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.

AGENDA

JSON & DATA STRUCTURES

WRAP UP: CANVAS USERS SPA

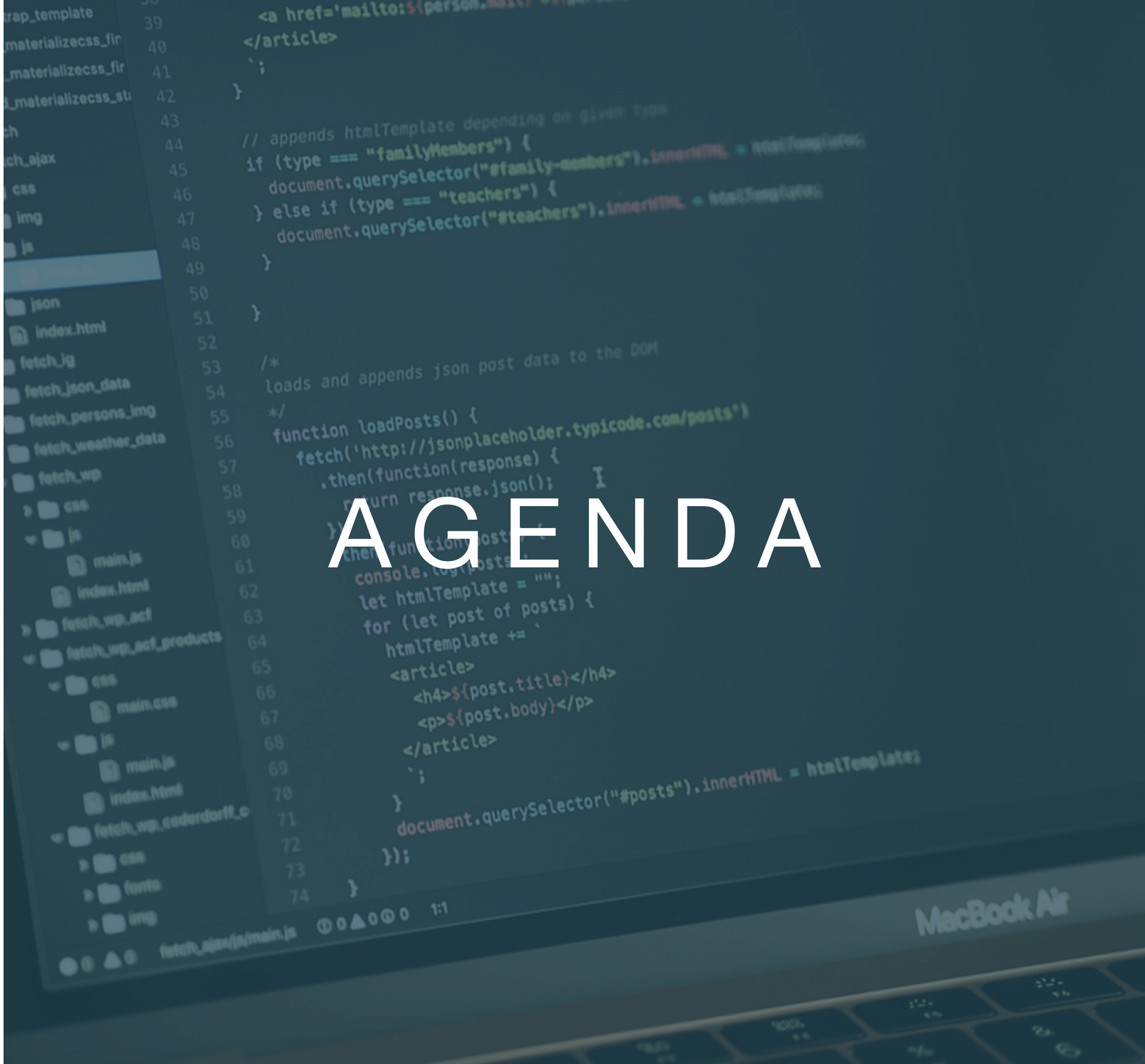
ASYNC JS: FETCH, HEADERS, HTTP METHODS, BODY & API

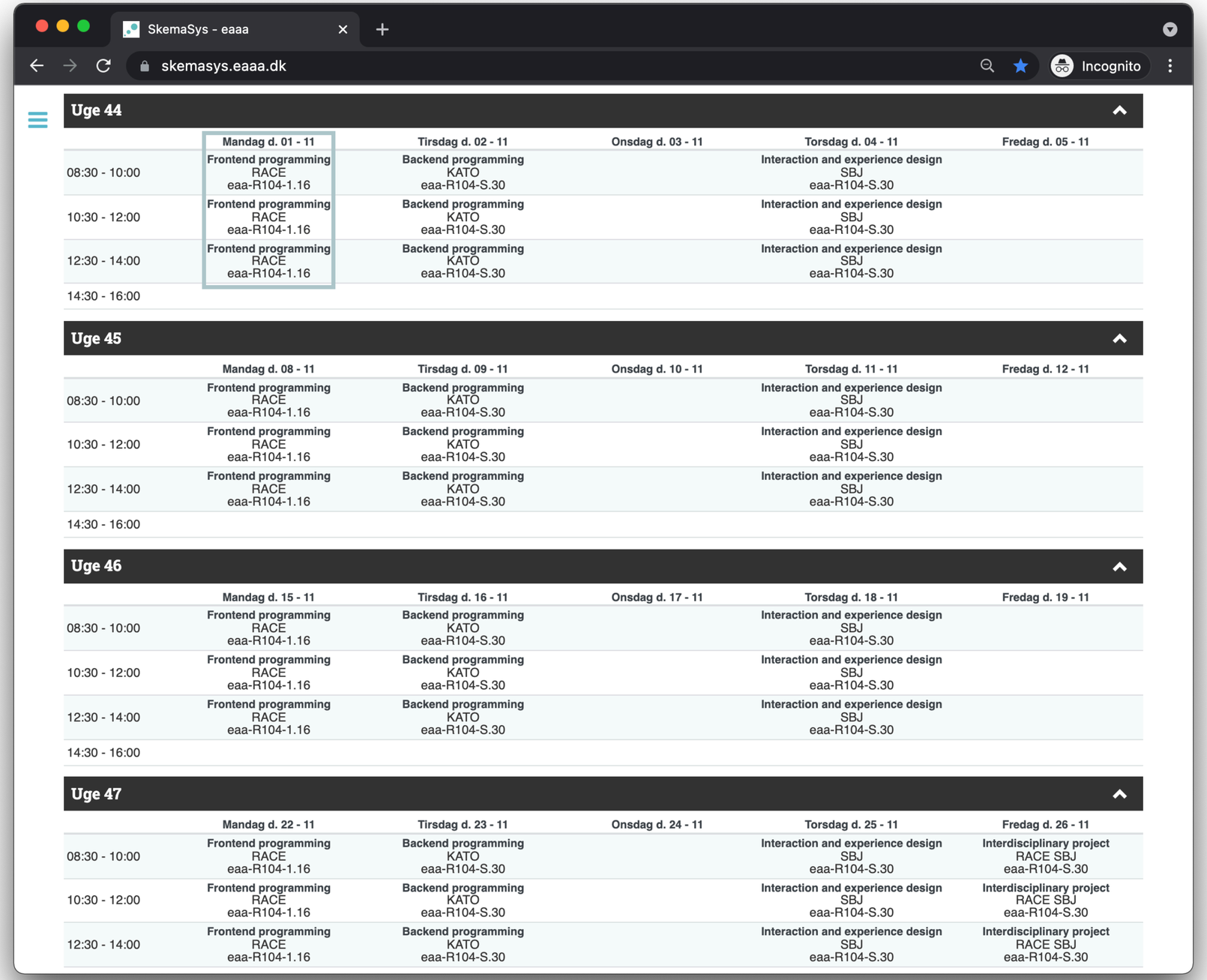
CASE: DATING SPA

10.07: SPA & ROUTER

11.21: FILTER, SORT & SEARCH

12.30: 2ND SEMESTER & ELECTIVES





Uge 44

	Mandag d. 01 - 11	Tirsdag d. 02 - 11	Onsdag d. 03 - 11	Torsdag d. 04 - 11	Fredag d. 05 - 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	Fredag d. 12 - 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	Fredag d. 19 - 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	Fredag d. 26 - 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	**Interdisciplinary project** RACE 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	**Interdisciplinary project** RACE 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	**Interdisciplinary project** RACE SBJ eaa-R104-S.30

BACK

JSON & DATA STRUCTURES

JSON

JAVA**S**CRIPT **O**BJECT **N**OTATION

... A SYNTAX FOR STORING &

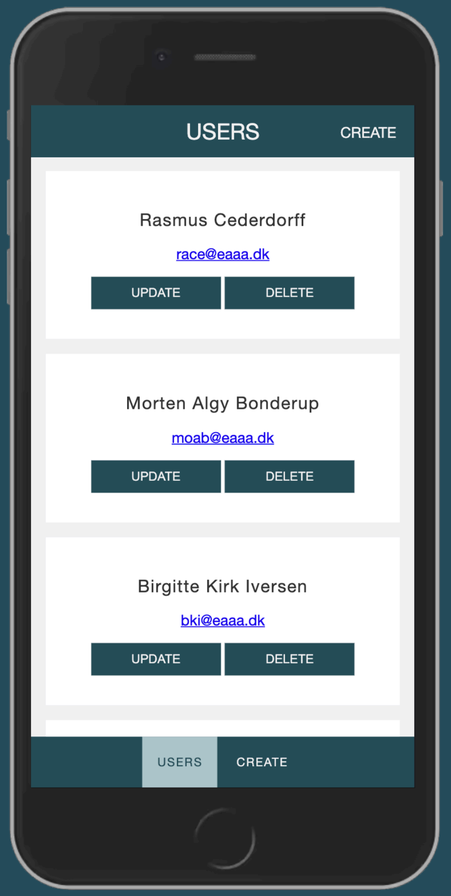
EXCHANGING DATA OVER THE WEB

FETCH, HTTP REQUEST & RESPONSE

WEB APP

API

DATA SOURCE

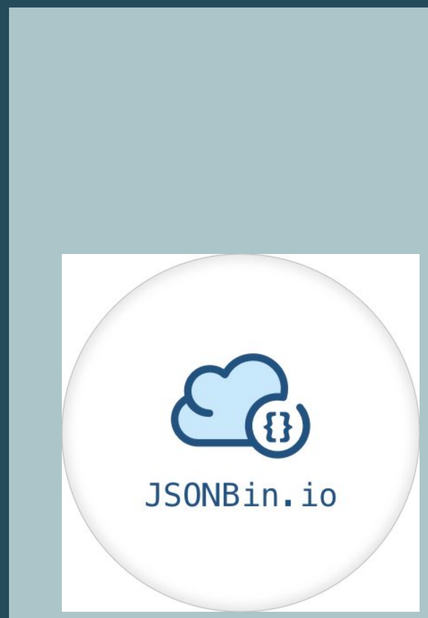


FETCH(...)

