
    name: "John",  
    age: 31  
};  
  
const response = await fetch('/article/fetch/post/user', {  
    method: 'POST',  
    headers: {  
        'Content-Type': 'application/json; charset=utf-8'  
    },  
    body: JSON.stringify(user)  
});  
  
const result = await response.json();
```

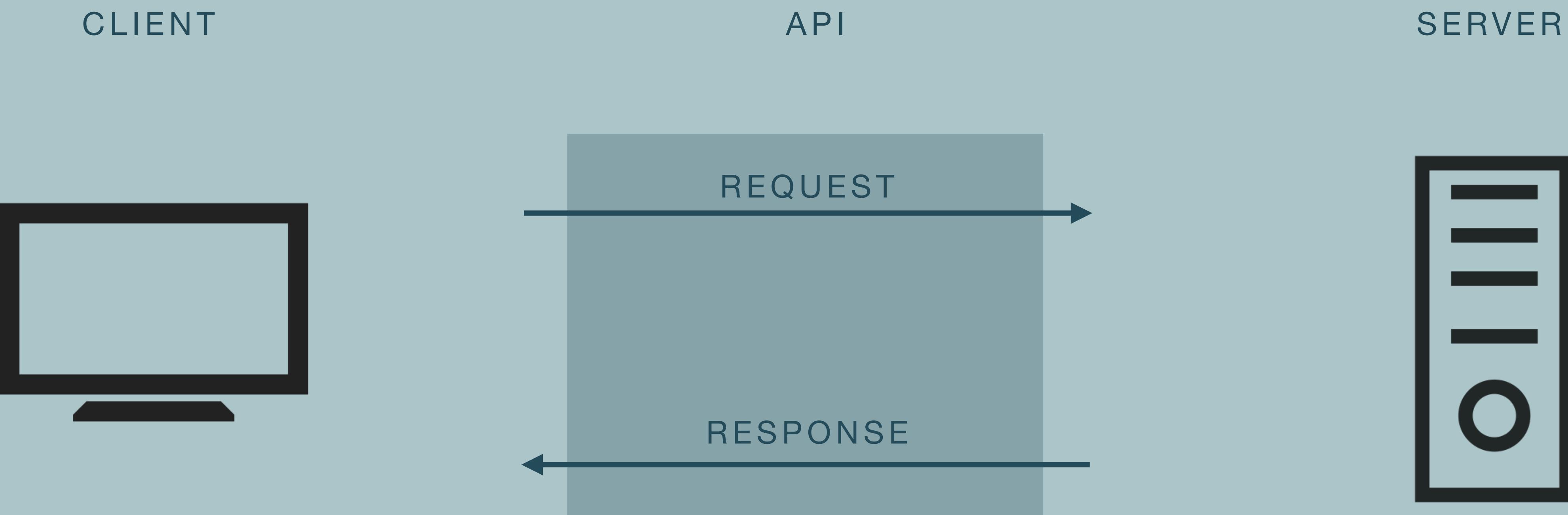
API

GET, POST & INTERACT WITH DATA & CONTENT

APPLICATION PROGRAMMING INTERFACE

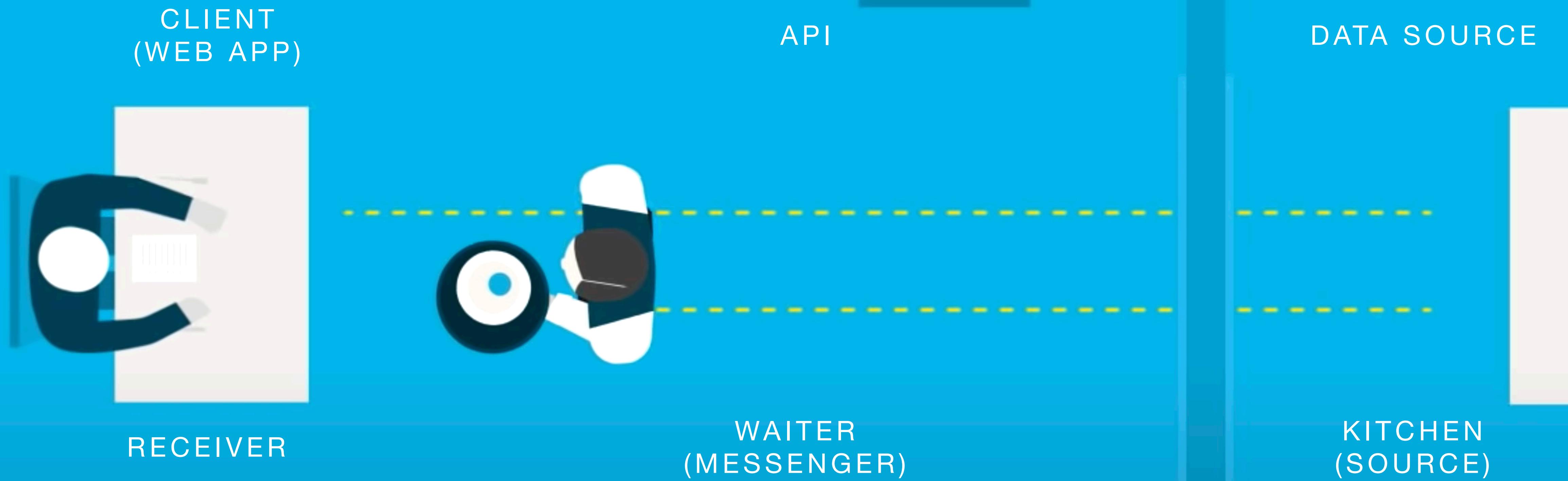


APPLICATION PROGRAMMING INTERFACE

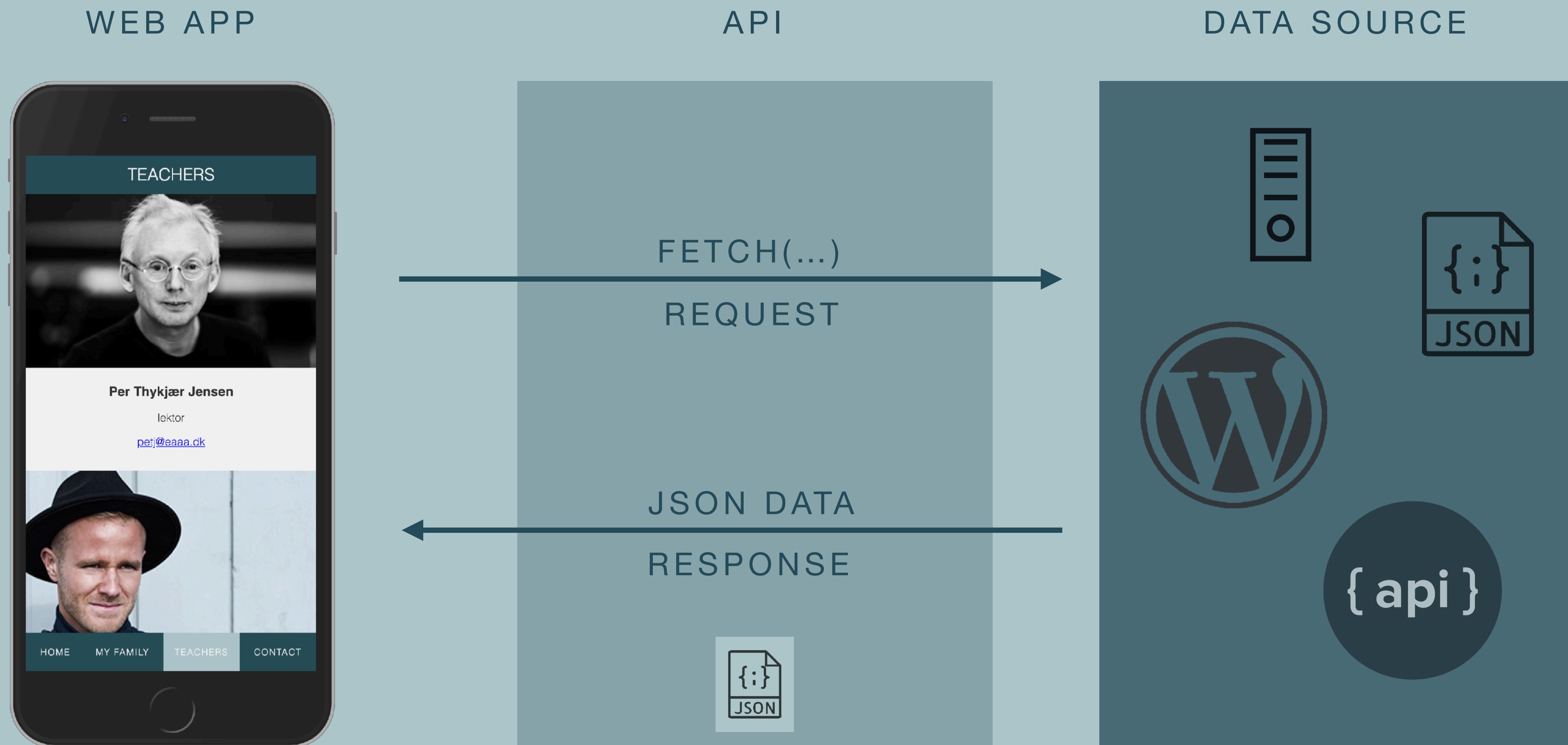




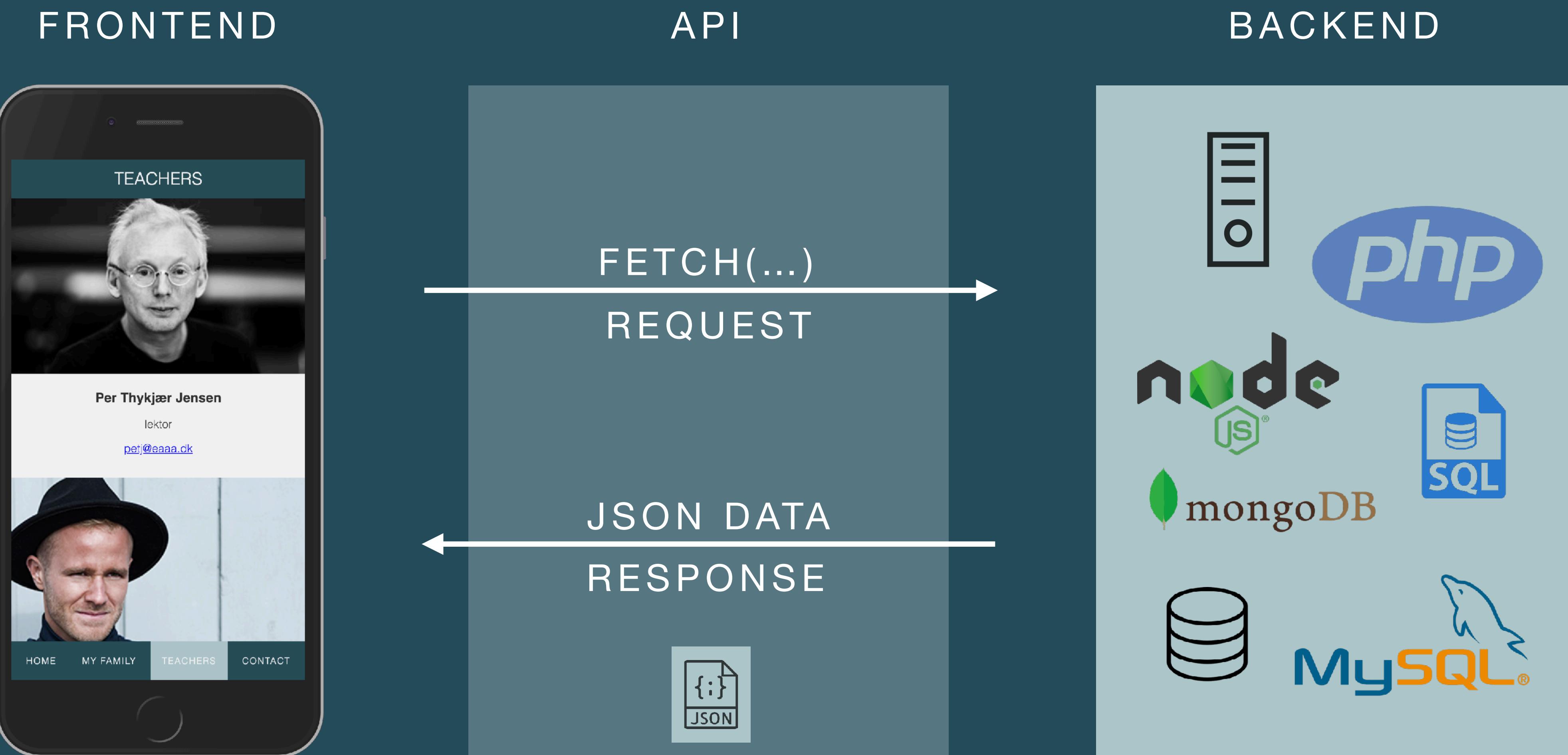
<https://www.youtube.com/watch?v=s7wmiS2mSXY>



APPLICATION PROGRAMMING INTERFACE

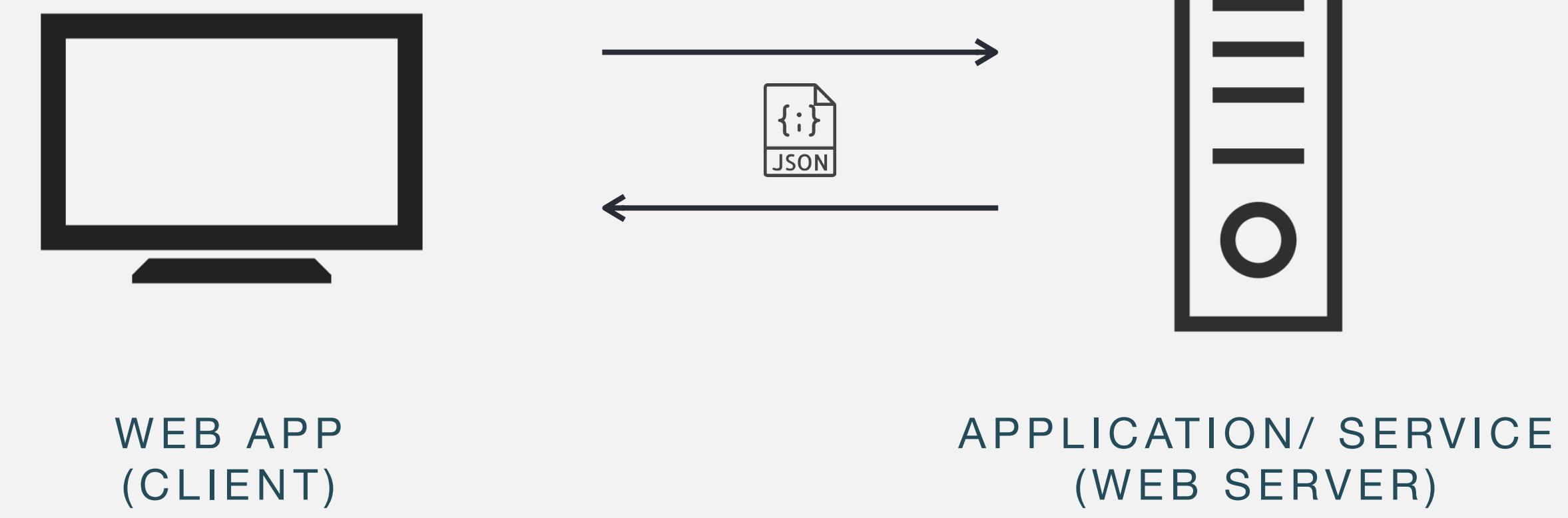


APPLICATION PROGRAMMING INTERFACE

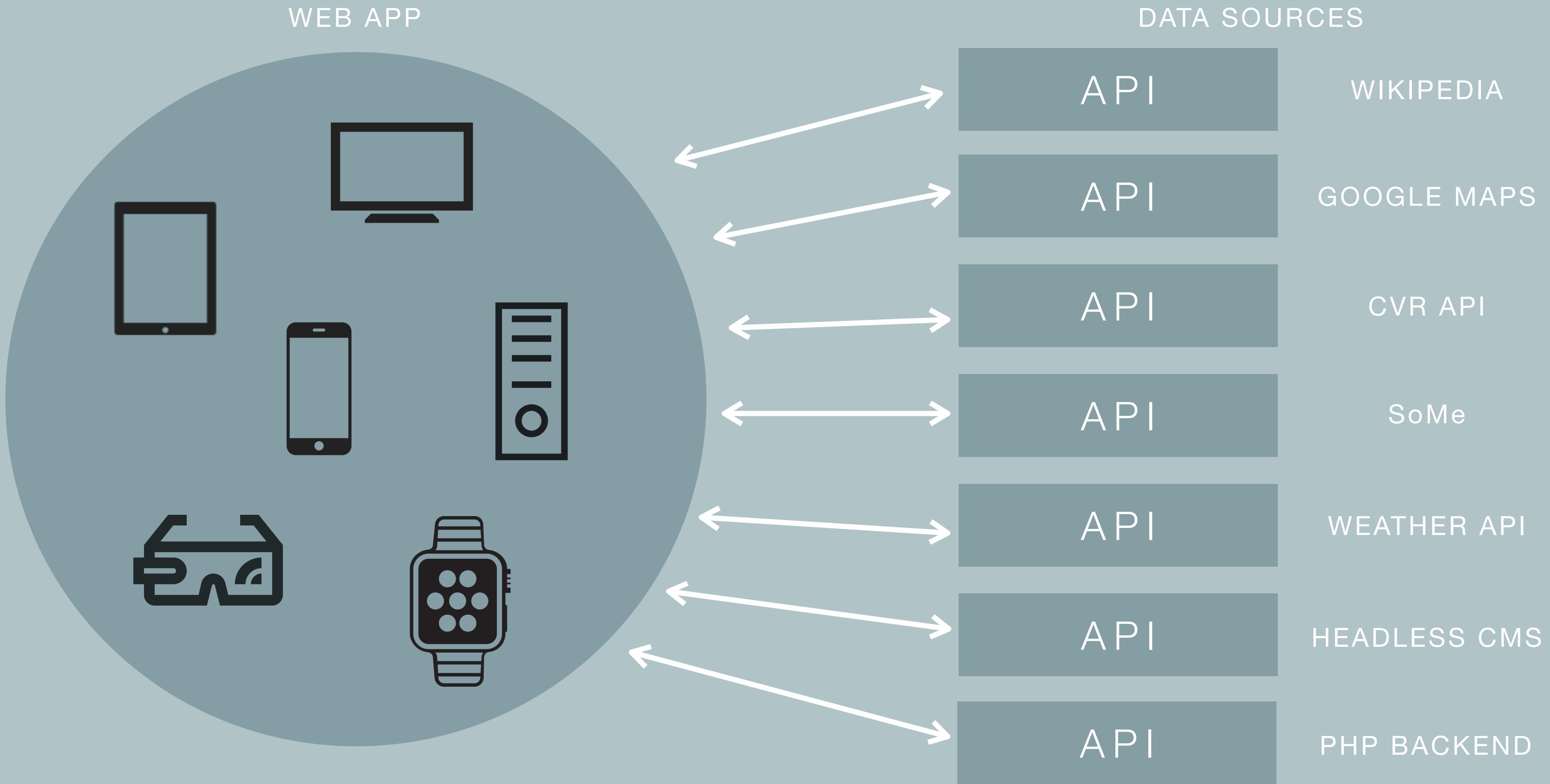


APPLICATION PROGRAMMING INTERFACE

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



API



JSON Storage & JSON Hosting x

jsonbin.io

Features Developers Company Pricing Login

SIMPLE & ROBUST JSON STORAGE SOLUTION

JSONBin.io provides a simple REST interface to store & retrieve your JSON data from the cloud. It helps developers focus more on app development by taking care of their Database Infrastructure.

Get Started

SCROLL DOWN

Feedback

The screenshot shows the homepage of JSONBin.io. At the top, there's a dark header with the site's name and a navigation bar with links for Features, Developers, Company, Pricing, and Login. The 'Login' button is highlighted with a black border. Below the header, the main title 'SIMPLE & ROBUST JSON STORAGE SOLUTION' is displayed in large, bold, black and blue letters. A descriptive paragraph follows, explaining the service's purpose. At the bottom of the page, there's a large, semi-transparent blue button with the text 'Get Started' in white. Below this button is a small circular icon with a dot and the text 'SCROLL DOWN' in capital letters. To the right, there's a vertical 'Feedback' button with a speech bubble icon.

{ } JSONBIN.io



The JSONBin.io API is organized around REST. Our API has predictable resource-oriented URLs, returns JSON-encoded responses, and uses standard HTTP response codes and verbs.

BINS API

Bins are nothing but JSON records. Bins API is the one you will be using the most to give you complete control of Creating, Updating, Reading & Deleting your JSON Data.

<https://jsonbin.io/api-reference>

CASE: DATING SPA

BACK

The screenshot shows a grid of user profiles. Each profile card contains the user's name, age, gender, and the gender they are looking for. The cards are arranged in four rows and three columns.

USERS		
Kasper Topp Age: 34, Gender: Male, Looking for: Female	Nicklas Andersen Age: 24, Gender: Male, Looking for: Female	Sarah Dybvad Age: 34, Gender: Female, Looking for: Male
Alex Handhiuc Age: 23, Gender: Male, Looking for: Female	Piotr Pospiech Age: 20, Gender: Male, Looking for: Female	Kristine Dzumakajeva Age: 25, Gender: Female, Looking for: Male
Jesson Getapal Age: 27, Gender: Male, Looking for: Female	Stiliyan Parzhanov Age: 28, Gender: Male, Looking for: Female	Aleksandra Wytulany Age: 23, Gender: Female, Looking for: Male
Barbora Byetusová Age: 24, Gender: Female, Looking for: Male	Sandra Nielsen Age: 21, Gender: Female, Looking for: Male	Rasmus Cederdorff Age: 31, Gender: Male, Looking for: Male

At the bottom of the page, there are two buttons: 'USERS' and 'CREATE'.

This screenshot shows the profile of a user named Nicklas. It includes the user's name, age, gender, and the number of matches found. Below the profile, there are buttons for 'UPDATE' and 'DELETE'. A list of potential matches is displayed in a grid format.

NICKLAS		
Nicklas Andersen Age: 24, Gender: Male Number of matches: 4		
UPDATE	DELETE	
Kristine Dzumakajeva Age: 25, Gender: Female	Aleksandra Wytulany Age: 23, Gender: Female	Barbora Byetusová Age: 24, Gender: Female
Sandra Nielsen Age: 21, Gender: Female		

At the bottom of the page, there are two buttons: 'USERS' and 'CREATE'.

Frontend: [users-match-frontend](#)

PHP Backend Service: [users-match-backend](#)

CASE: USER CRUD

CREATE, READ, UPDATE & DELETE

The screenshot shows a Single Page Application (SPA) titled "User CRUD SPA" running at `127.0.0.1:5502/spa-user-crud-jsonbin/index.html`. The application displays a grid of user records:

Name	Email	Actions
Rasmus Cederdorff	race@eaaa.dk	UPDATE DELETE
Dan Okkels Brendstrup	dob@eaaa.dk	UPDATE DELETE
Birgitte Kirk Iversen	bki@eaaa.dk	UPDATE DELETE
Morten Algy Bonderup	moab@eaaa.dk	UPDATE DELETE
Test	test@mail.dk	UPDATE DELETE
Testert	thahta@mail.dk	UPDATE DELETE

A "CREATE" button is located in the top right corner of the main header. At the bottom, there are "USERS" and "CREATE" buttons.

The browser's developer tools are open, specifically the "Console" tab, which shows the following JSON data structure:

```
Object { metadata: { createdAt: "2021-08-18T09:27:47.647Z", id: "61138ef2d5667e403a3fb6a1", private: false }, record: [Object; 6] }  
[Object; 6]  
0: { name: 'Rasmus Cederdorff', mail: 'race@eaaa.dk', id: 1620661599026 }  
1: { name: 'Dan Okkels Brendstrup', mail: 'dob@eaaa.dk', id: 1628164660 }  
2: { name: 'Birgitte Kirk Iversen', mail: 'bki@eaaa.dk', id: 1631111730 }  
3: { name: 'Morten Algy Bonderup', mail: 'moab@eaaa.dk', id: 1631111740 }  
4: { name: 'Test', mail: 'test@mail.dk', id: 1631179045479 }  
5: { name: 'Testert', mail: 'thahta@mail.dk', id: 1631179132944 }  
length: 6
```

spa-user-crud-jsonbin

```
// ===== GLOBAL VARS =====
let _users = [];
let _selectedUser;
const _baseUrl = "https://api.jsonbin.io/v3/b/61138ef2d5667e403a3fb6a1"
const _headers = {
  "X-Master-Key": "$2b$10$Uf1lbMtIPrrWeneN3Wz6JuDcyBu0z.1LbHiUg32QexCCJz3n0pos2",
  "Content-Type": "application/json"
};

// ===== READ =====

async function loadPersons() {
  const url = _baseUrl + "/latest"; // make sure to get the latest version
  const response = await fetch(url, {
    headers: _headers
  });
  const data = await response.json();
  console.log(data);
  _users = data.record;
  appendUsers(_users);
}
loadPersons();

function appendUsers(users) {
  let htmlTemplate = "";
  for (let user of users) {
    htmlTemplate += /*html*/
      `