REQUEST

JSON DATA

RESPONSE

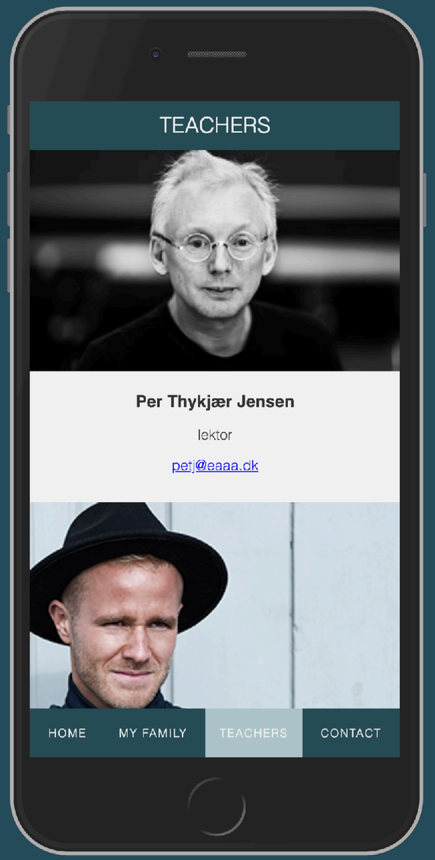


THE GLUE: JSON & API

FRONTEND

API

BACKEND



FETCH(...)

REQUEST

JSON DATA

RESPONSE

JSON

… a syntax for storing and exchanging data over the web

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

JSON OBJECT

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

LIST OF JSON OBJECTS

JAVASCRIPT OBJECT NOTATION

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


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

JSON METHODS

```
const user = {
  name: "John",
  age: 30,
  gender: "male",
  lookingFor: "female"
};

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

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

json-parse-stringify

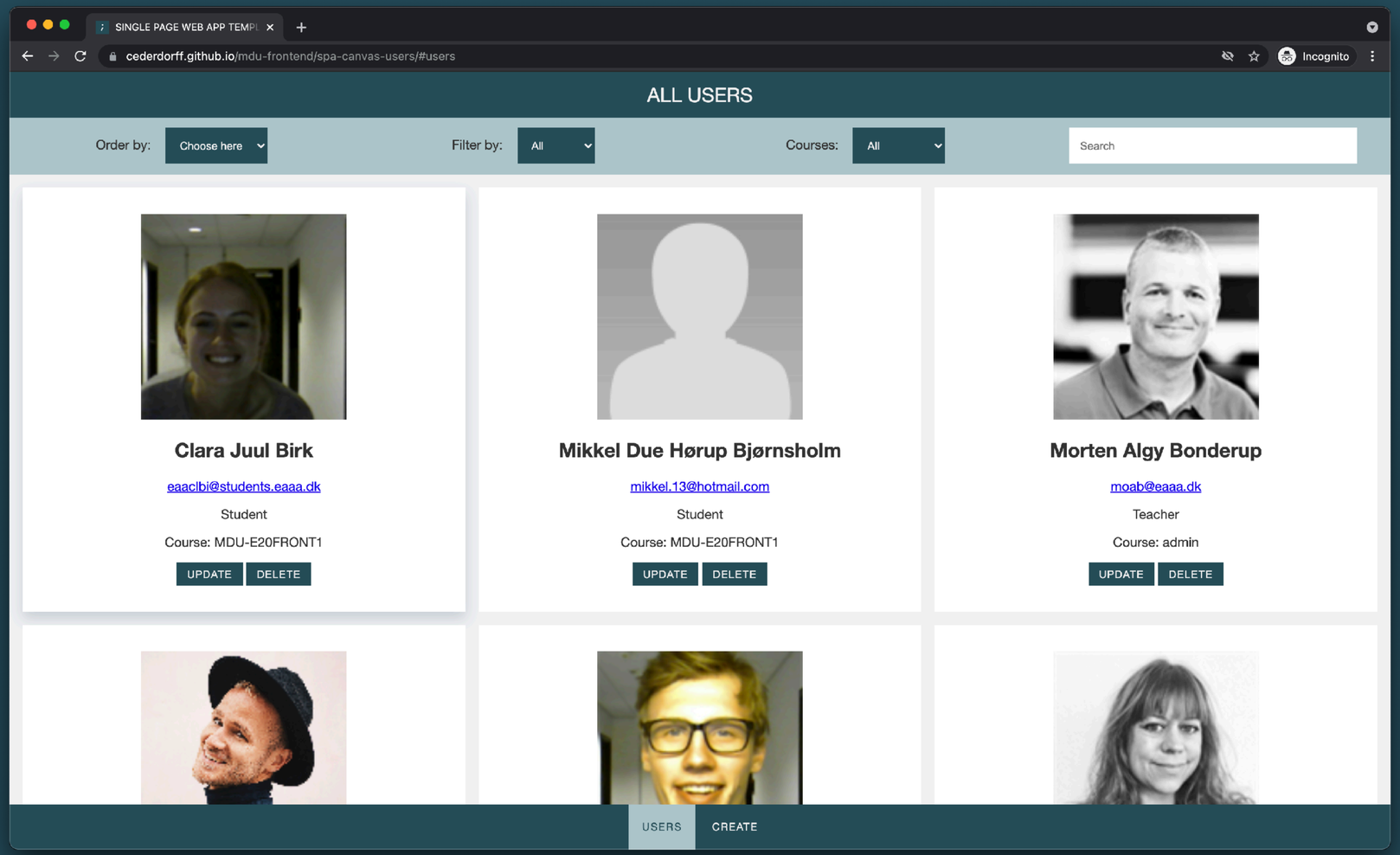
FETCH

... 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



spa-canvas-users

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

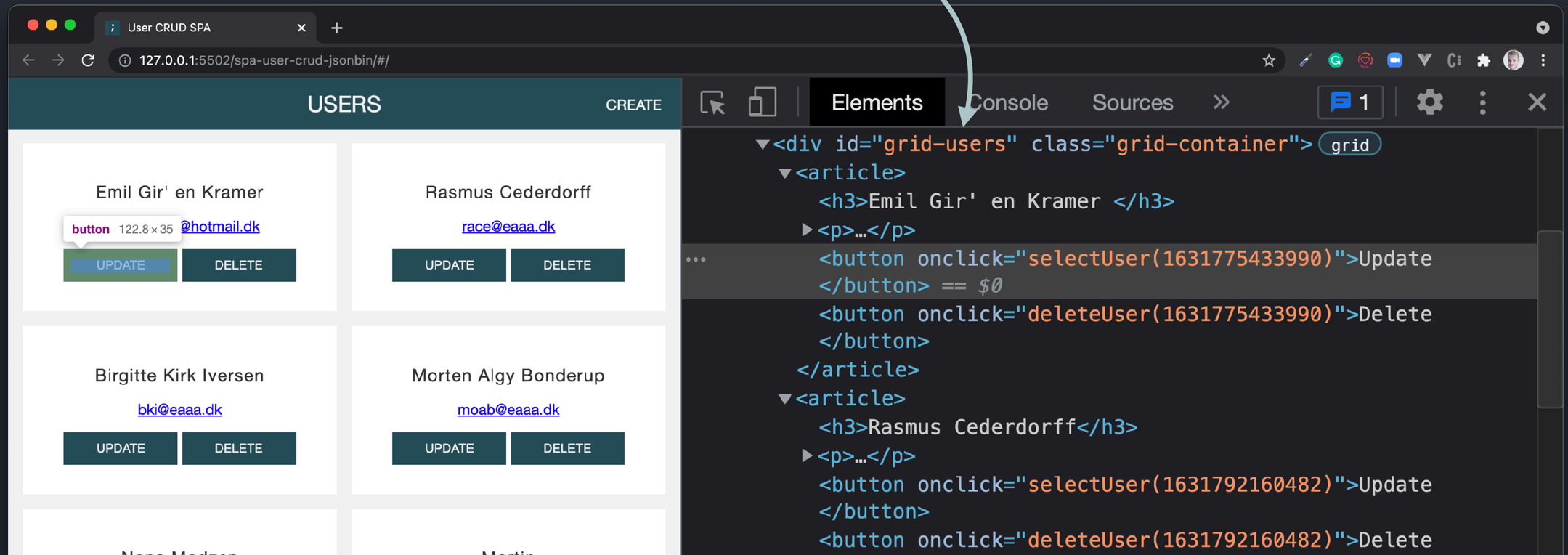
calling `selectUser(...)` with `id` of user

```
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 indout values with selected user values
  nameInput.value = user.name;
  mailInput.value = user.mail;
  navigateTo("#/update");
}
```

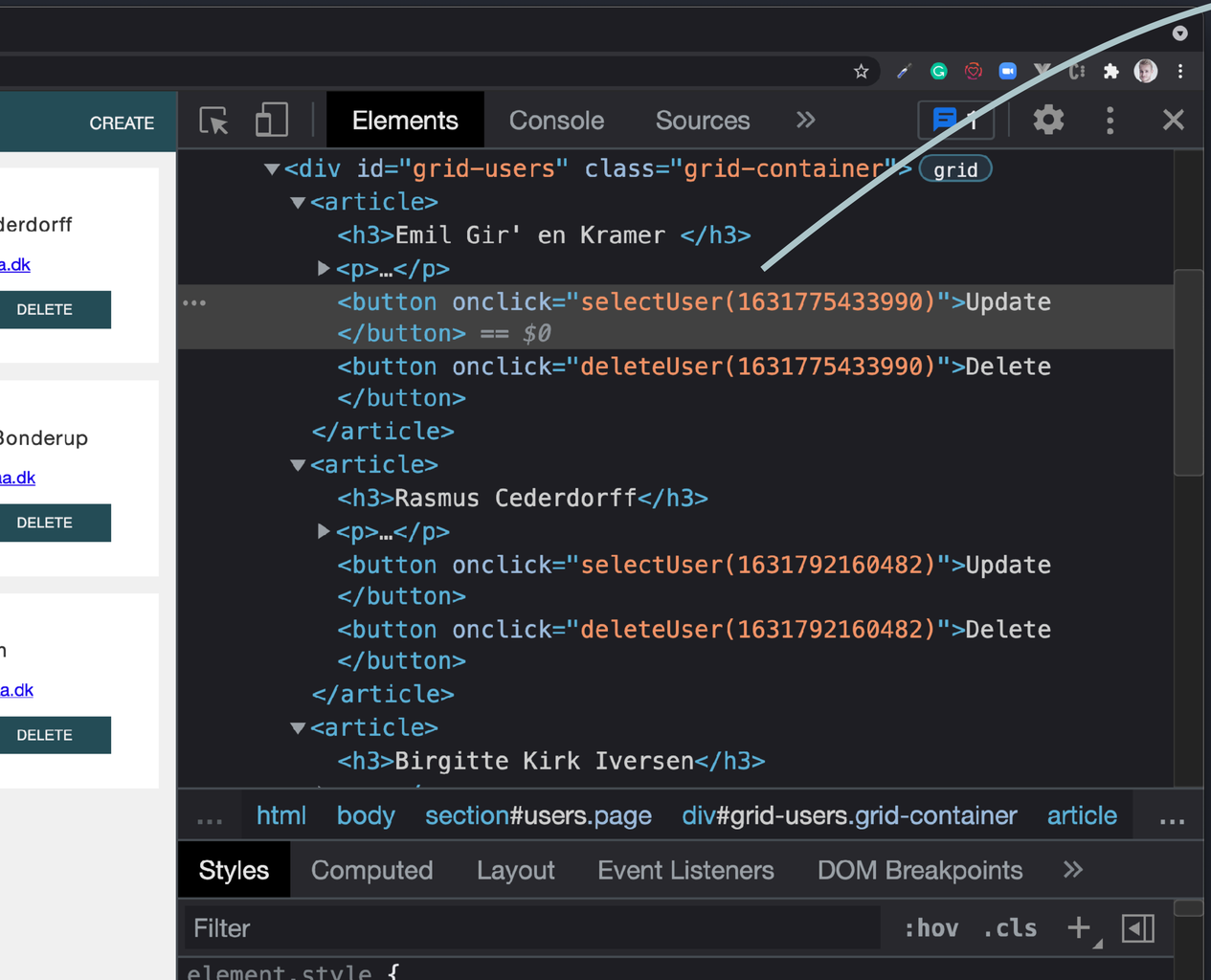
```
function appendUsers(users) {
  let htmlTemplate = "";
  for (const user of users) {
    htmlTemplate += /*html*/ `
      <article>
        <h3>${user.name}</h3>
        <p><a href="mailto:${user.mail}">${user.mail}</a></p>
        <button onclick="selectUser(${user.id})">Update</button>
       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

append to #grid-users



ID'S & FUNCTIONS: UPDATE

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




```
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 indout values with selected user values
  nameInput.value = user.name;
  mailInput.value = user.mail;
  navigateTo("#/update");
}
```

ID'S & FUNCTIONS: UPDATE

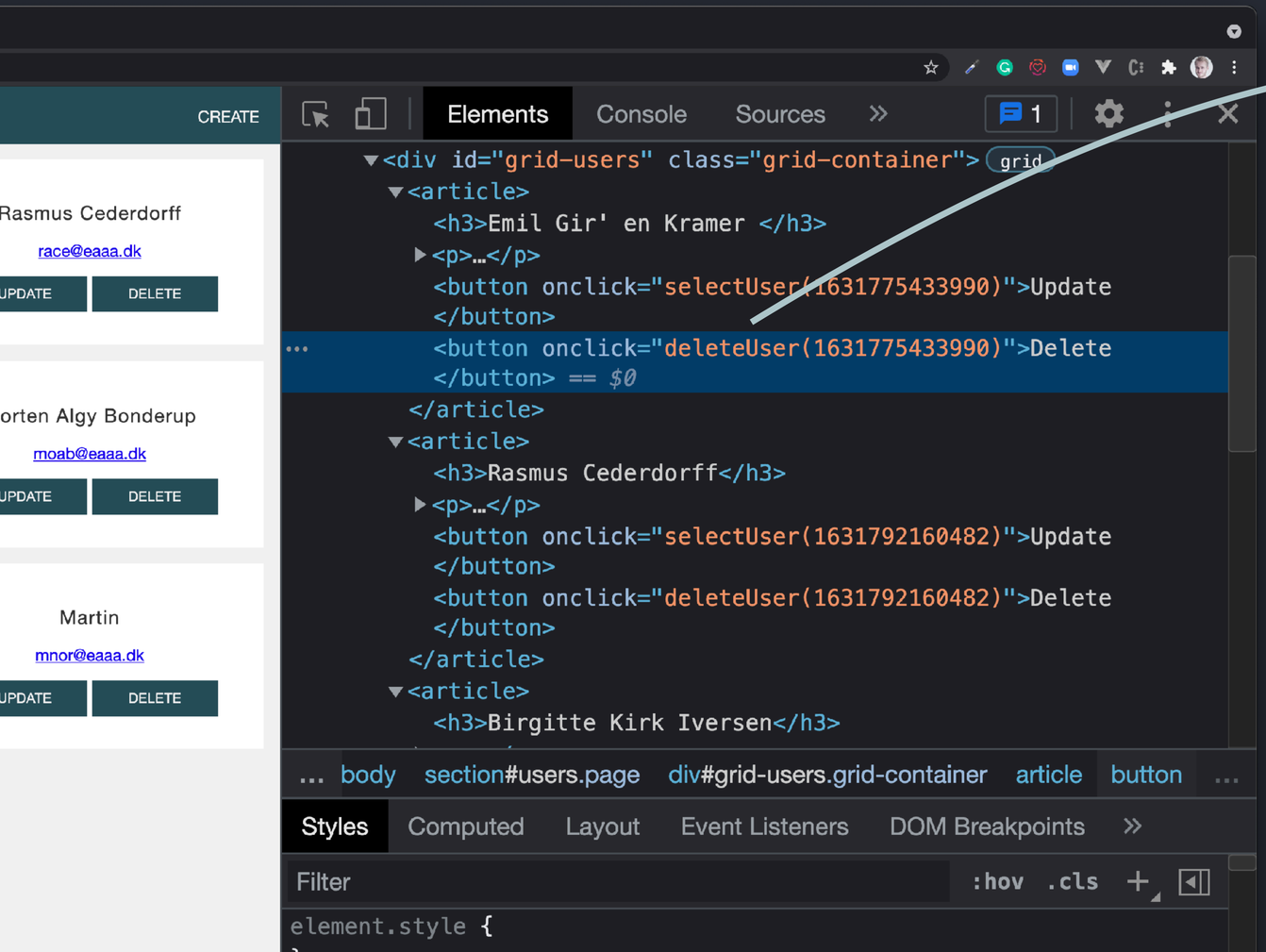
```html
<section id="update" class="page">
  <header class="topbar">
    <a href="#/" class="nav-link left">Back</a>
    <h2>Update user</h2>
    <a class="right" onclick="updateUser()">Update</a>
  </header>
  <form>
    <input type="text" id="name-update" placeholder="Type your name" required>
   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

```js
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




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

BACK

ASYNC JS

JavaScript reads and runs the script from top to bottom.

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

... by default JavaScript is synchronous.

JS IS SYNCHRONOUS & SINGLE-THREADED

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

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

mySecond();
myFirst();
```

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

/* ------------------------------------- */

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

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

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

initApp();
```