<h3>${user.name}</h3>
        <p><a href="mailto:${user.mail}">${user.mail}</a></p>
        <button onclick="selectUser('${user.id}')">Update</button>
        <button onclick="deleteUser('${user.id}')">Delete</button>
      </article>`;
  }
  document.querySelector("#grid-users").innerHTML = htmlTemplate;
  showLoader(false);
}


```

The screenshot shows the JSONBin.io dashboard with a modal window open. The modal displays a list of users from a bin with ID 61138ef2d5667e403a3fb6a1. The users listed are Jan, Morten, Birgitte, and Rasmus Cederdorff, each with their name, email, and a unique ID.

DASHBOARD

BIN ID: 61138ef2d5667e403a3fb6a1

STATISTICS

GENERAL BINS (S)

UTILITIES

JSON Store **JSON Validator** **GeoIP Lookup** **Blog** **Change Log** **Contact**

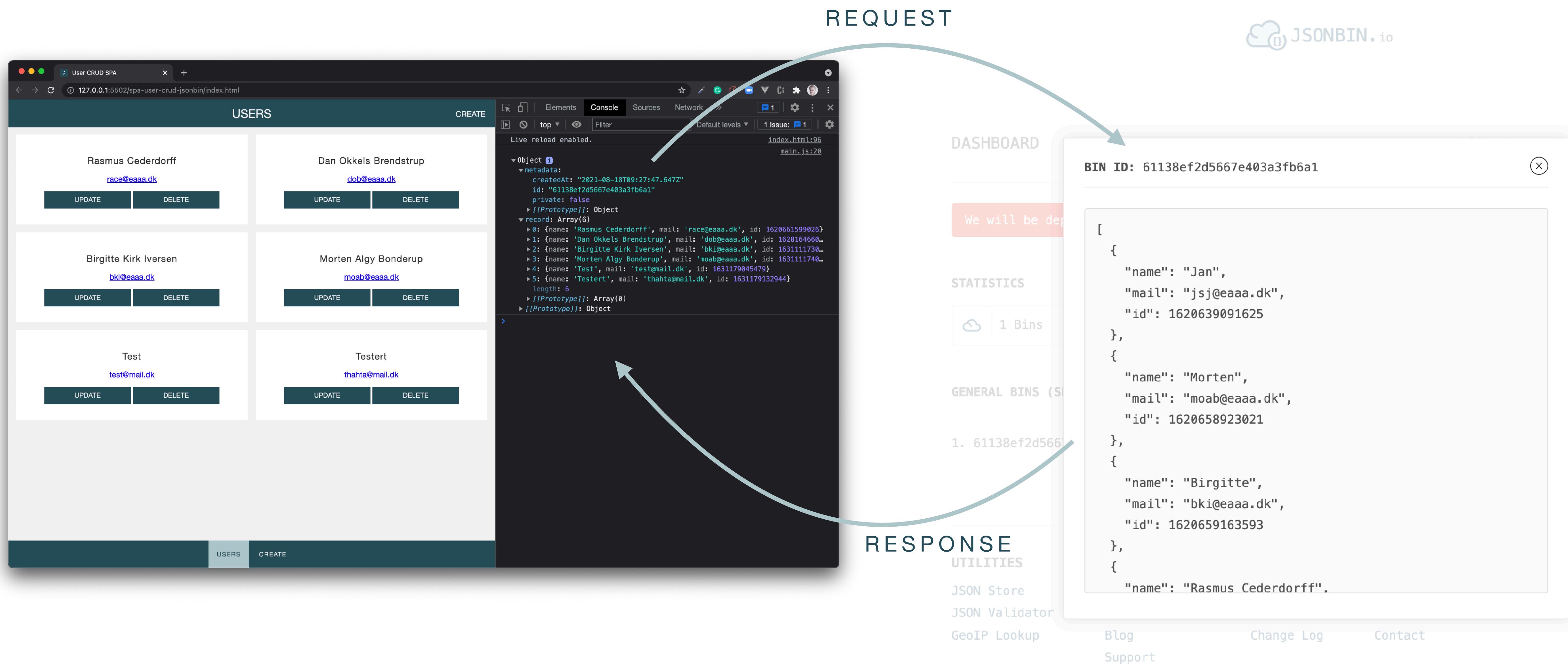
Support

DETAILS **DELETE**

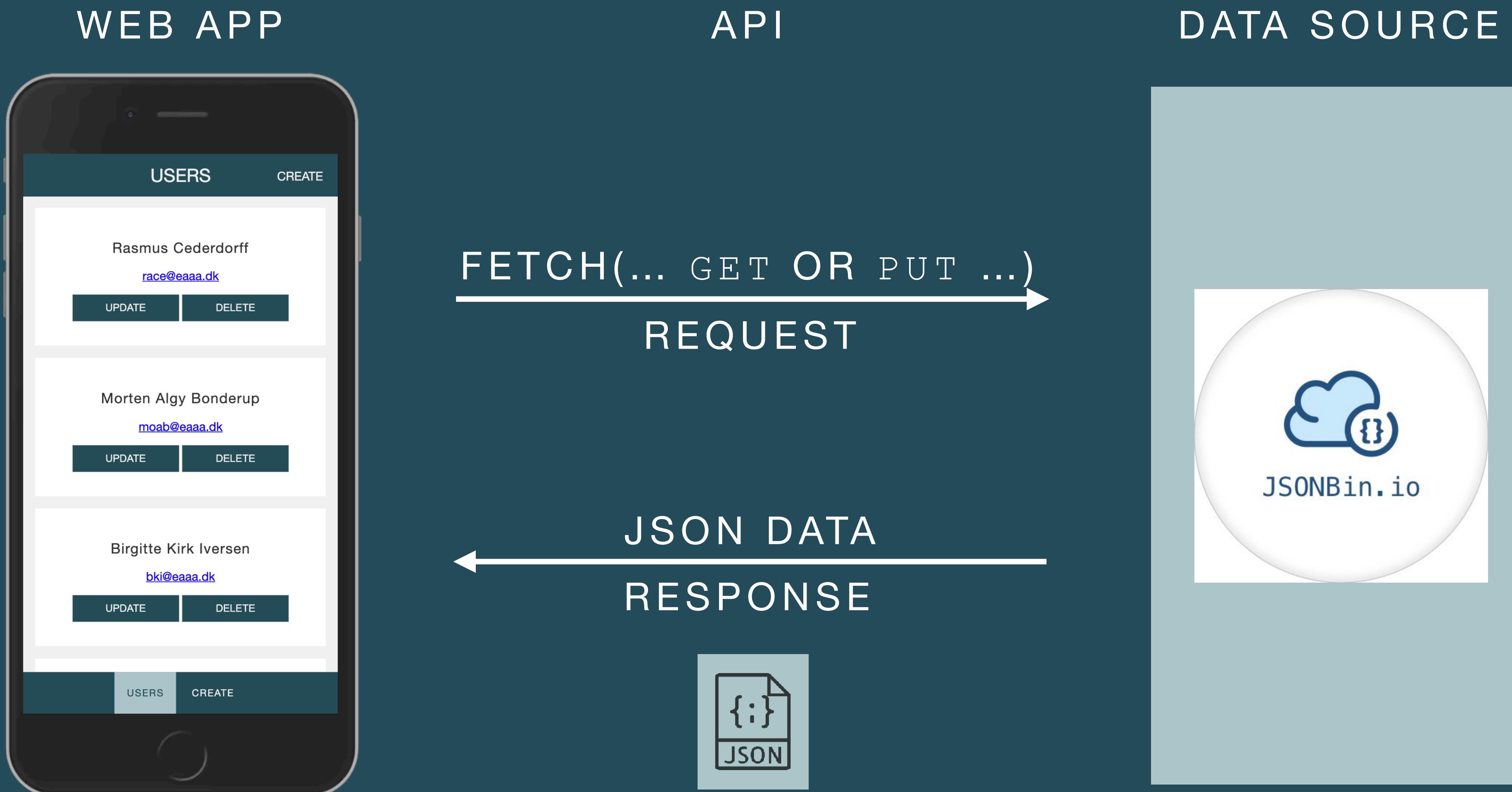
ID	Name	Email	ID
1. 61138ef2d5667e403a3fb6a1	Jan	jsj@aaaa.dk	1620639091625
	Morten	moab@aaaa.dk	1620658923021
	Birgitte	bki@aaaa.dk	1620659163593
	Rasmus Cederdorff		

CASE: USER CRUD

CREATE, READ, UPDATE & DELETE



FETCH, HTTP REQUEST & RESPONSE



STEPS

Template: dating-spa-template

1. Login and setup JSONBIN.io
2. Read users from your JSONBIN
3. Create a new user & update your JSONBIN
4. Update a user & update your JSONBIN
5. Delete a user & update your JSONBIN
6. Filter & sort by, search and detail view
7. Detail View with matches
7. Implement your own PHP Backend Service

CRUD

CREATE - a new object, a person, user, movie, posts, product etc.

READ - objects, one or all persons, users, movies, posts, products etc.

UPDATE - update an object, a person, user, movie, posts, product etc.

DELETE - delete an object, a person, user, movie, posts, product etc.

JSONBIN.IO

The screenshot shows the homepage of JSONBIN.io. The page features a large, dark teal header with the site's name. Below the header is a navigation bar with links for 'Features', 'Developers', 'Company', 'Pricing', and 'Login'. The 'Login' button is highlighted with a black border. The main content area has a white background with a light blue abstract geometric pattern at the bottom. It includes a title 'SIMPLE & ROBUST JSON STORAGE SOLUTION' and a descriptive paragraph about the service. A 'Get Started' button is located below the text. On the right side, there is a vertical sidebar with a 'Feedback' button.

{ } JSONBIN.io

Features Developers Company Pricing Login

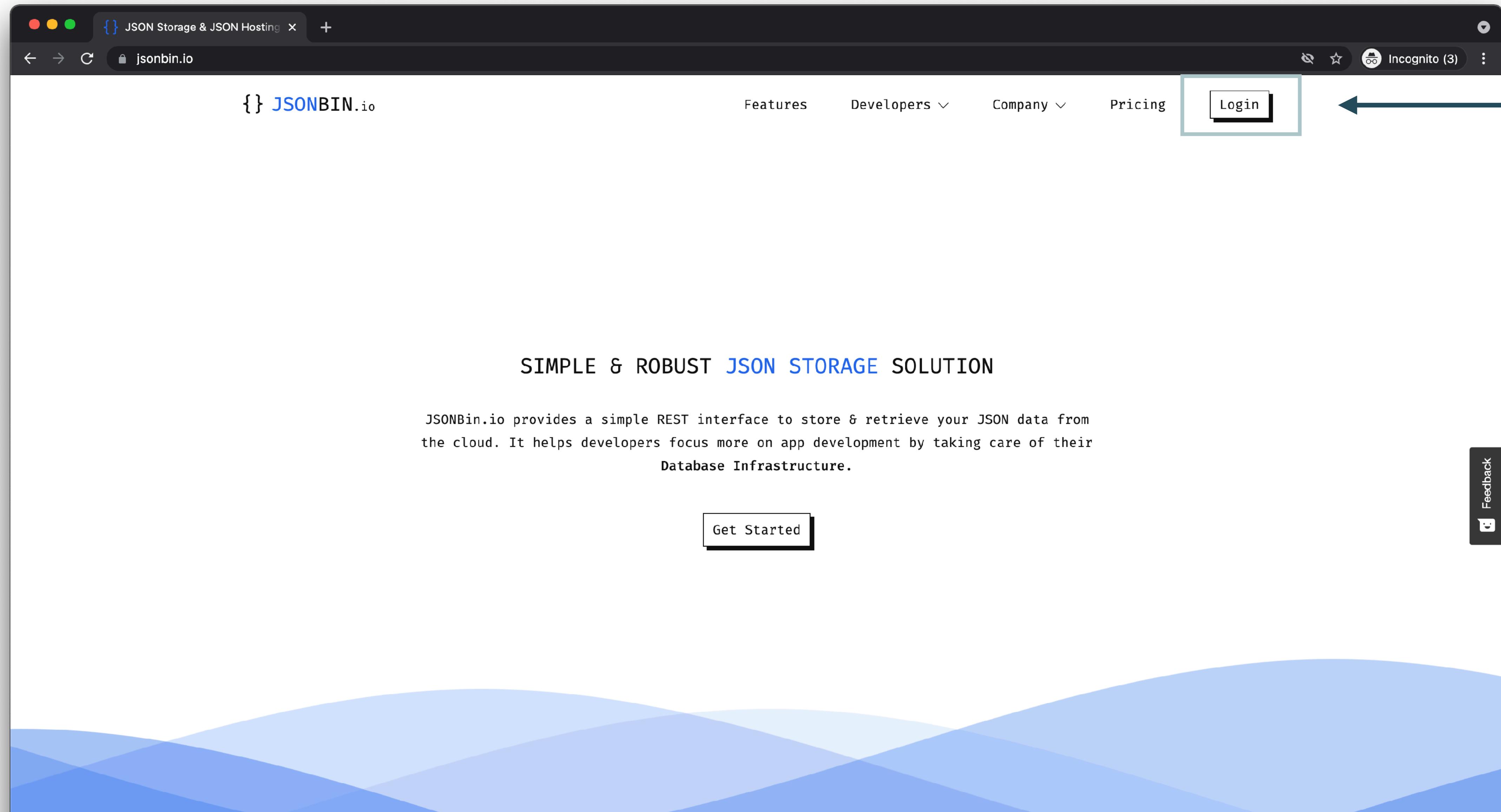
SIMPLE & ROBUST JSON STORAGE SOLUTION

JSONBIN.io provides a simple REST interface to store & retrieve your JSON data from the cloud. It helps developers focus more on app development by taking care of their Database Infrastructure.

Get Started

Feedback

LOGIN / SIGN UP



LOGIN / SIGN UP

A screenshot of a web browser window displaying the JSONBin.io login page. The page features a large "LOGIN / SIGN UP" header at the top. Below it is a navigation bar with links for "Features", "Developers", "Company", "Pricing", and a prominent "Login" button, which is highlighted with a black border. The main content area contains a heading "WHAT'S INCLUDED WITH A FREE ACCOUNT?" followed by a list of features. On the left side, there is a "SIGN UP OR LOGIN" section containing four social media sign-in buttons: "Google", "Github", "Facebook", and "Twitter". A blue arrow points from the bottom-left towards the "SIGN UP OR LOGIN" section. The URL "jsonbin.io/login" is visible in the browser's address bar.

{ } Login - JSONBin.io

{} jsonbin.io/login

{} JSONBIN.io

Features Developers Company Pricing Login

SIGN UP OR LOGIN

Google Github

Facebook Twitter

WHAT'S INCLUDED WITH A FREE ACCOUNT?

JSONBin.io is filled with rich features and follows an easy Store & Retrieve JSON approach. Try them now without upgrading to a paid plan.

- Store up to 10,000 JSON Files
- Create a Collection
- 10,000 API Requests to Try
- Public & Private JSON Files
- API Access Logs

SERVING OVER MILLION REQUESTS EVERY DAY AND IS TRUSTED BY MORE THAN 40,000 CUSTOMERS WORLDWIDE.

Feedback

DASHBOARD -> CREATE NEW BIN

A screenshot of a web browser displaying the JSONBIN.io dashboard. The browser window has a dark theme with a light gray header bar. The title bar shows the URL "jsonbin.io/dashboard". The main content area features the JSONBIN logo (a blue cloud icon with a gear) and the word "JSONBIN.io". Below the logo, there are two sections: "DASHBOARD" on the left and a user profile on the right. The user profile says "Hello, Rasmus Cederdorff" and includes a small profile picture. A red banner at the bottom of the dashboard section contains the text: "We will be deprecating API v2 on 1st October, 2021. Checkout API v3 Documentation for more info." In the center, under the heading "STATISTICS", there are three boxes: "0 Bins" (with a cloud icon), "0 Collections" (with a folder icon), and "10000 requests pending" (with an HTTP icon). Below the statistics, there is a section titled "GENERAL BINS (SHOWING RECENT 200 NON-COLLECTION RECORDS ONLY)" which displays the message "No bins found." To the right of this section is a "VIEW MASTER KEY" link and a "CREATE NEW" button, which is highlighted with a blue border and a black arrow pointing to it from the right side of the image. At the bottom of the page, there are links for "UTILITIES", "RESOURCES", "OTHERS", "ABOUT", and "SOCIAL".

{} Dashboard | JSONbin.io | Free x +

← → C jsonbin.io/dashboard 🔍 ☆ 🕵️ Inkognito ⋮

JSONBIN.io

Hello, Rasmus Cederdorff

We will be deprecating API v2 on 1st October, 2021. Checkout API v3 Documentation for more info.

STATISTICS

0 Bins

0 Collections

10000 requests pending

GENERAL BINS (SHOWING RECENT 200 NON-COLLECTION RECORDS ONLY)

No bins found.

VIEW MASTER KEY

CREATE NEW

UTILITIES

RESOURCES

OTHERS

ABOUT

SOCIAL

JSON Store

Pricing

Status

About

Facebook

JSON Validator

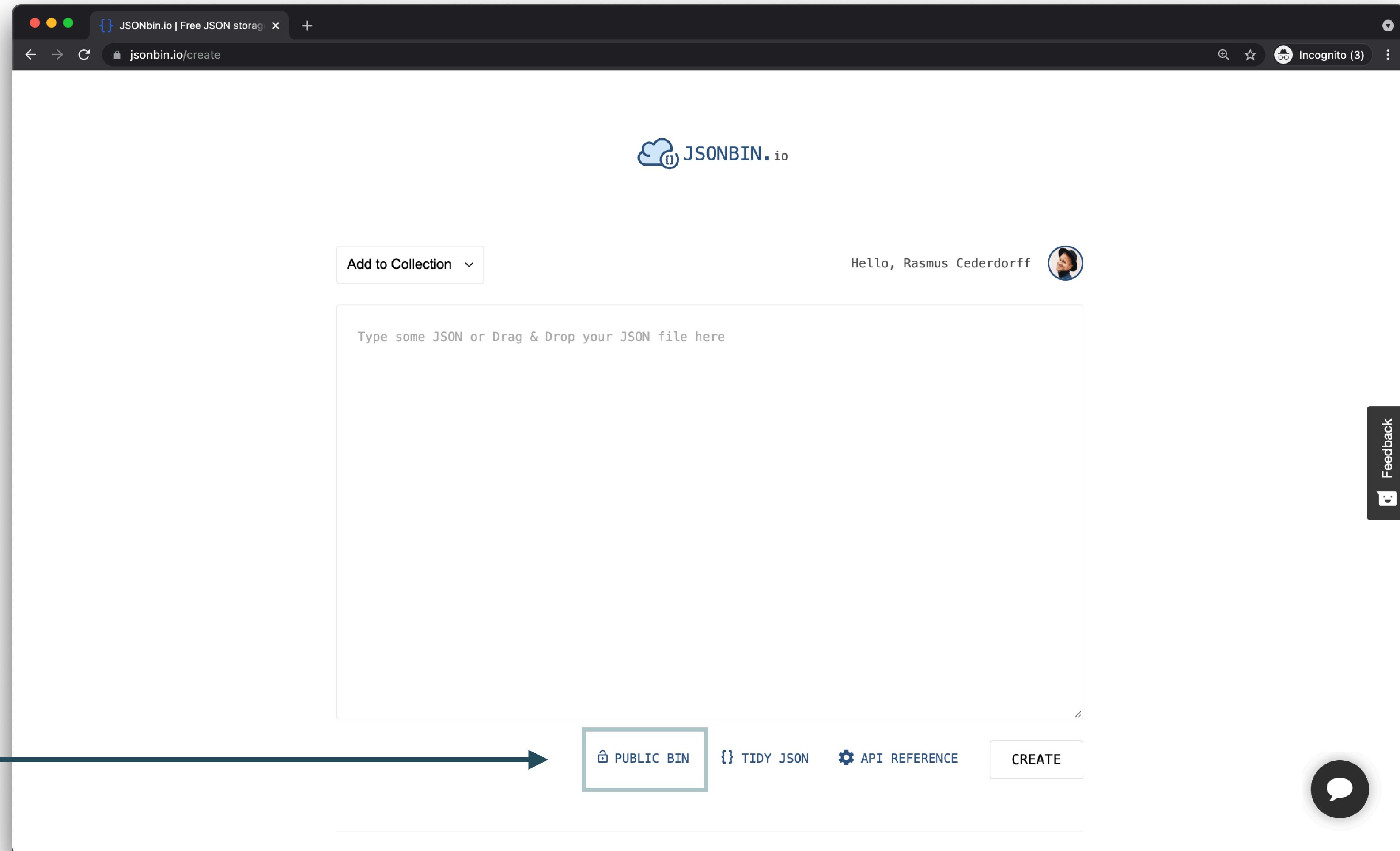
API Reference

Features

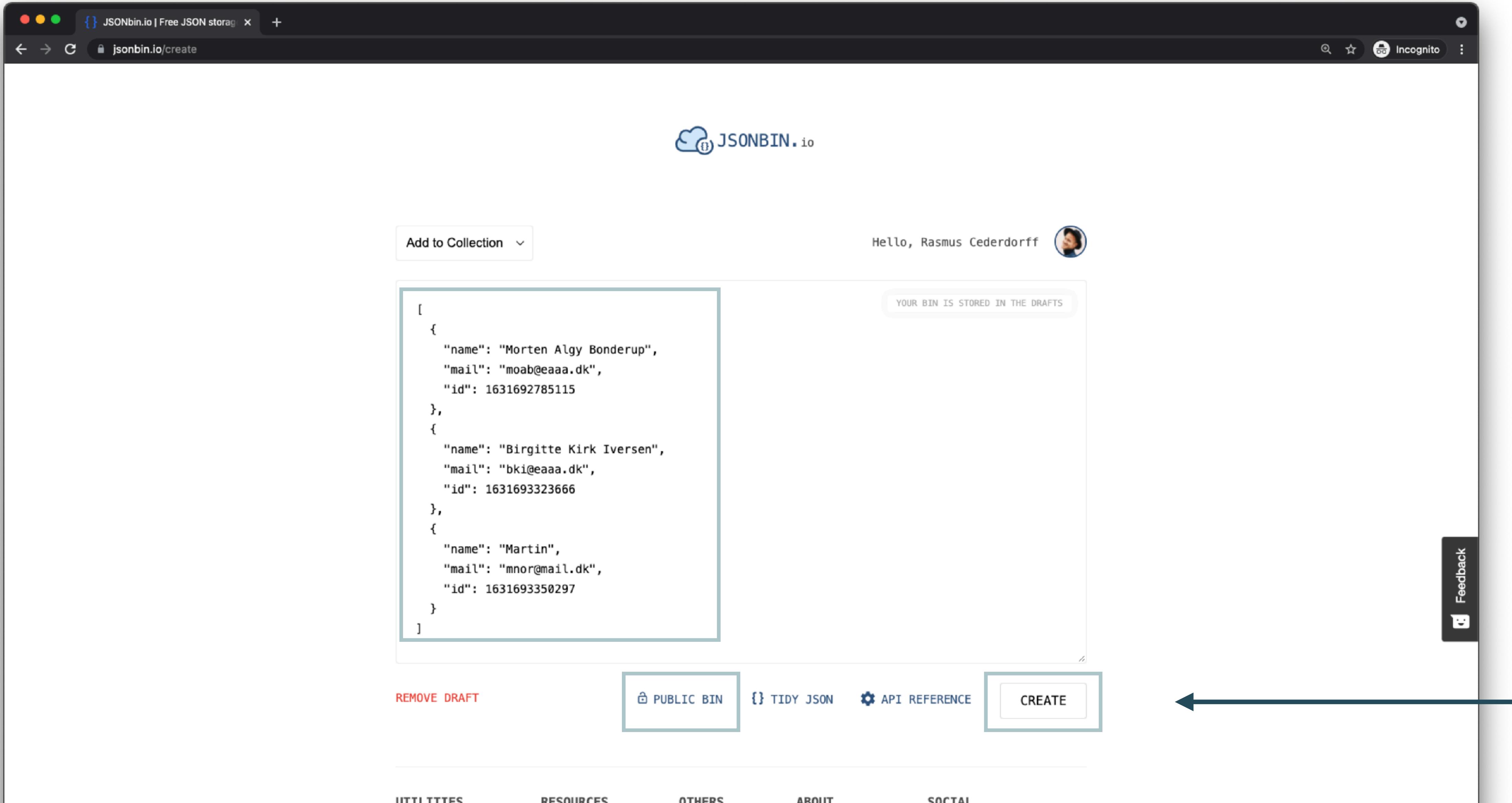
Testimonials

Twitter

CREATE PUBLIC BIN



PASTE SOME JSON & SAVE



A screenshot of a web browser displaying the JSONbin.io website. The page shows a list of three JSON objects stored in a draft bin. The objects represent people with their names, emails, and IDs. The interface includes a header with the JSONBIN logo, user authentication, and collection management. A large arrow points from the bottom right towards the 'CREATE' button.