WITH CALLBACKS WE CAN MAKE JS ASYNCHRONOUS

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

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

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

FETCH IS ASYNCHRONOUS

HTTP REQUEST IN JAVASCRIPT

...A WAY TO GET & POST DATA FROM & TO A DATA SOURCE

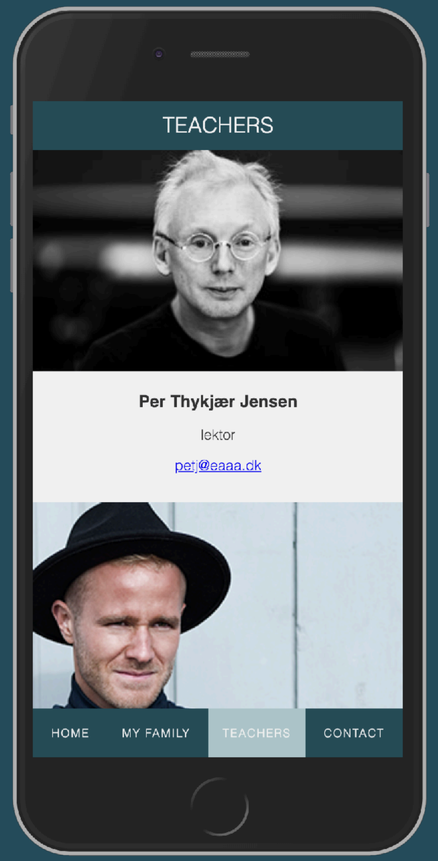
```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json")
    .then(function (response) {
        return response.json();
    })
    .then(function (data) {
        console.log(data);
    });
//
//
// or with async/await
const response = await fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json");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

WEB DEVELOPMENT

FRONTEND

(API)

BACKEND



FETCH(...)

REQUEST

JSON DATA

RESPONSE

"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

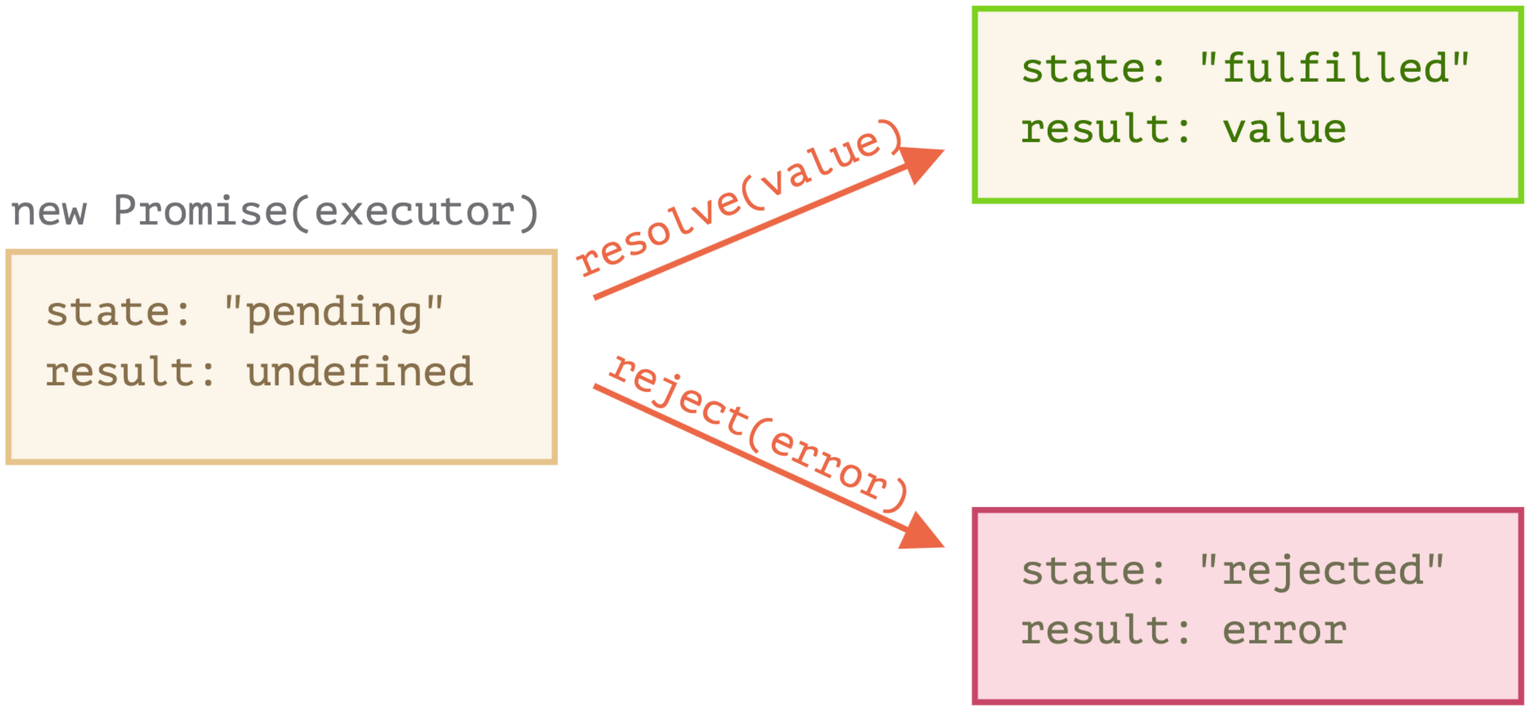
ASYNC MAKES A FUNCTION RETURN A PROMISE

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



https://javascript.info/promise-basics

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

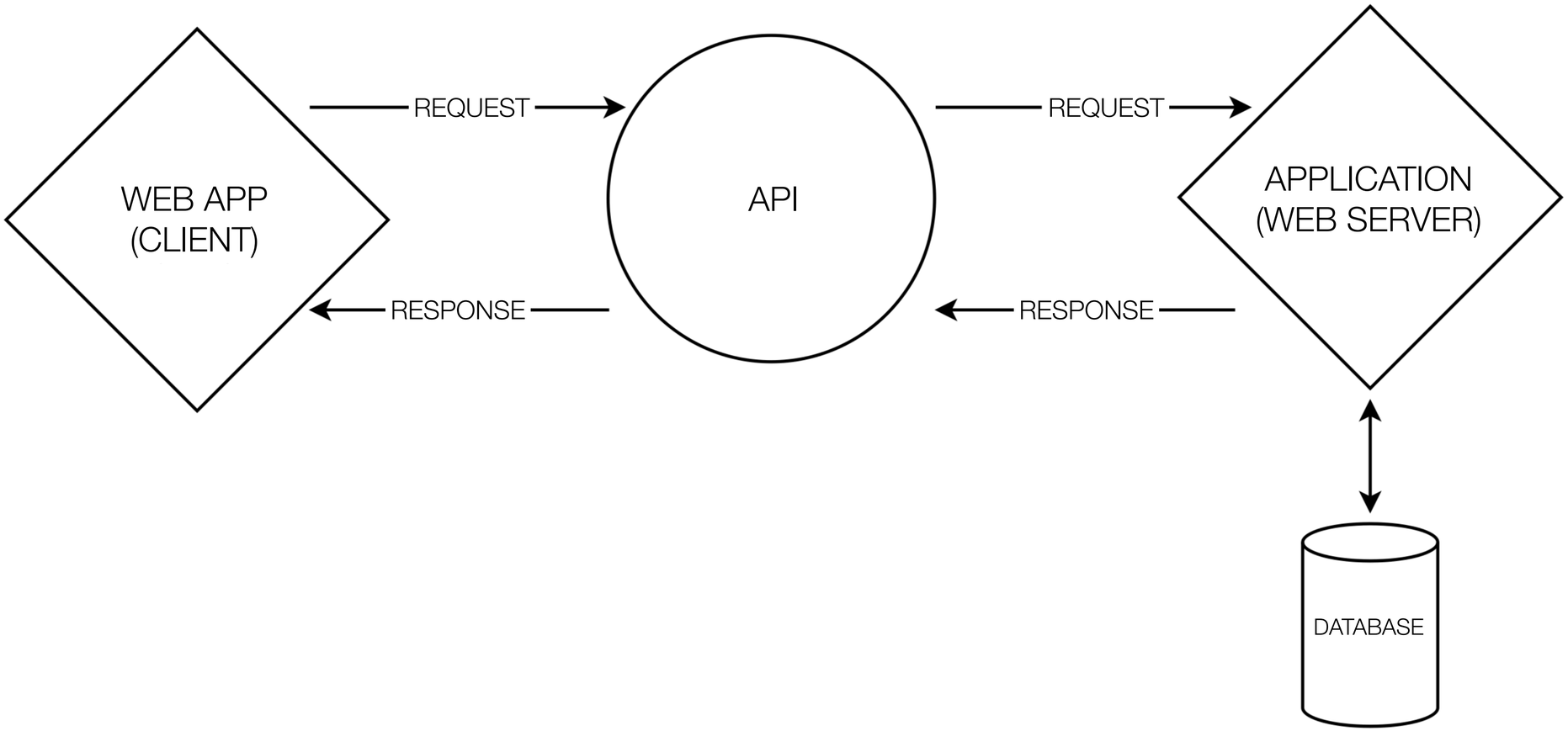
... AND HTTP METHOD "POST"

```
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

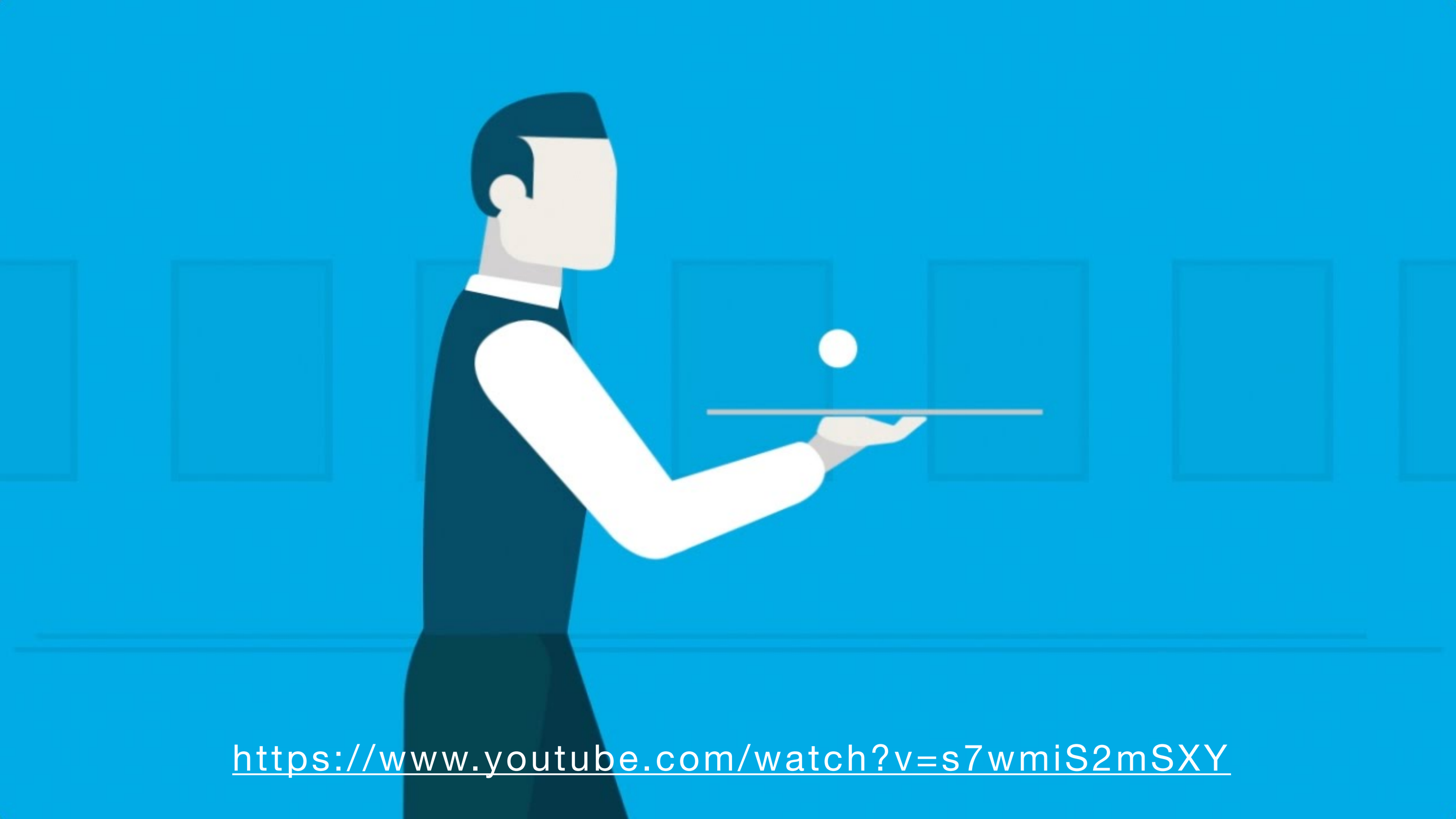


APPLICATION PROGRAMMING INTERFACE

CLIENT

API

SERVER



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

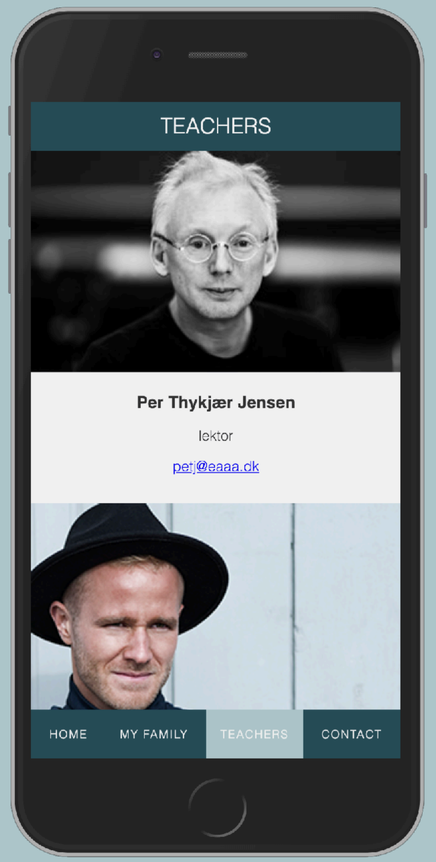
CLIENT
(WEB APP)
API
DATA SOURCE
RECEIVER
WAITER
(MESSENGER)
KITCHEN
(SOURCE)

APPLICATION PROGRAMMING INTERFACE

WEB APP

API

DATA SOURCE



FETCH(...)

REQUEST

JSON DATA

RESPONSE

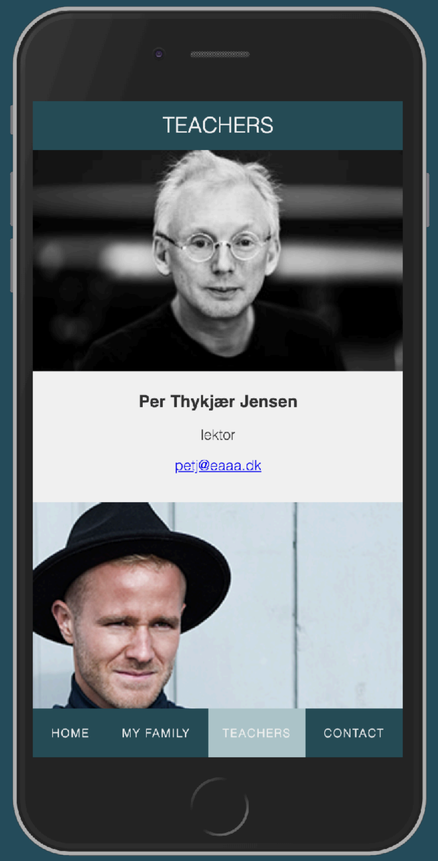
JSON

JSON

{ api }

APPLICATION PROGRAMMING INTERFACE

FRONTEND



API

FETCH(...)

REQUEST

JSON DATA

RESPONSE

JSON

BACKEND

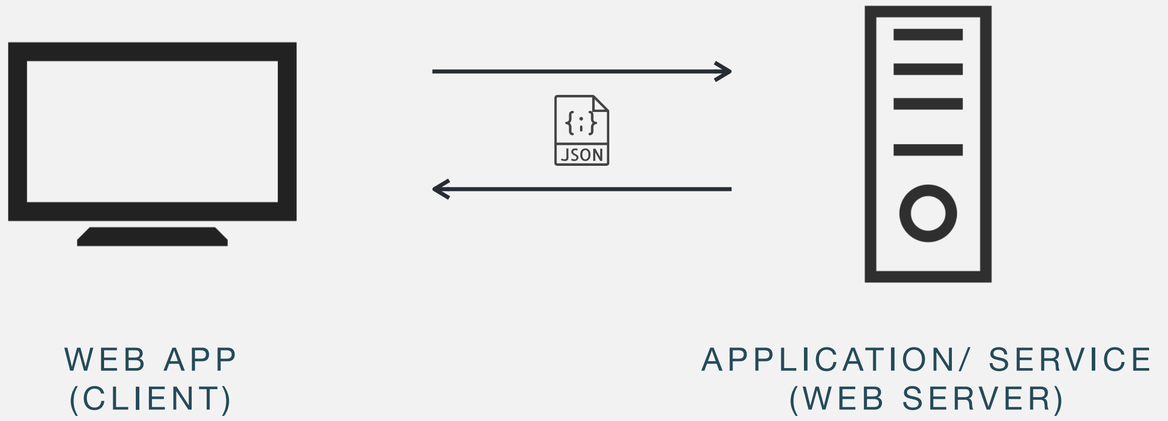
APPLICATION PROGRAMMING INTERFACE

- 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.

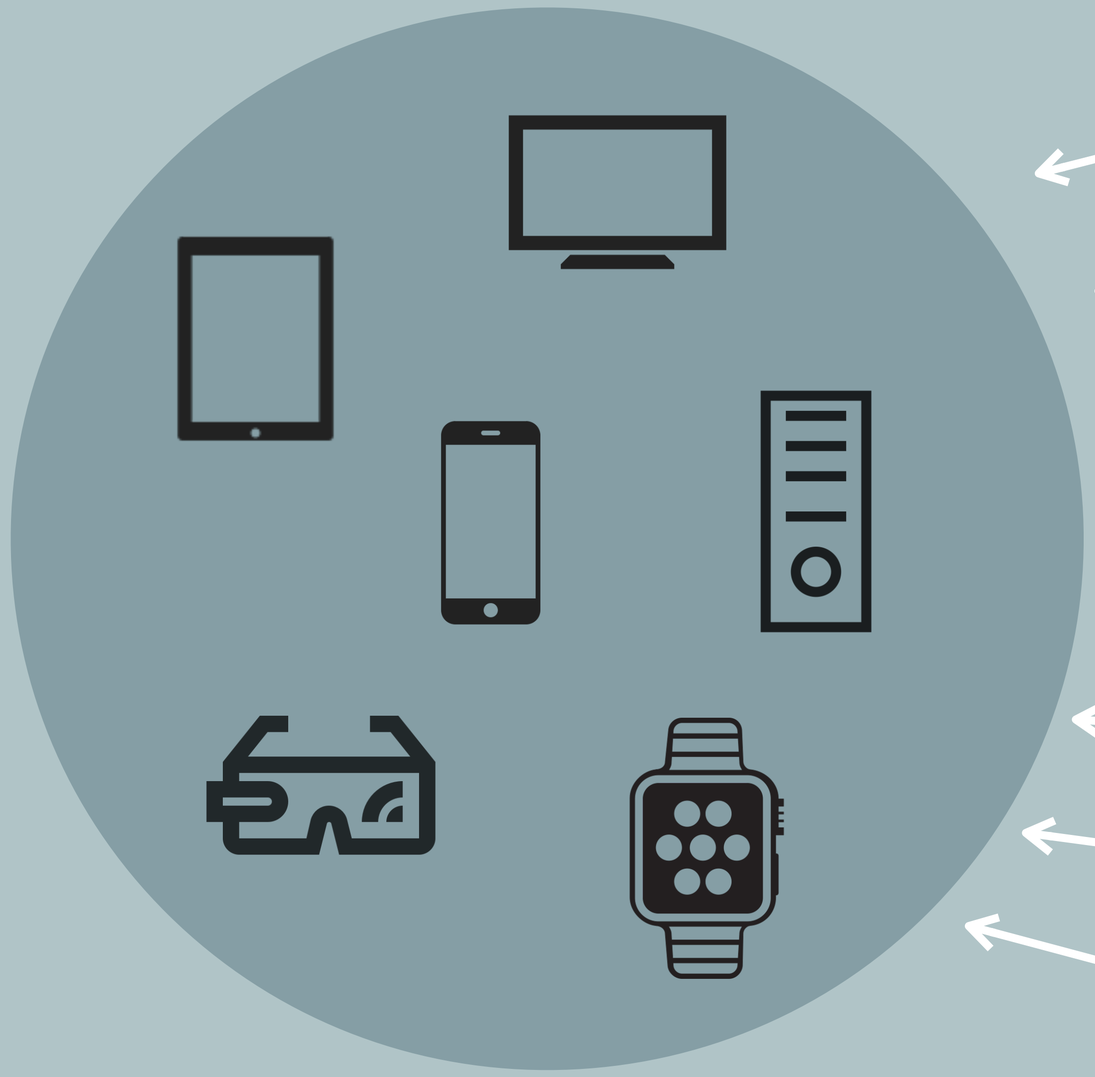
- Platform independent: Can be used by different clients, devices and users: websites, web apps, mobile apps, webshops and other clients.