```
[{"name": "Morten Algy Bonderup", "mail": "moab@eaaa.dk", "id": 1631692785115}, {"name": "Birgitte Kirk Iversen", "mail": "bki@eaaa.dk", "id": 1631693323666}, {"name": "Martin", "mail": "mnor@mail.dk", "id": 1631693350297}]
```

Add to Collection ▾

Hello, Rasmus Cederdorff

YOUR BIN IS STORED IN THE DRAFTS

REMOVE DRAFT

CREATE

Feedback

PASTE SOME INITIAL JSON

```
[  
  {  
    "id": "1",  
    "name": "Kasper",  
    "age": 34,  
    "gender": "Male",  
    "lookingFor": "Female"  
  },  
  {  
    "id": "2",  
    "name": "Nicklas",  
    "age": 22,  
    "gender": "Male",  
    "lookingFor": "Female"  
  },  
  {  
    "id": "3",  
    "name": "Sarah",  
    "age": 34,  
    "gender": "Female",  
    "lookingFor": "Male"  
  }]
```

Array

Objects

GO BACK TO DASHBOARD

A screenshot of a web browser displaying the JSONBin.io dashboard. The URL in the address bar is `jsonbin.io/6141ae5d9548541c29b24fdc`. The page title is `{ } JSONBin.io | Free JSON storage`. The browser interface includes standard controls like back, forward, and search.

The main content area shows a JSON array with three objects:

```
[  
  {  
    "name": "Morten Algy Bonderup",  
    "mail": "moab@eaaa.dk",  
    "id": 1631692785115  
  },  
  {  
    "name": "Birgitte Kirk Iversen",  
    "mail": "bki@eaaa.dk",  
    "id": 1631693323666  
  },  
  {  
    "name": "Martin",  
    "mail": "mnor@mail.dk",  
    "id": 1631693350297  
  }]
```

Below the JSON array, there is an [Access URL](#) button with the value `https://api.jsonbin.io/b/6141ae5d9548541c29b24fdc`, and buttons for [VIEW](#) and [COPY](#).

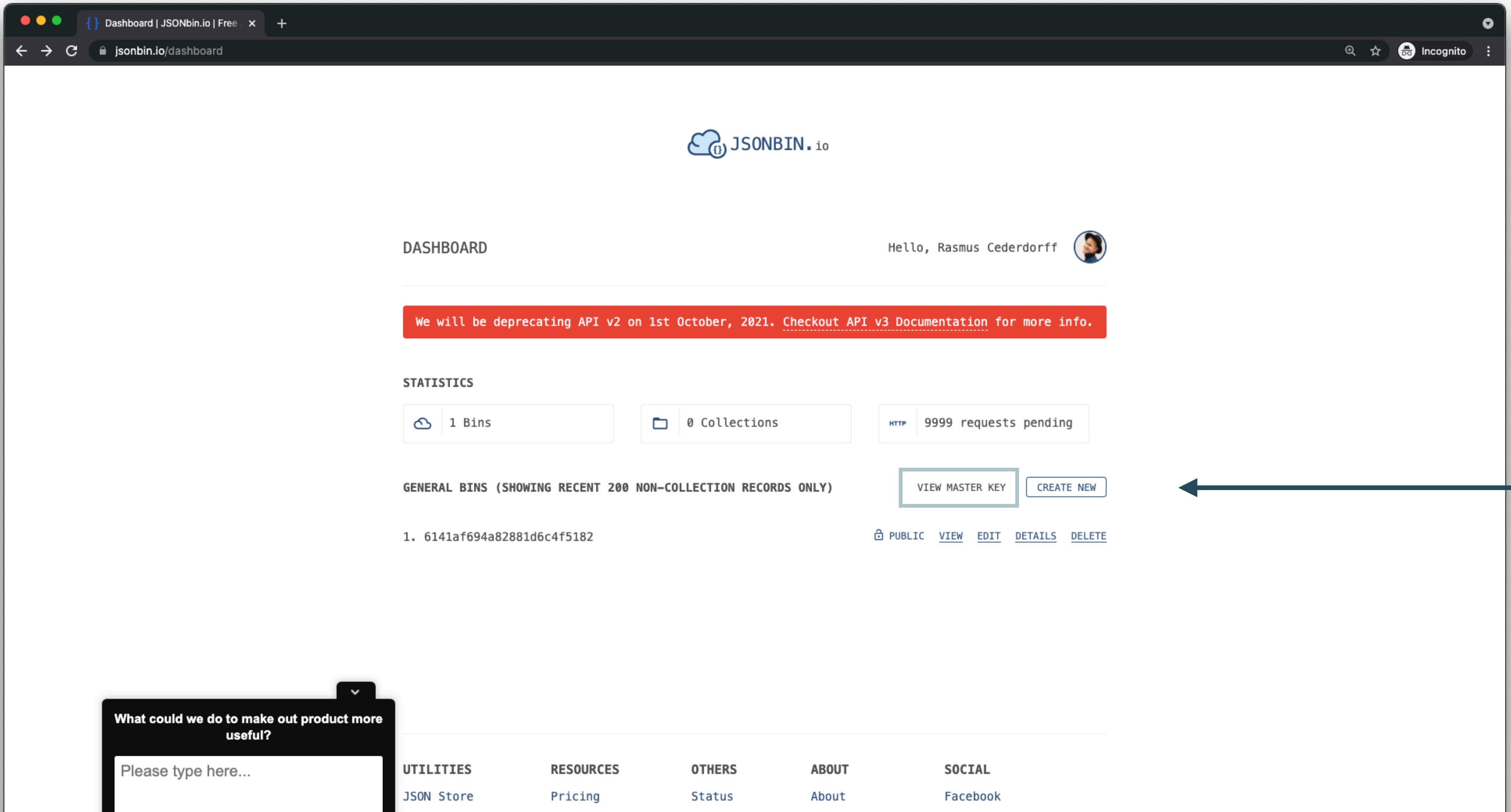
The top right corner features a user profile menu for "Rasmus Cederdorff". The "Dashboard" item in this menu is highlighted with a green box and a large arrow pointing towards it from the left side of the image.

The user profile menu items include:

- Dashboard (highlighted)
- Collections
- Schema Docs
- Webhooks
- API Key
- API Activity
- My Account
- API Reference
- Buy Pro Plan
- Logout

A vertical sidebar on the right is labeled "Feedback".

VIEW MASTER KEY



A screenshot of a web browser displaying the JSONBIN.io dashboard. The page features a dark header with the JSONBIN logo and navigation links. Below the header is a prominent red banner with white text: "We will be deprecating API v2 on 1st October, 2021. Checkout API v3 Documentation for more info." The main content area includes sections for "STATISTICS" (1 Bin, 0 Collections, 9999 requests pending), "GENERAL BINS" (listing one item with ID 6141af694a82881d6c4f5182), and a "SOCIAL" section at the bottom. A large blue arrow points from the right side of the image towards the "VIEW MASTER KEY" button, which is highlighted with a light blue border. The overall interface is clean and modern, with a focus on data storage and management.

Dashboard | JSONBIN.io | Free

jsonbin.io/dashboard

Incognito

JSONBIN.io

Hello, Rasmus Cederdorff

DASHBOARD

We will be deprecating API v2 on 1st October, 2021. Checkout API v3 Documentation for more info.

STATISTICS

1 Bins

0 Collections

9999 requests pending

GENERAL BINS (SHOWING RECENT 200 NON-COLLECTION RECORDS ONLY)

1. 6141af694a82881d6c4f5182

PUBLIC VIEW EDIT DETAILS DELETE

VIEW MASTER KEY

CREATE NEW

What could we do to make our product more useful?

Please type here...

UTILITIES

JSON Store

RESOURCES

Pricing

OTHERS

Status

ABOUT

About

SOCIAL

Facebook

COPY AND PASTE KEY IN JS

The image shows a screenshot of a web browser window displaying the JSONBIN.io website. The browser title bar says "Incognito". The main content area shows the "API KEYS" section with a "Secret Key / X-Master-Key" field containing the value "\$2b\$10\$iA0LguBJ8cd9y87RJTcXy.hlVPE8tVaiURQ1q9DJH2YLmj.4pmhq". Below this is a "GENERATE KEY / COPY KEY" button. A large green arrow points from this button down to a code editor window on the right. The code editor contains the following JavaScript code:

```
// ===== GLOBAL VARIABLES =====
let _users = [];
let _selectedUserId;
const _baseUrl = "https://api.jsonbin.io/v3/b/6141af694a82881d6c4f5182";
const _headers = {
  "X-Master-Key": "$2b$10$iA0LguBJ8cd9y87RJTcXy.hlVPE8tVaiURQ1q9DJH2YLmj.4pmhq",
  "Content-Type": "application/json"
};
```

At the bottom of the page, there are navigation links for UTILITIES, RESOURCES, OTHERS, ABOUT, and SOCIAL.

GO BACK TO DASHBOARD. COPY & PASTE ID OF YOUR BIN

The image shows a screenshot of the JSONBIN.io dashboard. At the top, there's a red banner with the text: "We will be deprecating API v2 on 1st October, 2021. Checkout API v3 Documentation for more info." Below the banner, the dashboard has sections for DASHBOARD, STATISTICS, and GENERAL BINS. The GENERAL BINS section shows one bin, no collections, and 9999 pending requests. A specific bin entry is highlighted with a blue arrow pointing to its ID in the code editor on the right. The code editor contains a snippet of JavaScript-like code:

```
// ===== GLOBAL VARIABLES =====
let _users = [];
let _selectedUserId;
const _baseUrl = "https://api.jsonbin.io/v3/b/6141af694a82881d6c4f5182";
const _headers = {
  "X-Master-Key": "$2b$10$iAOlgubJ8cd9y87RJTcXy.hlVPE8tVaiaURQ1q9DJH2YLmj.4pmhq",
  "Content-Type": "application/json"
};
```

The bin ID "6141af694a82881d6c4f5182" is highlighted with a yellow box in the code editor.

UTILITIES **RESOURCES** **OTHERS** **ABOUT** **SOCIAL**

[JSON Store](#) [Pricing](#) [Status](#) [About](#) [Facebook](#)
[JSON Validator](#) [API Reference](#) [Features](#) [Testimonials](#) [Twitter](#)

```
// ===== GLOBAL VARS =====
let _users = [];
let _selectedUser;
const _baseUrl = "https://api.jsonbin.io/v3/b/61138ef2d5667e403a3fb6a1"
const _headers = {
  "X-Master-Key": "$2b$10$Uf1lbMtIPrrWeneN3Wz6JuDcyBu0z.1LbHiUg32QexCCJz3n0poS2",
  "Content-Type": "application/json"
};

// ===== READ =====

async function loadPersons() {
  const url = _baseUrl + "/latest"; // make sure to get the latest version
  const response = await fetch(url, {
    headers: _headers
  });
  const data = await response.json();
  console.log(data);
  _users = data.record;
  appendUsers(_users);
}
loadPersons();

function appendUsers(users) {
  let htmlTemplate = "";
  for (let user of users) {
    htmlTemplate += /*html*/
      `

<h3>${user.name}</h3>
        <p><a href="mailto:${user.mail}">${user.mail}</a></p>
        <button onclick="selectUser('${user.id}')">Update</button>
        <button onclick="deleteUser('${user.id}')">Delete</button>
      </article>`;
  }
  document.querySelector("#grid-users").innerHTML = htmlTemplate;
  showLoader(false);
}


```

The screenshot shows the JSONBin.io dashboard with a modal window open. The modal displays a list of users from a specific bin. A large green arrow points from the bin ID in the modal back to the bin ID in the browser's address bar.

DASHBOARD

BIN ID: 61138ef2d5667e403a3fb6a1

We will be deployed in 1 minute for more info.

STATISTICS

1 Bins

GENERAL BINS (S)

1. 61138ef2d5667e403a3fb6a1

UTILITIES

JSON Store JSON Validator GeoIP Lookup Blog Change Log Contact Support

DETAILS **DELETE**

```
[{"name": "Jan", "mail": "jsj@aaaa.dk", "id": 1620639091625}, {"name": "Morten", "mail": "moab@aaaa.dk", "id": 1620658923021}, {"name": "Birgitte", "mail": "bki@aaaa.dk", "id": 1620659163593}, {"name": "Rasmus Cederdorff", "mail": "rasmus@cederdorff.com", "id": 1620659163594}]
```

BACK

READY TO CODE



READ USERS

- Use fetch to read data from your JSONBIN.
- Remember to inspect the data structure.
- Append users to the DOM.

```
async function loadUsers() {  
  const url = _baseUrl + "/latest"; // make sure to get the latest version  
  const response = await fetch(url, {  
    headers: _headers  
  });  
  const data = await response.json();  
  console.log(data);  
  _users = data.record;  
  appendUsers(_users);  
}
```

CREATE USER

- Create a form with inputs fields in your html.
- Declare a function creating a new user with properties and values from the input fields
- Required properties: id, name, age, gender, lookingFor, & image. Add more if you like: title, hobbies, mail, birthDate, etc.
- Make sure to update the DOM when the new user is created.

```
async function createUser() {
  showLoader(true);
  // references to input fields
  let nameInput = document.querySelector("#name");
  let mailInput = document.querySelector("#mail");
  // dummy generated user id
  const userId = Date.now();
  // declaring a new user object
  const newUser = {
    name: nameInput.value,
    mail: mailInput.value,
    id: userId
  };
  // pushing the new user object to the _users array
  _users.push(newUser);
  // wait for update
  await updateJSONBIN(_users);
}
```

UPDATE JSONBIN

- Create a function to update your JSONBIN.
- Use fetch to “PUT” your users array to JSONBIN (Fetch’s default method is “GET”).
- Use the response result to update the DOM (reuse `appendUsers(...)`).

```
async function updateJSONBIN(users) {  
  // put users array to jsonbin  
  const response = await fetch(_baseUrl, {  
    method: "PUT",  
    headers: _headers,  
    body: JSON.stringify(users)  
  });  
  // waiting for the result  
  const result = await response.json(); // the new updated users array
```

UPDATE USER

- Create some logic selecting a user when you click on the update button. You could use `Array.find()` to find the user by id in your global `_users` array.
- You need an update view with a form and inputs fields.
- Set the values in the update view with properties from selected user.
- Declare another function handling the logic of updating the user's properties. Again, find user by id in your global `_users` array. When `_users` array is updated, update JSONBIN and the DOM.