- The client does not need to know anything about the service or program provided by the API and visa versa.



API

WEB APP



DATA SOURCES

API — WIKIPEDIA

API — GOOGLE MAPS

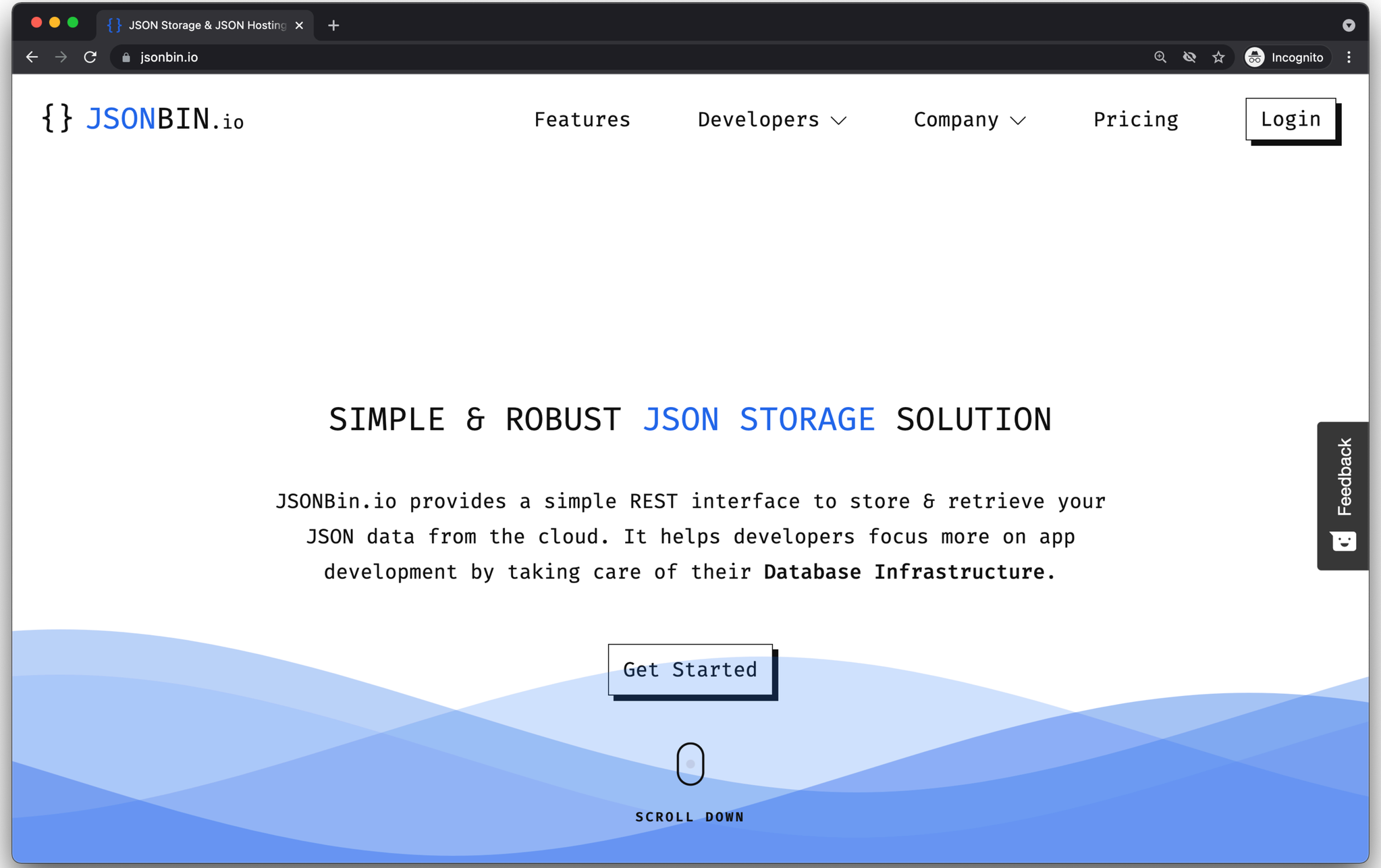
API — CVR API

API — SoMe

API — WEATHER API

API — HEADLESS CMS

API — PHP BACKEND


JSON Storage & JSON Hosting
jsonbin.io
Incognito
{} 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

{} JSONBIN.io

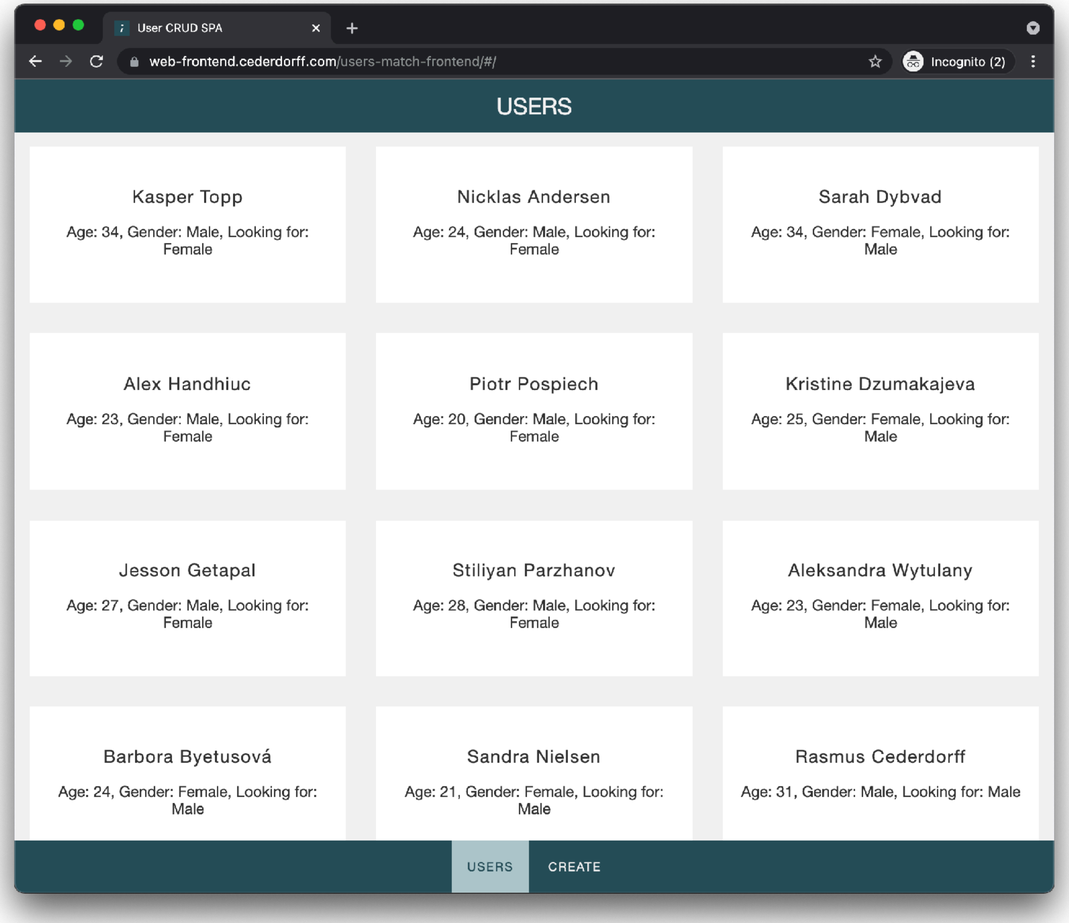
The JSONBin.io API is organized around REST. Our API has predictable resource-oriented URLs, returns JSON-encoded responses, and uses standard HTTP response codes and verbs.

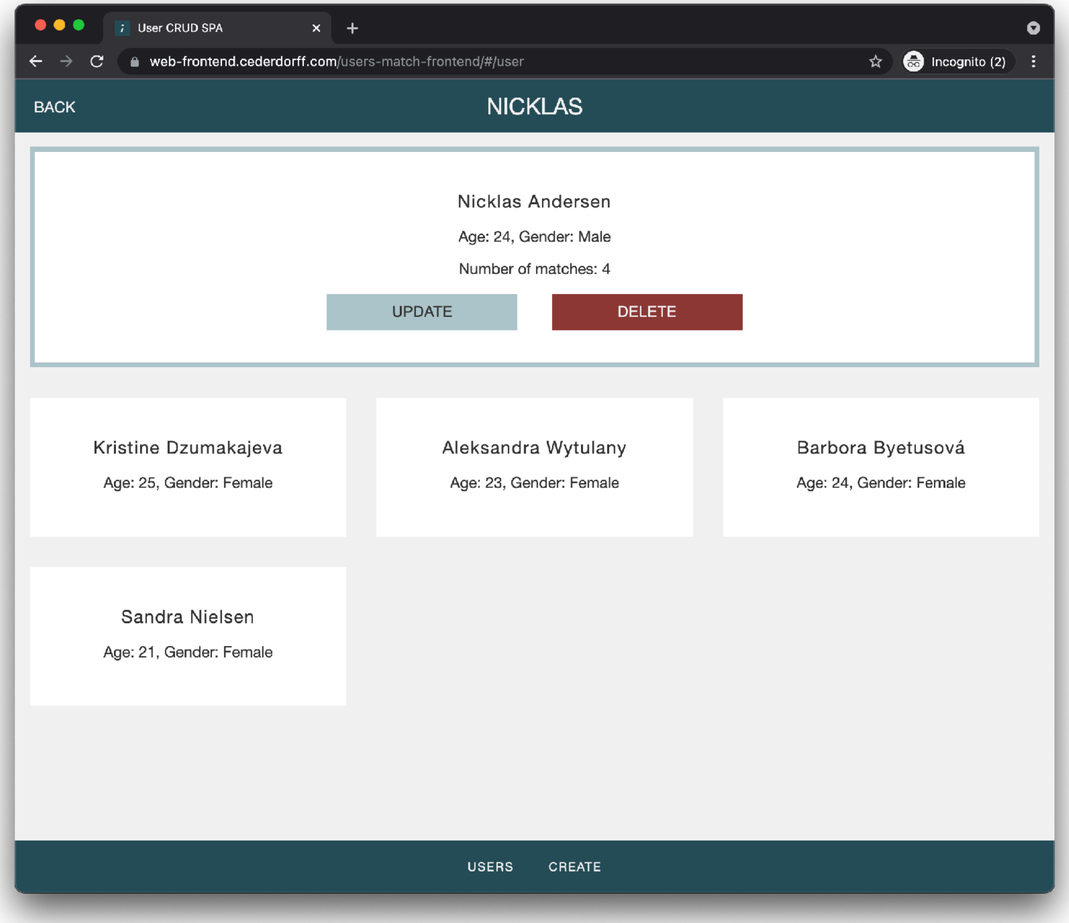
BINS API
Bins are nothing but JSON records. Bins API is the one you will be using the most to give you complete control of Creating, Updating, Reading & Deleting your JSON Data.

https://jsonbin.io/api-reference

CASE: DATING SPA

BACK



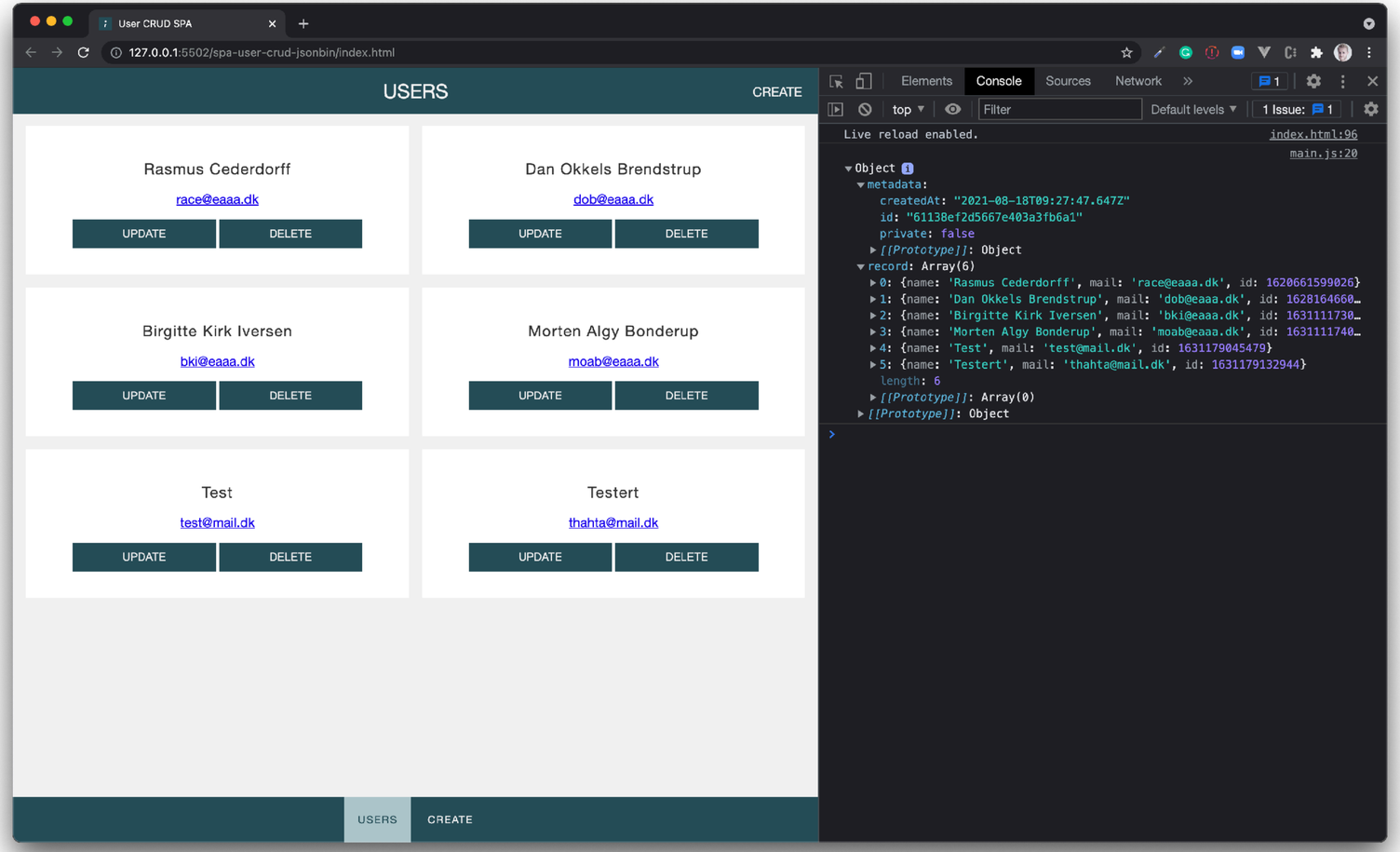


Frontend: `users-match-frontend`

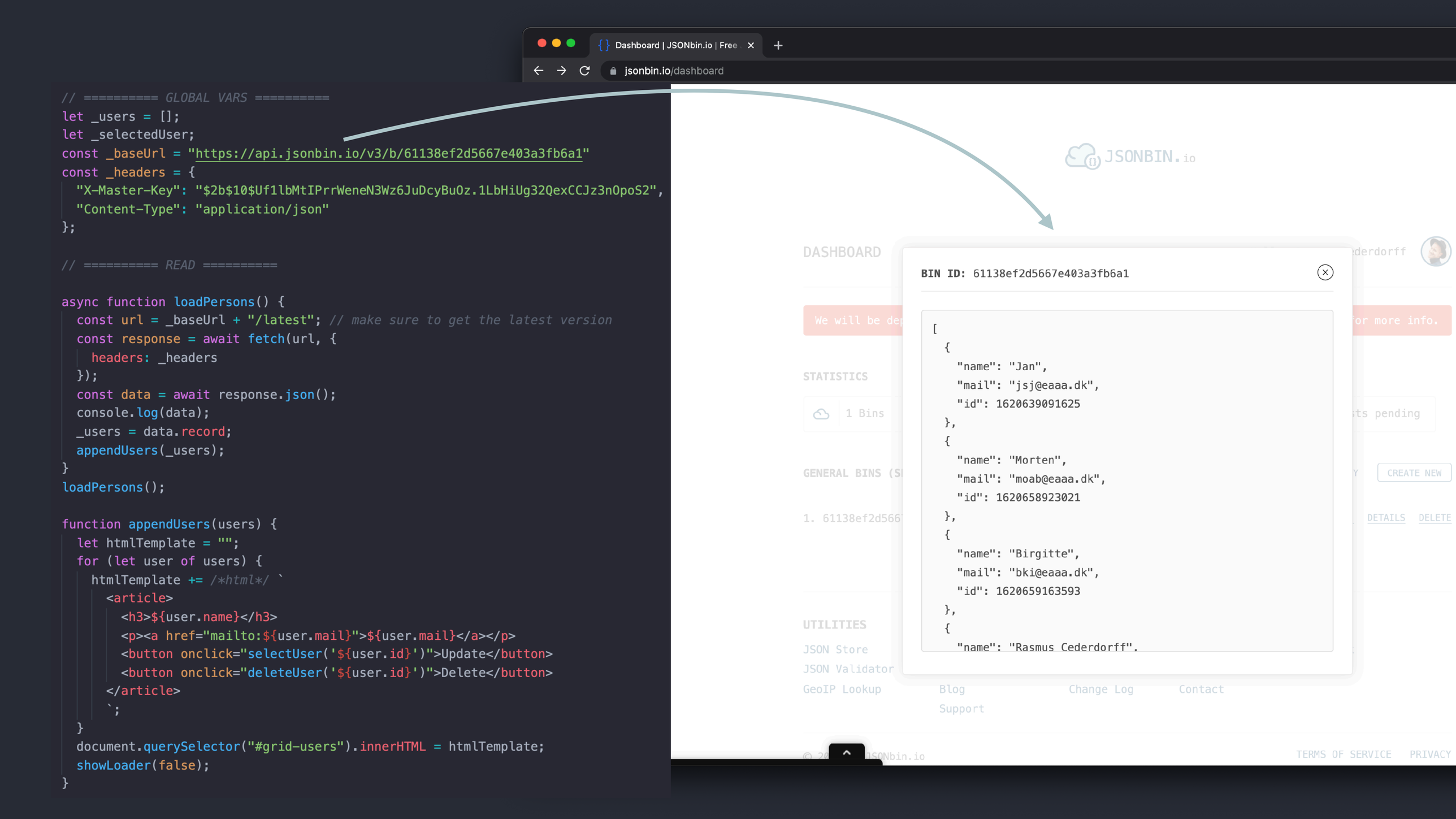
PHP Backend Service: `users-match-backend`