```
function selectUser(id) {  
  _selectedUserId = id;  
  // find user by given user id  
  const user = _users.find(user => user.id == _selectedUserId);  
  //...  
}  
  
async function updateUser() {  
  // references to input fields  
  const nameInput = document.querySelector("#name-update");  
  const mailInput = document.querySelector("#mail-update");  
  // find user to update by given user id  
  const userToUpdate = _users.find(user => user.id === _selectedUserId);  
  // update values of user in array  
  userToUpdate.name = nameInput.value;  
  userToUpdate.mail = mailInput.value;  
  // wait for update  
  await updateJSONBIN(_users);  
  //...  
}
```

DELETE USER

- Create some logic deleting the user when you click on the delete button. You can use `Array.find()` to remove user by id in your global `_users` array.
- When you have a `_users` array without the deleted user, update your JSONBIN.
- Extra: Display a confirm dialog before deleting user: <https://javascript.info/alert-prompt-confirm#confirm> or https://www.w3schools.com/jsref/met_win_confirm.asp
 - Instead of the default JS confirm, implement your own confirm modal, matching the styles of your SPA.

```
async function deleteUser(id) {  
  _users = _users.filter(user => user.id !== id);  
  await updateJSONBIN(_users);  
}
```

127.0.0.1:5502 says
Do you want to delete user?

Cancel OK

FILTER & SORT BY AND SEARCH

BACK

- Sort by name, age, gender &/ or lookingFor, etc.
- Filter by age, gender &/or lookingFor etc.
- Search field, searching on name
- Implement loader/ spinner.

Display on load, fetch and append.

```
function sortByName() {
  _users.sort((user1, user2) => {
    return user1.name.localeCompare(user2.name);
  });
  appendUsers(_users);
}

function search(value) {
  resetFilterByCourse();
  resetFilterByEnrollment();
  value = value.toLowerCase();
  const results = _users.filter(user => {
    const name = user.name.toLowerCase();
    if (name.includes(value)) {
      return user;
    }
  });
  appendUsers(results);
}
```

Projects:
spa-canvas-users
spa-headless-cms-movies
array-teachers-filter-keywords
array-teachers-search
array-teachers-search-keywords
etc.

DETAIL VIEW WITH MATCHES

BACK

- Detail view:
 - Display info about the user
 - Get (calculate) and display matches.
 - Style your dating SPA differently than race.js 
 - What happens when you refresh the page, standing in the detail view of your solution? Think of a way to improve the solution. Hint: localStorage or url params.
 - Make it possible to change the age range dynamically.
 - Do other improvements and consider how you can use all this knowledge in the interdisciplinary project.

```
function appendMatches(data) {
  let htmlTemplate = /*html*/ `

    <article class="selectedUser">
      <h3>${data.selectedUser.firstname} ${data.selectedUser.lastname}</h3>
      <p>Age: ${data.selectedUser.age}, Gender: ${data.selectedUser.gender}</p>
      <p>Number of matches: ${data.matchCount}</p>
      <button type="button" class="update" onclick="showUserUpdate(${data.selectedUser.id})">Update</button>
      <button type="button" class="delete" onclick="deleteUser(${data.selectedUser.id})">Delete</button>
    </article>You are welcome to get
  `;

  inspired by the existing
  project on GitHub and reuse
  the algorithm from users -
  match-backend or your
  </article>
`; mandatory assignment in
}

document.querySelector("#grid-matches").innerHTML = htmlTemplate;
document.querySelector("#user .title").innerHTML = data.selectedUser.firstname;
showLoader(false);
}

// ===== CREATE =====
/** 
 * creates a new user and saving in JSON file using the php backend service
 */
asvnc function createUser(firstname, lastname, age, haircolor, countryName)
```

PHP BACKEND SERVICE

BACK

Instead of using JSONBIN, implement a PHP Backend Service reading and writing to a JSON file or a MySQL Database. The PHP Backend Service must include the following functionality:

- Get and read all data as JSON (GET)
- Create new objects, saved and stored in JSON file or database. The new objects must be posted from the frontend as JSON (POST). Remember to generate `id` for new objects.
- Update existing object by the `id`. Use PUT to put the updates to the PHP Backend service and save updates in JSON file or database.
- Delete object by the `id`. Use DELETE to delete objects and save changes in JSON file or database.
- Extra: Implement upload of images as file instead of URL.

Inspiration: [users-match-frontend](#), [users-match-backend](#) and [file-upload](#)

IT'S ALL
OBJECTS &
ARRAYS

BACK

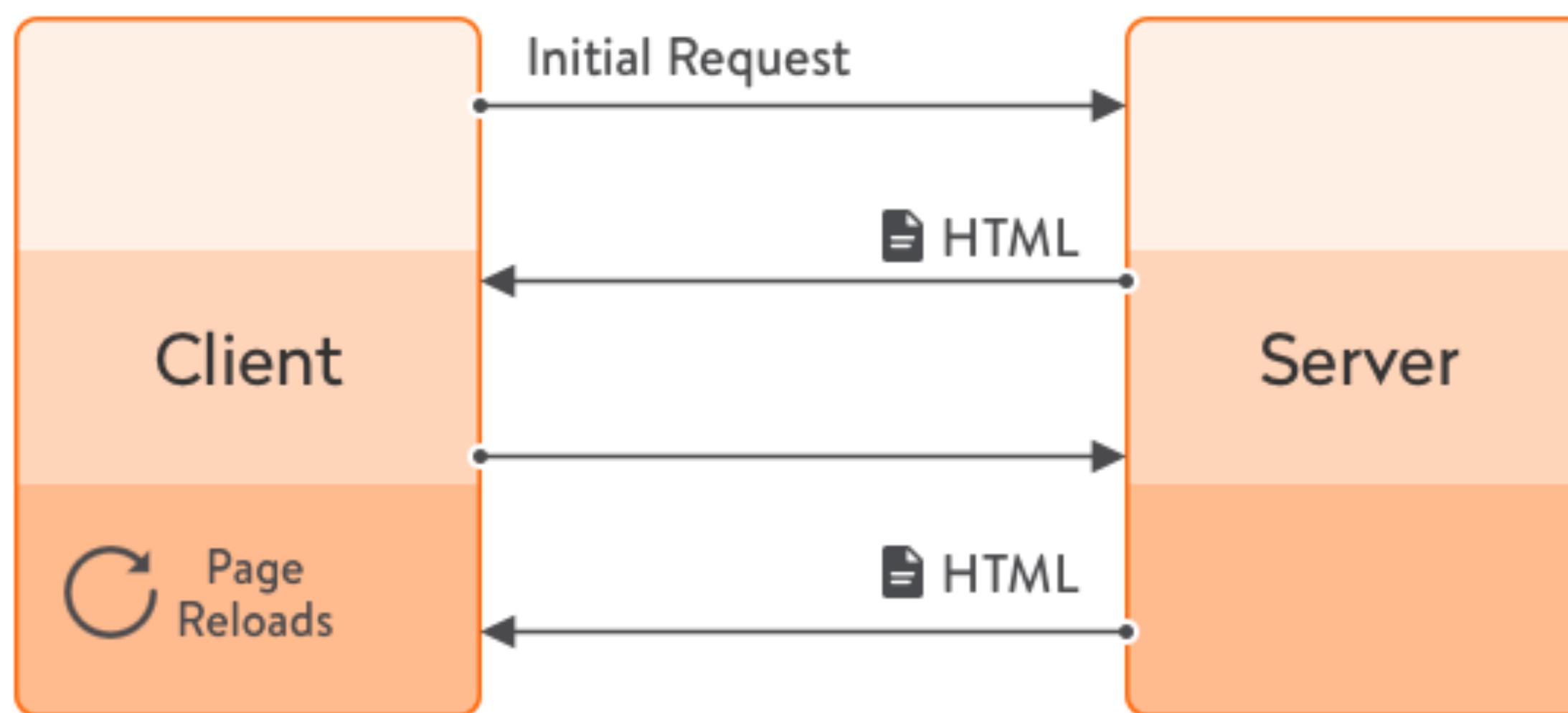
SINGLE PAGE APPLICATION & ROUTER

SINGLE PAGE APPLICATION

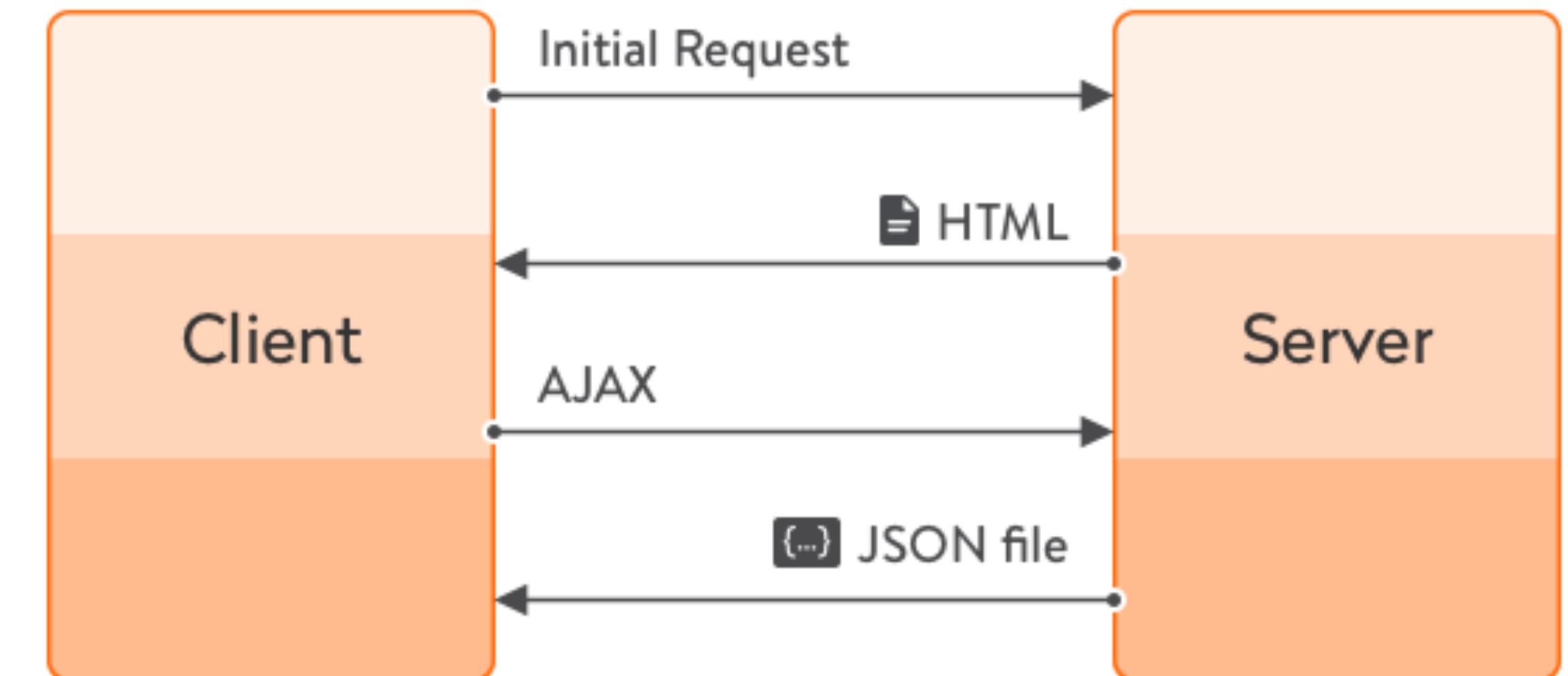
“A single-page application is an application that loads a single HTML page and all the necessary assets (such as JavaScript and CSS) required for the application to run. Any interactions with the page or subsequent pages do not require a round trip to the server which means the page is not reloaded.”

<https://reactjs.org/docs/glossary.html#single-page-application>

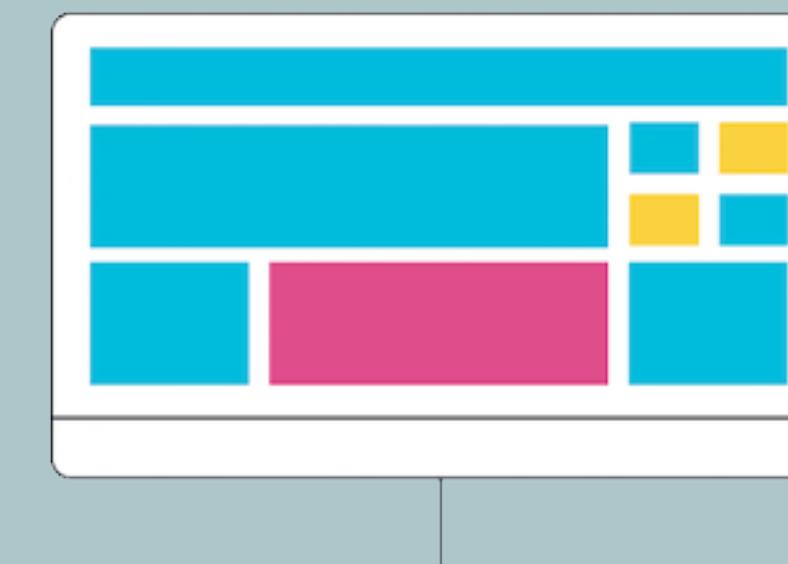
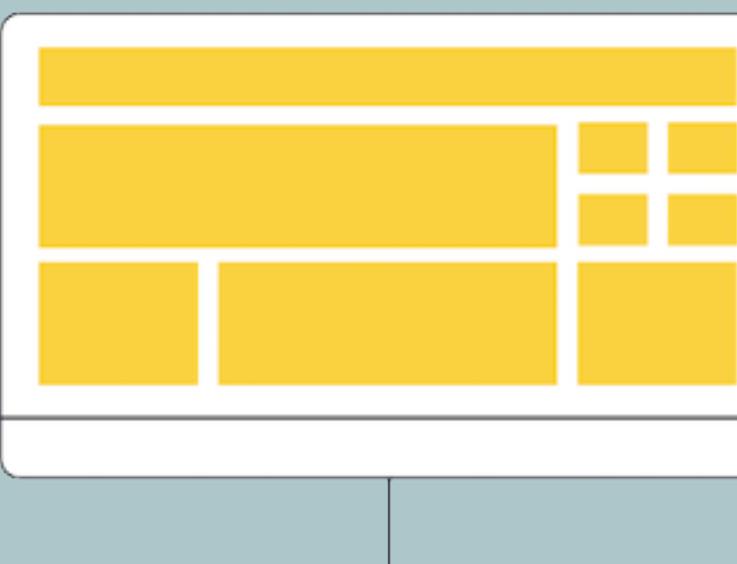
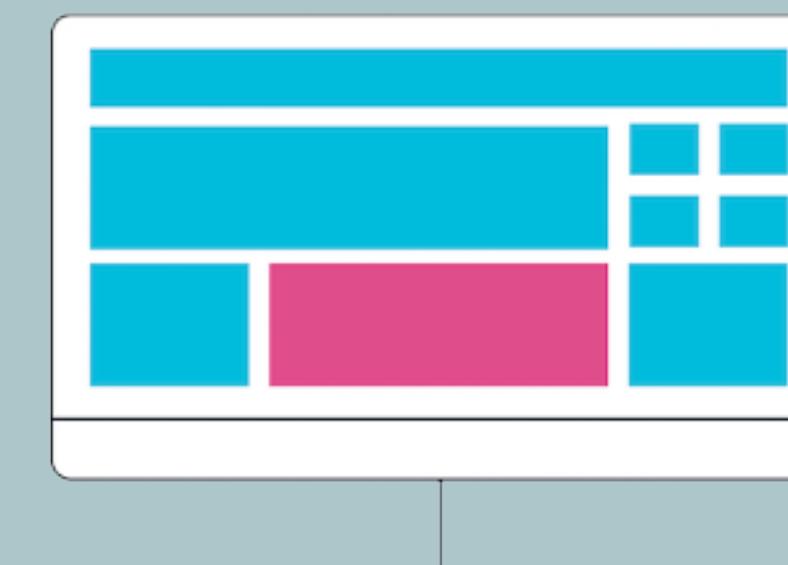
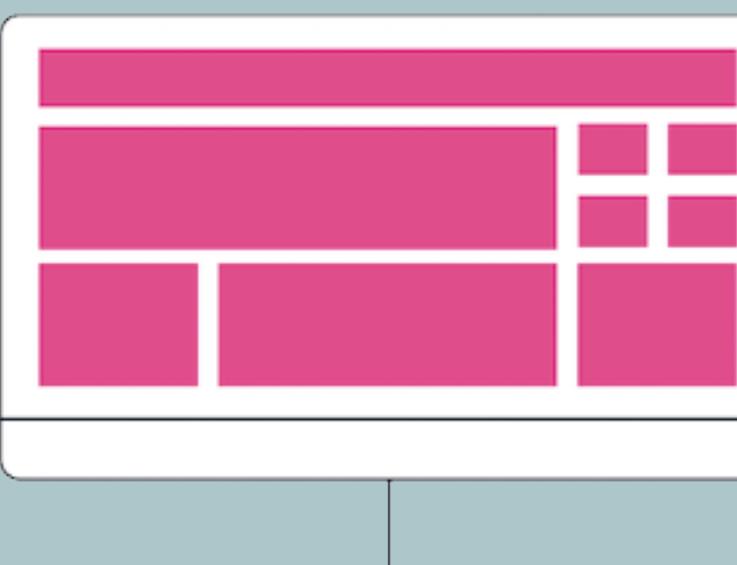
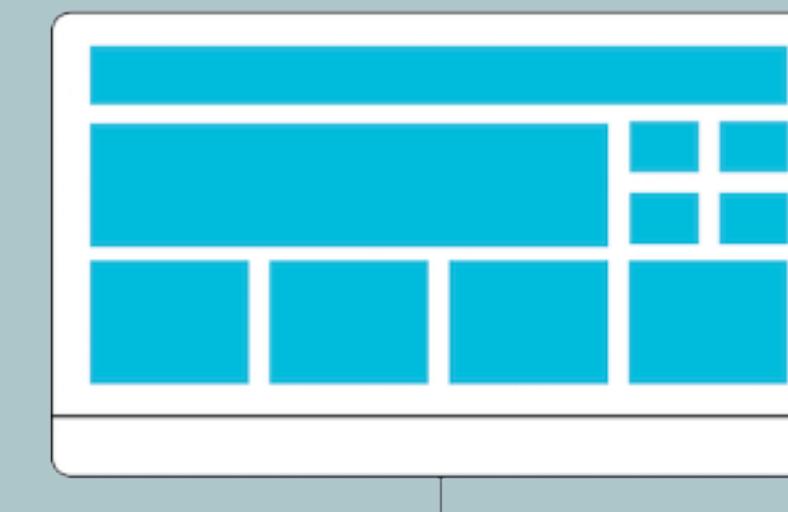
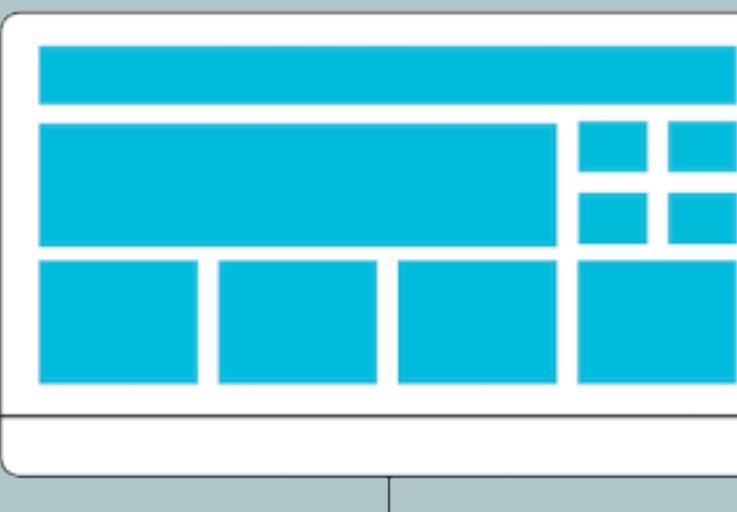
Multi-page app lifecycle



Single-page app lifecycle



M P A V S S P A



A close-up photograph of a person's hands interacting with a black wireless router. One hand holds a blue Ethernet cable and is in the process of plugging it into one of the four yellow LAN ports on the front panel of the router. The router also features three black external antennas. The background is blurred.

WHAT IS A
ROUTER?

ROUTER

“It is the piece of software in charge to organize the states of the application, switching between different views.”

It's a key component in Single Page Applications.

https://medium.com/@fro_g/routing-in-javascript-d552ff4d2921

SPA & ROUTER

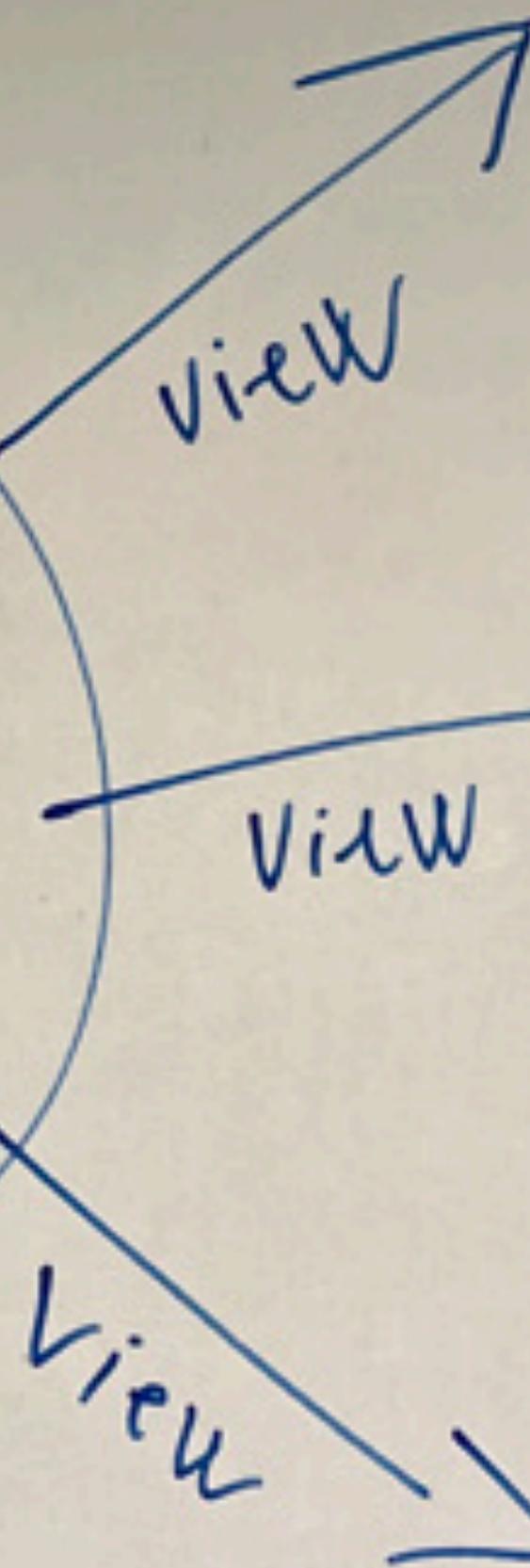
In an SPA the router is in charge of displaying the right view by watching changes on the URL.

The router makes it look like the web app has multiple pages and HTML files. But it's just changing the view, what is displayed, inside of the single `index.html` file. The page transition and change of content, is done dynamically by JavaScript (`router.js`).



/create
route

Router
(router.js)



users
page

create
page

update
page

[www.webpage.com/ pathname](http://www.webpage.com/pathname)



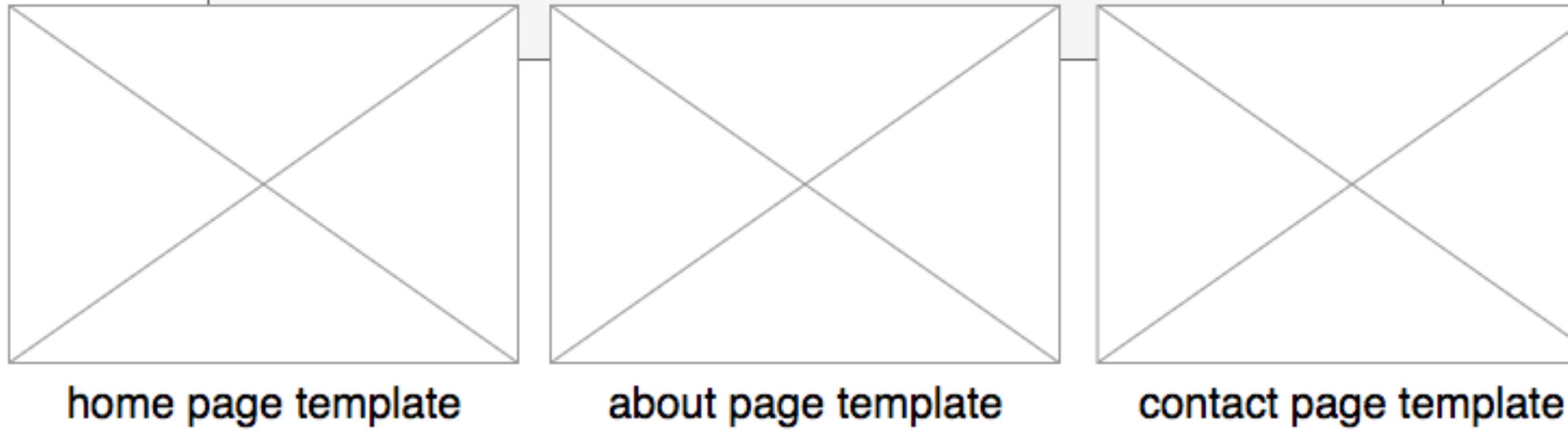
<div id="content"/>

populate content div based on url pathname

/home

/about

/contact



```
✓ vanilla-js-spa-router-hash
  > css
  > img
  < JS app.js
  < JS router.js
```

```
<> index.html
```

```
* All routes of the SPA
* "path": "id of page in DOM"
*/
const _routes = {
  "#/": "home",
  "#/about": "about",
  "#/clients": "clients",
  "#/contact": "contact"
};
const _pages = document.querySelectorAll(".page");
const _basePath = location.pathname.replace("index.html", "");
const _navLinks = document.querySelectorAll("nav a");

/**
 * Changing display to none for all pages
 */
function hideAllPages() {
  for (const page of _pages) {
    page.style.display = "none";
  }
}

/**
 * Navigating SPA to specific page by given path
 */
function navigateTo(path) {
  window.history.pushState({}, path, _basePath + path);
  showPage(path);
}

/**
 * Displaying page by given path
*/
function showPage(path) {
```

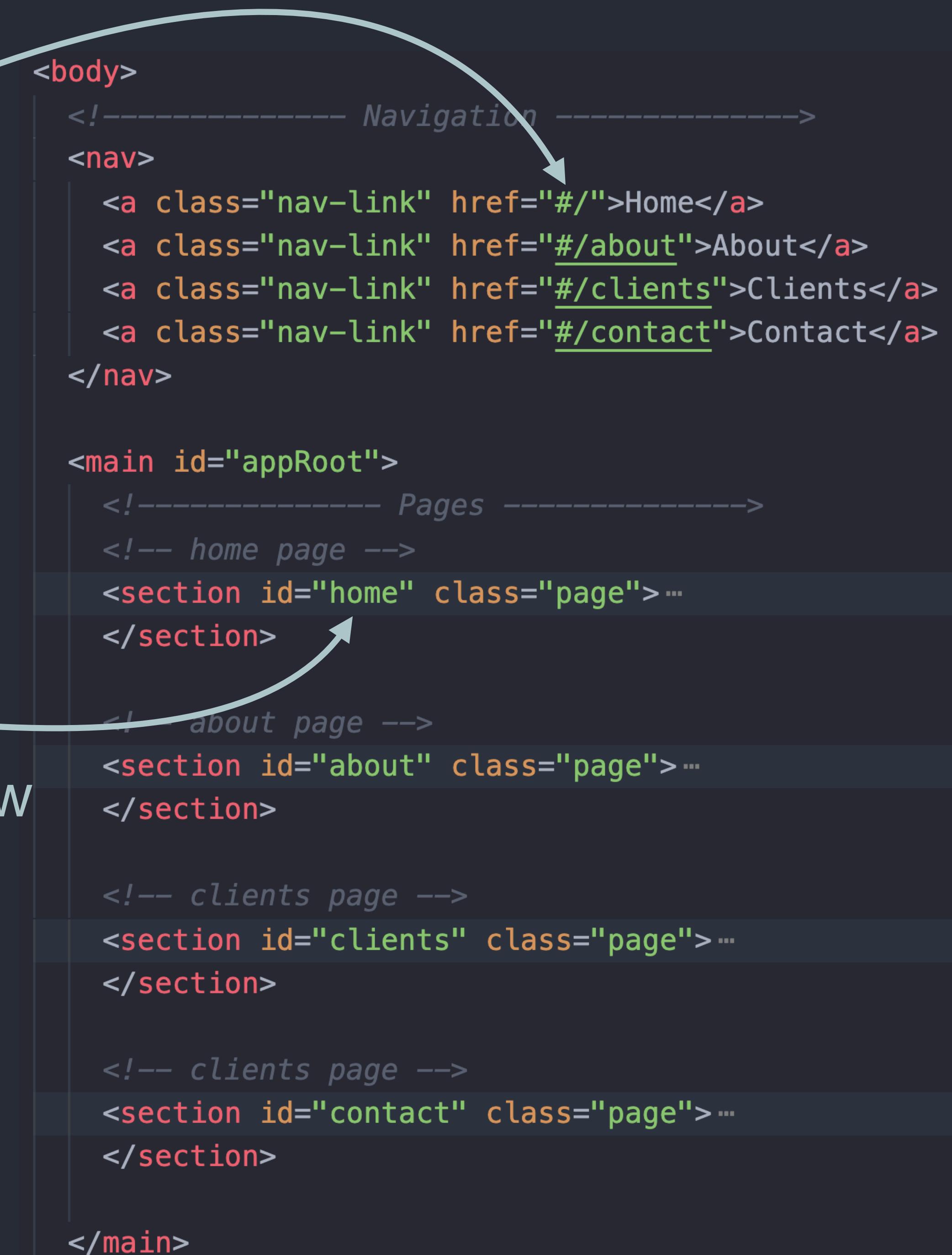
✓ vanilla-js-spa-router-hash

```
const routes = {  
  "/": "home",  
  "/about": "about",  
  "/clients": "clients",  
  "/contact": "contact"  
};
```

route

id of view

```
<body>  
  <!----- Navigation ----->  
  <nav>  
    <a class="nav-link" href="#/">Home</a>  
    <a class="nav-link" href="#/about">About</a>  
    <a class="nav-link" href="#/clients">Clients</a>  
    <a class="nav-link" href="#/contact">Contact</a>  
  </nav>  
  
  <main id="appRoot">  
    <!----- Pages ----->  
    <!-- home page -->  
    <section id="home" class="page">...</section>  
    <!-- about page -->  
    <section id="about" class="page">...</section>  
    <!-- clients page -->  
    <section id="clients" class="page">...</section>  
    <!-- contact page -->  
    <section id="contact" class="page">...</section>  
  </main>
```



change default event for all .nav-link items

```
<!------- Navigation ----->
<nav>
  <a class="nav-link" href="#/">Home</a>
  <a class="nav-link" href="#/about">About</a>
  <a class="nav-link" href="#/clients">Clients</a>
  <a class="nav-link" href="#/contact">Contact</a>
</nav>
```

```
/** 
 * Attaching event to nav links and preventing default anchor
 */
function attachNavLinkEvents() {
  const navLinks = document.querySelectorAll(".nav-link");
  for (const link of navLinks) {
    link.addEventListener("click", function (event) {
      const path = link.getAttribute("href");
      navigateTo(path);
      event.preventDefault();
    });
  }
}
```

when a `.nav-link` item is clicked,
`navigateTo(...)` is called with the value of `href`

```
<!------- Navigation ----->
<nav>
  <a class="nav-link" href="#/">Home</a>
  <a class="nav-link" href="#/about">About</a>
  <a class="nav-link" href="#/clients">Clients</a>
  <a class="nav-link" href="#/contact">Contact</a>
</nav>
```

```
/** 
 * Navigating SPA to specific page by given path
 */
function navigateTo(path) {
  window.history.pushState({}, path, _basePath + path);
  showPage(path);
}

/** 
 * Displaying page by given path
 */
function showPage(path) {
  hideAllPages(); // hide all pages
  document.querySelector(`#${_routes[path]}`).style.display = "block";
  setActiveTab(path);
}
```

SPA & ROUTER EXERCISE

BACK

- Inspect the `index.html` & `router.js` in `vanilla-js-spa-router-hash`
- Add another page called users.
- Remember to add a new section for users page, a nav-link and the route in `router.js`.
- Test you are able to navigate to the users page.
- Change the clients page to a posts page.
Remember to update id's, href & routes.
- Test your solution.
- Explain `navigateTo()`, `showPage()` & `initRouter()`

```
✓ vanilla-js-spa-router-hash
  > css
  > img
  ✓ js
    JS app.js
    JS router.js
  <> index.html
```

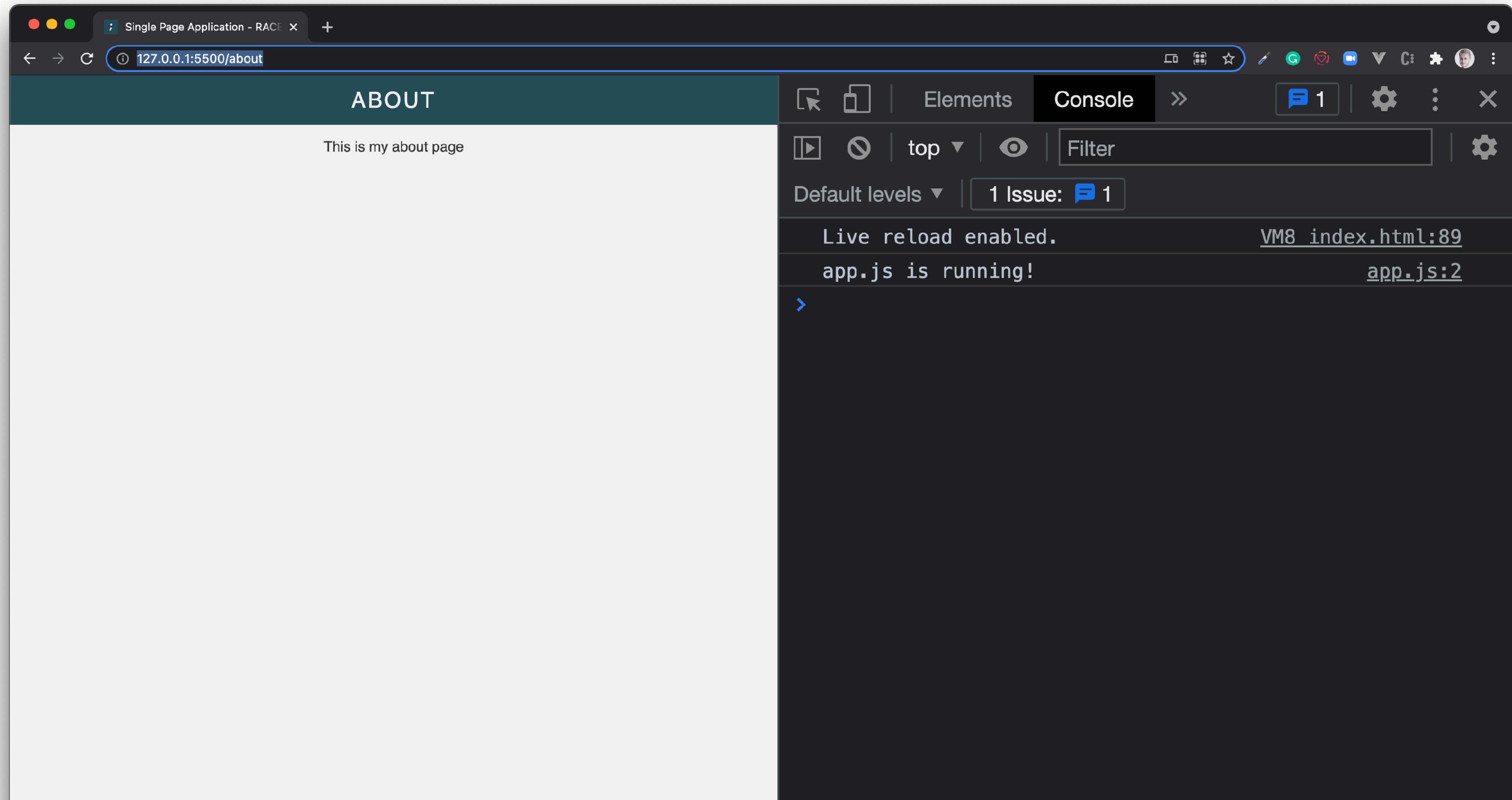
A file tree diagram showing the directory structure for the SPA exercise. The root folder is "vanilla-js-spa-router-hash". It contains three main subfolders: "css", "img", and "js". The "js" folder contains two files: "app.js" and "router.js". The "index.html" file is shown at the bottom level, indicated by a double-headed arrow symbol (<>) between the "js" folder and the file itself, suggesting it is a shared or generated file.

```
✓ vanilla-js-spa-router
  > css
  > img
  ✓ js
    JS app.js
    JS router.js
  <> index.html

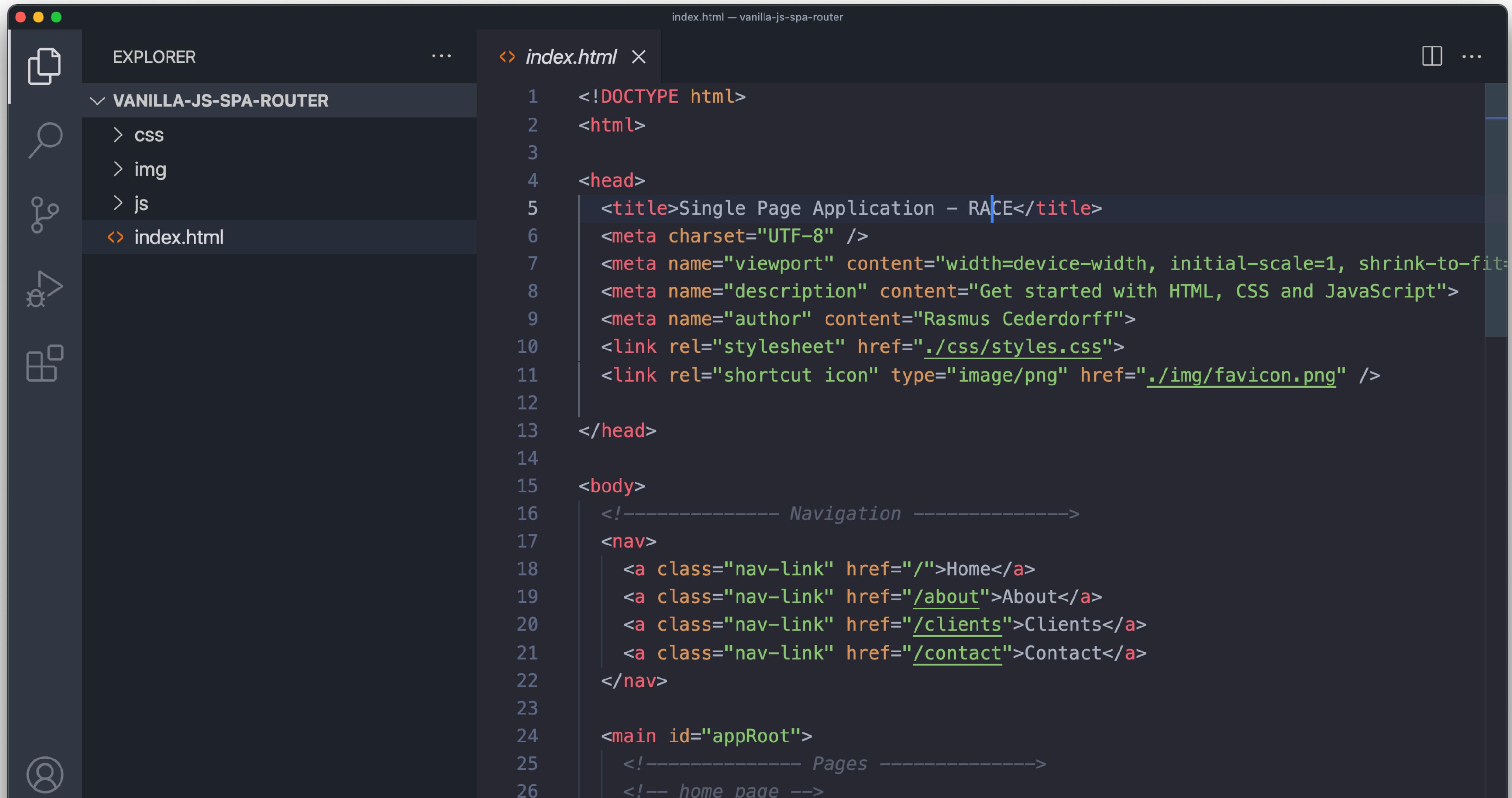
✓ vanilla-js-spa-router-hash
  > css
  > img
  ✓ js
    JS app.js
    JS router.js
  <> index.html
```

WHAT THE
“#/HASH”?

WITHOUT HASH



OPEN & RUN THIS PROJECT ONLY

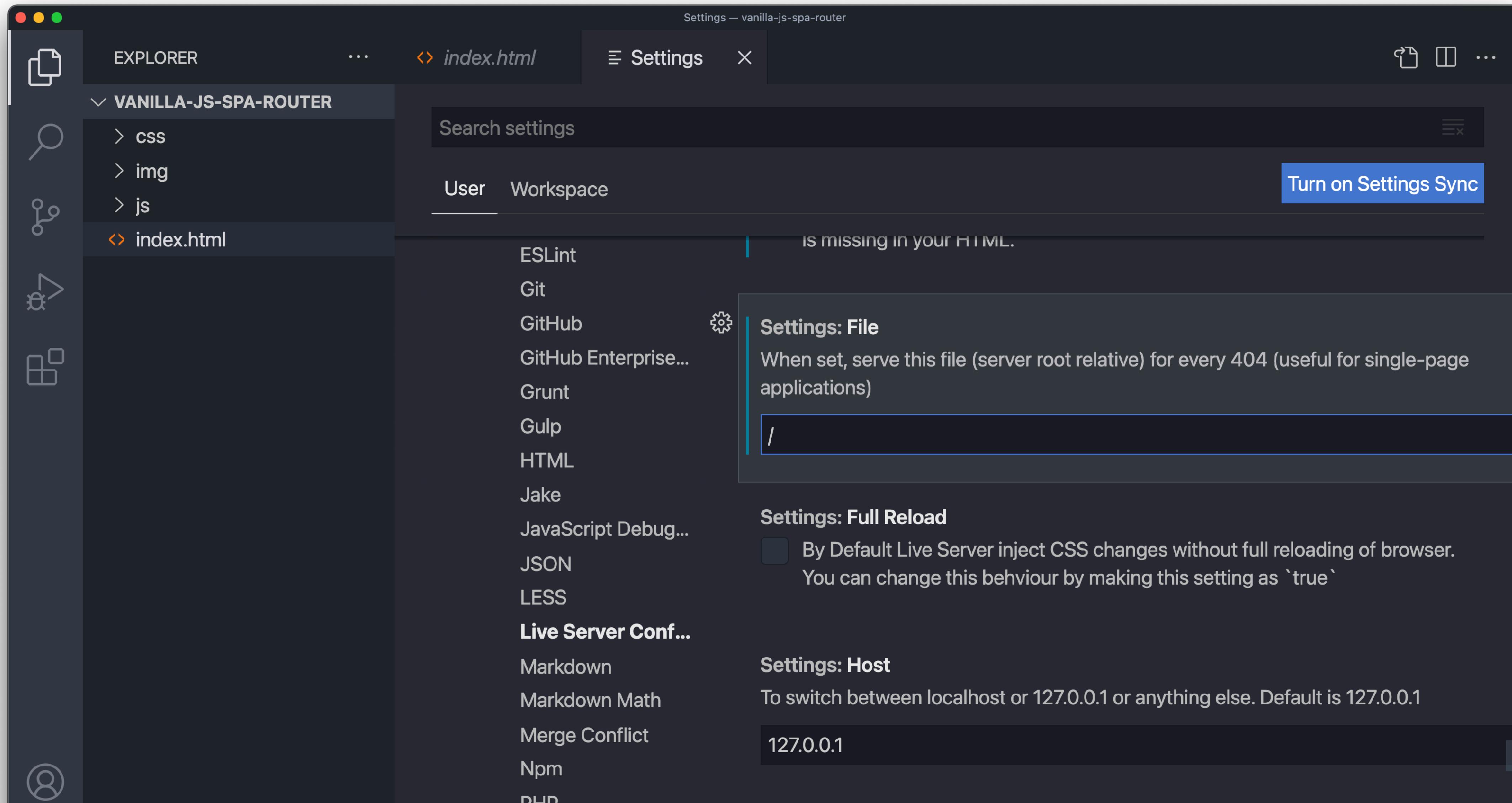


The screenshot shows a dark-themed code editor interface with the following details:

- Explorer Bar:** On the left, it shows a tree view of the project structure under "VANILLA-JS-SPA-ROUTER". The visible items are "css", "img", "js", and "index.html".
- Editor Area:** The main area displays the content of "index.html".
- Code Content:**

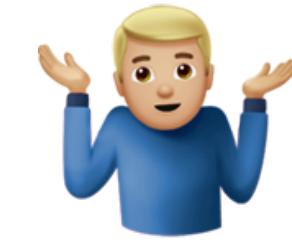
```
index.html — vanilla-js-spa-router
<!DOCTYPE html>
<html>
  <head>
    <title>Single Page Application - RACE</title>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=1" />
    <meta name="description" content="Get started with HTML, CSS and JavaScript">
    <meta name="author" content="Rasmus Cederdorff">
    <link rel="stylesheet" href="./css/styles.css">
    <link rel="shortcut icon" type="image/png" href="./img/favicon.png" />
  </head>
  <body>
    <!-- Navigation -->
    <nav>
      <a class="nav-link" href="/">Home</a>
      <a class="nav-link" href="/about">About</a>
      <a class="nav-link" href="/clients">Clients</a>
      <a class="nav-link" href="/contact">Contact</a>
    </nav>
    <main id="appRoot">
      <!-- Pages -->
      <!-- home page -->
    </main>
  </body>
</html>
```

CHANGE LIVE SERVER ROOT



REQUIRES SOME CONFIG WHEN UPLOAD
TO SERVER/ OWN DOMAIN:
ALWAYS LOOK AT ROOT OR INDEX.HTML

OR USE HASH



The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists a project folder named "VANILLA-JS-SPA-ROUTER-HASH" containing "css", "img", "js", and "index.html". The main area displays the "index.html" file content:

```
index.html — vanilla-js-spa-router-hash
<!DOCTYPE html>
<html>
<head>
<title>Single Page Application – RACE</title>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />
<meta name="description" content="Get started with HTML, CSS and JavaScript">
<meta name="author" content="Rasmus Cederdorff">
<link rel="stylesheet" href="./css/styles.css">
<link rel="shortcut icon" type="image/png" href="./img/favicon.png" />
</head>
<body>
<!-- Navigation -->
<nav>
<a class="nav-link" href="#/">Home</a>
<a class="nav-link" href="#/about">About</a>
<a class="nav-link" href="#/clients">Clients</a>
<a class="nav-link" href="#/contact">Contact</a>
</nav>
<main id="appRoot">
<!-- Pages -->
<!-- home page -->
```

BACK

FILTER, SORT & SEARCH

ARRAY METHODS

.PUSH()

.FIND()

.MAP()

.REDUCE()

.FILTER()

.SORT()

.CONCAT()

...

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>

LOOPS

```
for (const familyMember of familyMembers) {  
  console.log(familyMember);  
}
```

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

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

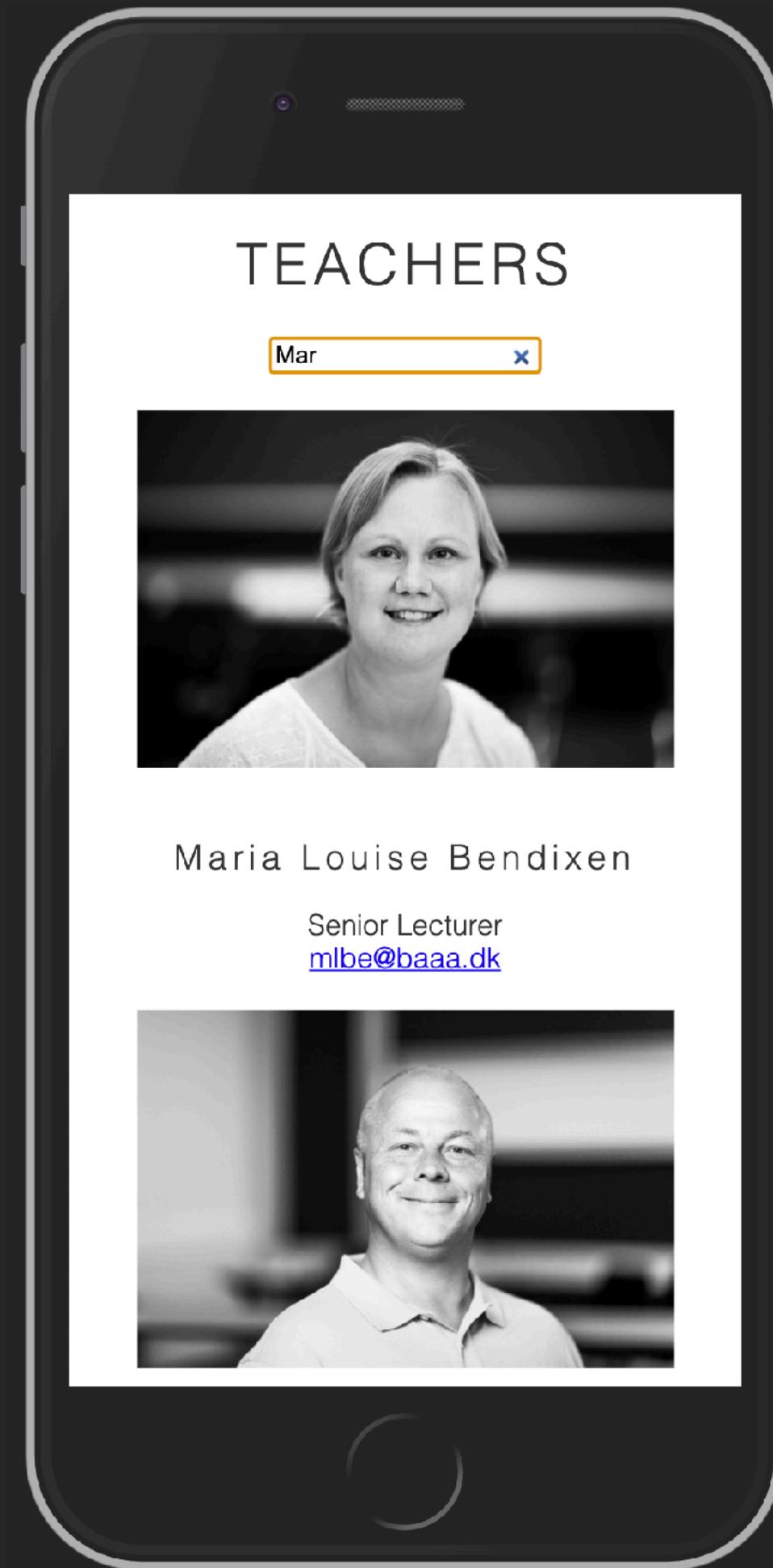
<https://javascript.info/array#loops>

<https://javascript.info/while-for>

LOOPS

... SEARCH

```
function search(value) {  
    console.log(value);  
    let filteredTeachers = [];  
    for (let teacher of teachers) {  
        let name = teacher.name.toLowerCase();  
        if (name.includes(value.toLowerCase())) {  
            filteredTeachers.push(teacher);  
        }  
    }  
    console.log(filteredTeachers);  
    appendTeachers(filteredTeachers);  
}
```



array-teachers-search

ARRAY

.FILTER(...)

```
let users = [
  { age: 35, name: "John" },
  { age: 40, name: "Pete" },
  { age: 44, name: "Mary" }
];
```

// returns array of with users older than 39

```
let someUsers = users.filter(item => item.age > 39);
```

```
console.log(someUsers);
```

▼ Array(2) ⓘ

- ▶ 0: {age: 40, name: "Pete"}
- ▶ 1: {age: 44, name: "Mary"}

length: 2

ARRAY METHODS

<https://javascript.info/array-methods#filter>

- Chapter
- Data types
- Lesson navigation
- Add/remove items
- Iterate: forEach
- Searching in array**
- Transform an array
- Array.isArray
- Most methods support "thisArg"
- Summary
- Tasks (13)
- Comments
- Share
- [Edit on GitHub](#)
- Ads



filter

The `find` method looks for a single (first) element that makes the function return `true`. If there may be many, we can use `arr.filter(fn)`.

The syntax is similar to `find`, but `filter` returns an array of all matching elements:

```
1 let results = arr.filter(function(item, index, array) {  
2   // if true item is pushed to results and the iteration continues  
3   // returns empty array if nothing found  
4});
```

For instance:

```
1 let users = [  
2   {id: 1, name: "John"},  
3   {id: 2, name: "Pete"},  
4   {id: 3, name: "Mary"}  
5];  
6  
7 // returns array of the first two users  
8 let someUsers = users.filter(item => item.id < 3);  
9  
10 alert(someUsers.length); // 2
```

Transform an array

Let's move on to methods that transform and reorder an array.

map

The `arr.map` method is one of the most useful and often used. It calls the function for each element of the array and returns the array of results.

FOR OF LOOP

```
function search(value) {  
    value = value.toLowerCase();  
    let filteredTeachers = [];  
    for (let teacher of teachers) {  
        const name = teacher.name.toLowerCase();  
        if (name.includes(value)) {  
            filteredTeachers.push(teacher);  
        }  
    }  
    appendTeachers(filteredTeachers);  
}
```

.FILTER & ARROW FUNCTION

```
function search(value) {  
    value = value.toLowerCase();  
    const filteredTeachers = teachers.filter(teacher => teacher.name.toLowerCase().includes(value));  
    appendTeachers(filteredTeachers);  
}
```

JavaScript Async

Products

127.0.0.1:5501/spa-products-fetch-json-enhanced/index.html#products

PRODUCTS

Order by: Choose here ▾ Show out of stock ✓ Search



MacBook Pro 13"

Apple

Price: 11799 kr.

Status: outOfStock

[EDIT](#) [DELETE](#)



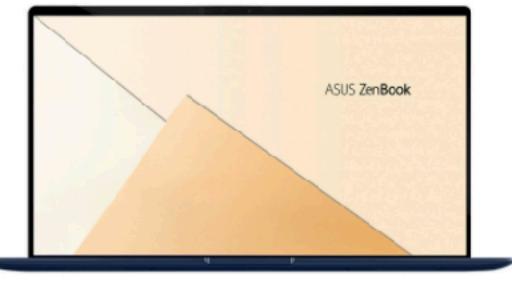
MacBook Pro 15"

Apple

Price: 21499 kr.

Status: inStock

[EDIT](#) [DELETE](#)



ASUS ZenBook

PRODUCTS ADD PRODUCT

Elements Console Sources »

```
<section id="products" class="page" style="display: block;">...

Order by:

... Show out of stock





MacBook Pro 13"



Apple



Price: 11799 kr.



Status: outOfStock



EDIT DELETE





MacBook Pro 15"



Apple



Price: 21499 kr.

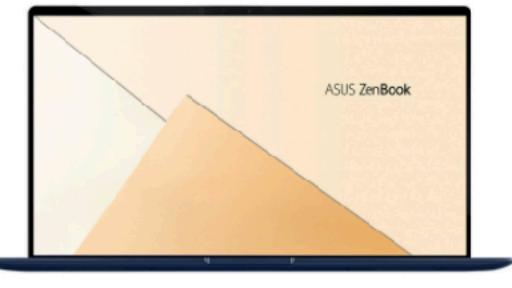


Status: inStock



EDIT DELETE





ASUS ZenBook



PRODUCTS ADD PRODUCT


```

Styles Computed Layout Event Listeners DOM Breakpoints »

Filter :hov .cls +

```
element.style {  
}  
  
select {  
    background-color: var(--green);  
    border: none;  
    color: var(--text-color-light);  
    padding: 1em;  
    margin: 1em;  
    max-width: 350px;  
}  
  
select:not(:-internal-list-box) {  
    overflow: visible !important;  
}
```

main.css:366 user agent stylesheet

SORT

```
function orderBy(option) {
  if (option === "brand") {
    orderByBrand();
  } else if (option === "model") {
    orderByModel();
  } else if (option === "price") {
    orderByPrice();
  }
}

function orderByBrand() {
  _products.sort((product1, product2) => {
    return product1.brand.localeCompare(product2.brand);
  });
  appendProducts(_products);
}
```

```
function orderByBrand() {  
  _products.sort((product1, product2) => {  
    return product1.brand.localeCompare(product2.brand);  
  });  
  appendProducts(_products);  
}
```

```
function orderByModel() {  
  _products.sort((product1, product2) => {  
    return product1.model.localeCompare(product2.model);  
  });  
  appendProducts(_products);  
}
```

```
function orderByPrice() {  
  _products.sort((product1, product2) => {  
    return product1.price - product2.price;  
  });  
  appendProducts(_products);  
}
```

.SORT & ARROW FUNCTIONS
INSIDE FUNCTIONS DECLARATIONS

JavaScript Async

Products

127.0.0.1:5501/spa-products-fetch-json-enhanced/index.html#products

label.checkmark-container 178.09x72

Order by: Choose here ▾ Show out of stock ✓ Search

MacBook Pro 13"

Apple

Price: 11799 kr.

Status: outOfStock

EDIT DELETE

MacBook Pro 15"

Apple

Price: 21499 kr.

Status: inStock

EDIT DELETE

ASUS ZenBook

PRODUCTS ADD PRODUCT

Elements Console Sources »

label.checkmark-container

<header class="topbar">...</header>

<section class="tools-grid"> grid

<label for="sortBy">

"Order by: "

<select id="sortBy" onchange="orderBy(this.value)">...</select>

</label>

<label for="outOfStock" class="checkmark-container">

"Show out of stock "

<input type="checkbox" id="outOfStock" onchange="showHideOutStock(this.checked)" checked> == \$0

...

</label>

<input type="search" placeholder="Search" onkeyup="search(this.value)" onsearch="search('')">

... page section.tools-grid label.checkmark-container input#outOfStock ...

Styles Computed Layout Event Listeners DOM Breakpoints »

Filter :hov .cls +

element.style {

}

.checkmark-container input {

position: absolute;

opacity: 0;

cursor: pointer;

height: 0;

width: 0;

}

input {

margin: 1em auto;

width: 100%;

max-width: 350px;

main.css:313

main.css:247

ARROW FUNCTIONS

```
function orderByBrand() {  
  _products.sort(function (product1, product2) {  
    return product1.brand.localeCompare(product2.brand);  
  });  
  appendProducts(_products);  
}
```

FUNCTION DECLARATION

```
function orderByBrand() {  
  _products.sort((product1, product2) => {  
    return product1.brand.localeCompare(product2.brand);  
  });  
  appendProducts(_products);  
}
```

ARROW FUNCTION

```
function orderByBrand() {  
  _products.sort((product1, product2) => product1.brand.localeCompare(product2.brand));  
  appendProducts(_products);  
}
```

ONE LINE ARROW FUNCTION

Array methods

← → C 🔒 javascript.info/array-methods

☰

Chapter

Data types

Lesson navigation

Add/remove items

Iterate: forEach

Searching in array

Transform an array

Array.isArray

Most methods support "thisArg"

Summary

Tasks (13)

Comments

Share

Edit on GitHub

Ad by Carbon

Byg og implementer Node.js-, Java- og Python-apps med Azure.

filter

The `find` method looks for a single (first) element that makes the function return `true`. If there may be many, we can use `arr.filter(fn)`.

The syntax is similar to `find`, but `filter` returns an array of all matching elements:

```
1 let results = arr.filter(function(item, index, array) {  
2   // if true item is pushed to results and the iteration continues  
3   // returns empty array if nothing found  
4 });
```

For instance:

```
1 let users = [  
2   {id: 1, name: "John"},  
3   {id: 2, name: "Pete"},  
4   {id: 3, name: "Mary"}  
5 ];  
6  
7 // returns array of the first two users  
8 let someUsers = users.filter(item => item.id < 3);  
9  
10 alert(someUsers.length); // 2
```

Transform an array

Let's move on to methods that transform and reorder an array.

map

The `arr.map` method is one of the most useful and often used.

It calls the function for each element of the array and returns the array of results.

FILTER

```
function showHideOfStock(checked) {  
  if (checked) {  
    appendProducts(_products);  
  } else {  
    const inStockProducts = _products.filter(product => product.status === "inStock");  
    appendProducts(inStockProducts);  
  }  
}
```

SEARCH

```
let _products = [];  
  
function search(value) {  
    let searchQuery = value.toLowerCase();  
    let filteredProducts = [];  
    for (let product of _products) {  
        let model = product.model.toLowerCase();  
        let brand = product.brand.toLowerCase();  
        if (model.includes(searchQuery) || brand.includes(searchQuery)) {  
            filteredProducts.push(product);  
        }  
    }  
    appendProducts(filteredProducts);  
}
```

SEARCH

```
function search(value) {  
    let searchQuery = value.toLowerCase();  
    let filteredProducts = [];  
    for (let product of _products) {  
        let model = product.model.toLowerCase();  
        let brand = product.brand.toLowerCase();  
        if (model.includes(searchQuery) || brand.includes(searchQuery)) {  
            filteredProducts.push(product);  
        }  
    }  
    appendProducts(filteredProducts);  
}  
  
function search(value) {  
    let searchQuery = value.toLowerCase();  
    let filteredProducts = _products.filter(product => {  
        let model = product.model.toLowerCase();  
        let brand = product.brand.toLowerCase();  
        if (model.includes(searchQuery) || brand.includes(searchQuery)) {  
            return product;  
        }  
    }  
);  
appendProducts(filteredProducts);  
}
```

String.prototype.includes() - JS | developer.mozilla.org

MDN Web Docs

Technologies References & Guides Feedback

Site search... (Press "/" to focus)

Web technology for developers > JavaScript > JavaScript reference > Standard built-in objects > String > String.prototype.includes()

Change language

Table of contents

- Syntax
- Description
- Polyfill
- Examples
- Specifications
- Browser compatibility
- See also

Related Topics

- Standard built-in objects
 - String
 - Properties
 - String length
 - Methods
 - String.prototype[`@@iterator`]()
 - String.prototype.anchor()
 - String.prototype.at()
 - String.prototype.big()
 - String.prototype.blink()
 - String.prototype.bold()

String.prototype.includes()

The `includes()` method performs a case-sensitive search to determine whether one string may be found within another string, returning `true` or `false` as appropriate.

JavaScript Demo: String.includes()

```
1 const sentence = 'The quick brown fox jumps over the lazy dog.';  
2  
3 const word = 'fox';  
4  
5 console.log(`The word "${word}" ${sentence.includes(word) ? 'is' : 'is not'} in the sentence`);  
6 // expected output: "The word "fox" is in the sentence"  
7
```

Run >

Reset

Syntax

```
includes(searchString)  
includes(searchString, position)
```

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



NEXT UP

OBJECT-ORIENTED JS

CLASSES AND OBJECTS

MODULES, IMPORT & EXPORT

COMPONENTS