CASE: USER CRUD

CREATE, READ, UPDATE & DELETE



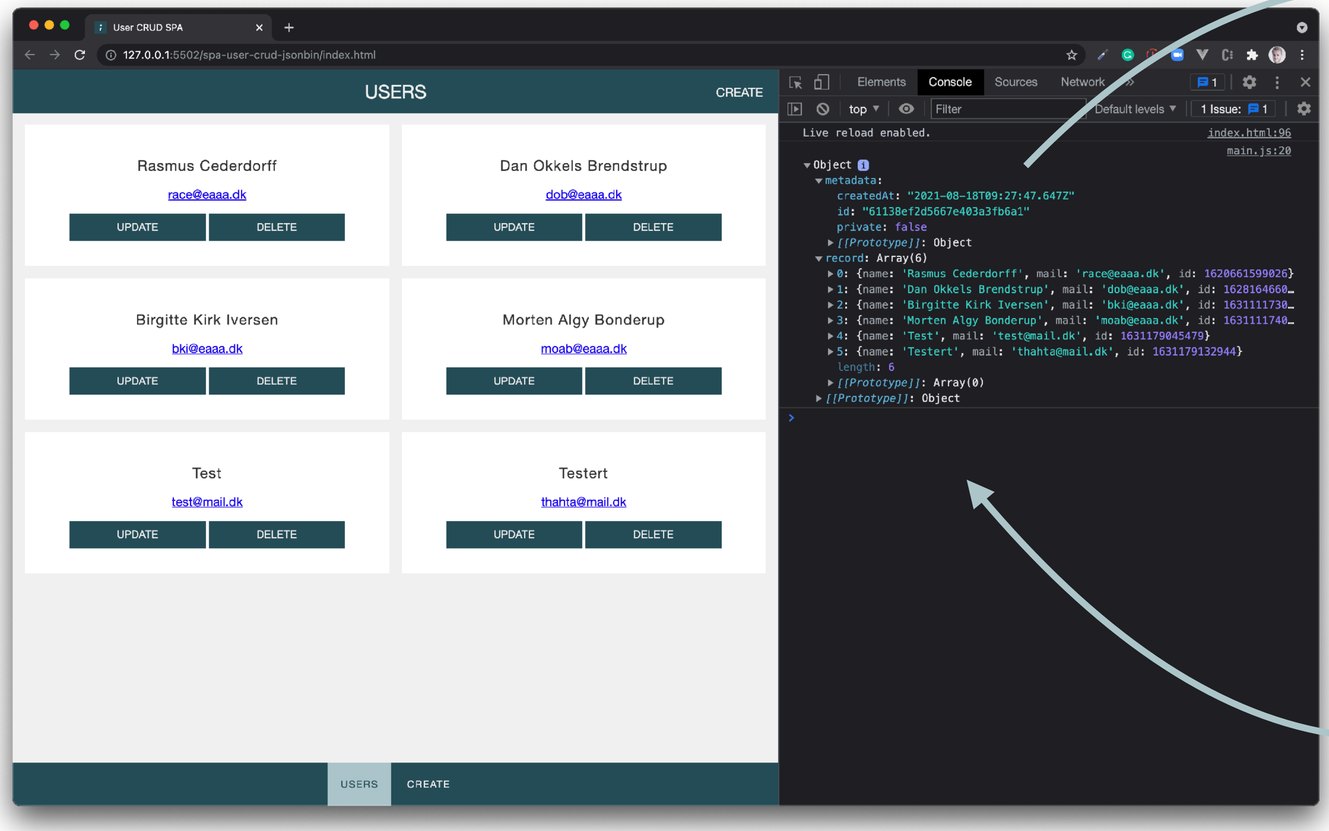
spa-user-crud-jsonbin


// ========== GLOBAL VARS ==========
let _users = [];
let _selectedUser;
const _baseUrl = "https://api.jsonbin.io/v3/b/61138ef2d5667e403a3fb6a1"
const _headers = {
"X-Master-Key": "$2b$10$Uf1lbMtIPrrWeneN3Wz6JuDcyBuOz.1LbHiUg32QexCCJz3nOpoS2",
"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*/ `
<article>
<h3>${user.name}</h3>
${user.mail}
<button onclick="selectUser('${user.id}')">Update</button>
<button onclick="deleteUser('${user.id}')">Delete</button>
</article>
`;
}
document.querySelector("#grid-users").innerHTML = htmlTemplate;
showLoader(false);
}
Dashboard | JSONbin.io | Free
jsonbin.io/dashboard
JSONBIN.io
DASHBOARD
BIN ID: 61138ef2d5667e403a3fb6a1
[
{
"name": "Jan",
"mail": "jsj@eaaa.dk",
"id": 1620639091625
},
{
"name": "Morten",
"mail": "moab@eaaa.dk",
"id": 1620658923021
},
{
"name": "Birgitte",
"mail": "bki@eaaa.dk",
"id": 1620659163593
},
{
"name": "Rasmus Cederdorff",
STATISTICS
1 Bins
GENERAL BINS
CREATE NEW
DETAILS
DELETE
UTILITIES
JSON Store
JSON Validator
GeoIP Lookup
Blog
Support
Change Log
Contact
TERMS OF SERVICE
PRIVACY

CASE: USER CRUD

CREATE, READ, UPDATE & DELETE

REQUEST



RESPONSE

JSONBIN.io

DASHBOARD

We will be dep

STATISTICS

1 Bins

GENERAL BINS (SI

1. 61138ef2d566

UTILITIES

JSON Store
JSON Validator
GeoIP Lookup

Blog
Support
Change Log
Contact

BIN ID: 61138ef2d5667e403a3fb6a1

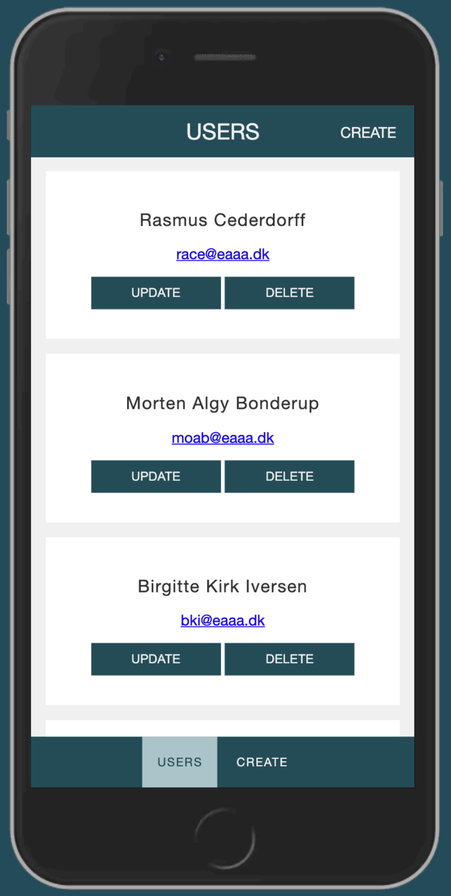

```
[
  {
    "name": "Jan",
    "mail": "jsj@eaaa.dk",
    "id": 1620639091625
  },
  {
    "name": "Morten",
    "mail": "moab@eaaa.dk",
    "id": 1620658923021
  },
  {
    "name": "Birgitte",
    "mail": "bki@eaaa.dk",
    "id": 1620659163593
  },
  {
    "name": "Rasmus Cederdorff",
```

FETCH, HTTP REQUEST & RESPONSE

WEB APP

API

DATA SOURCE



FETCH(... GET OR PUT ...)

REQUEST

JSON DATA

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

C R U D

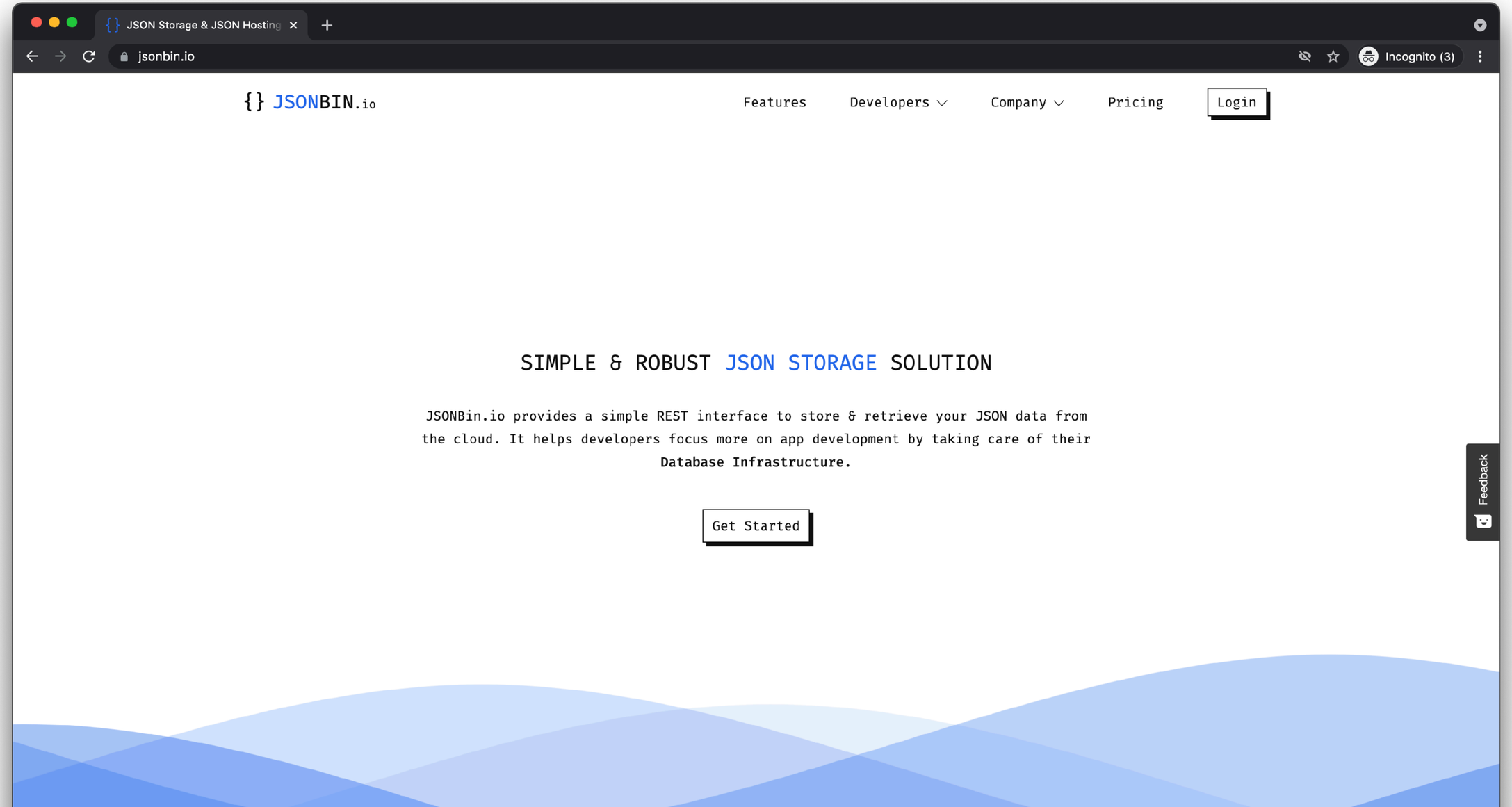
CREATE - a new object, a person, user, movie, posts, product etc.

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

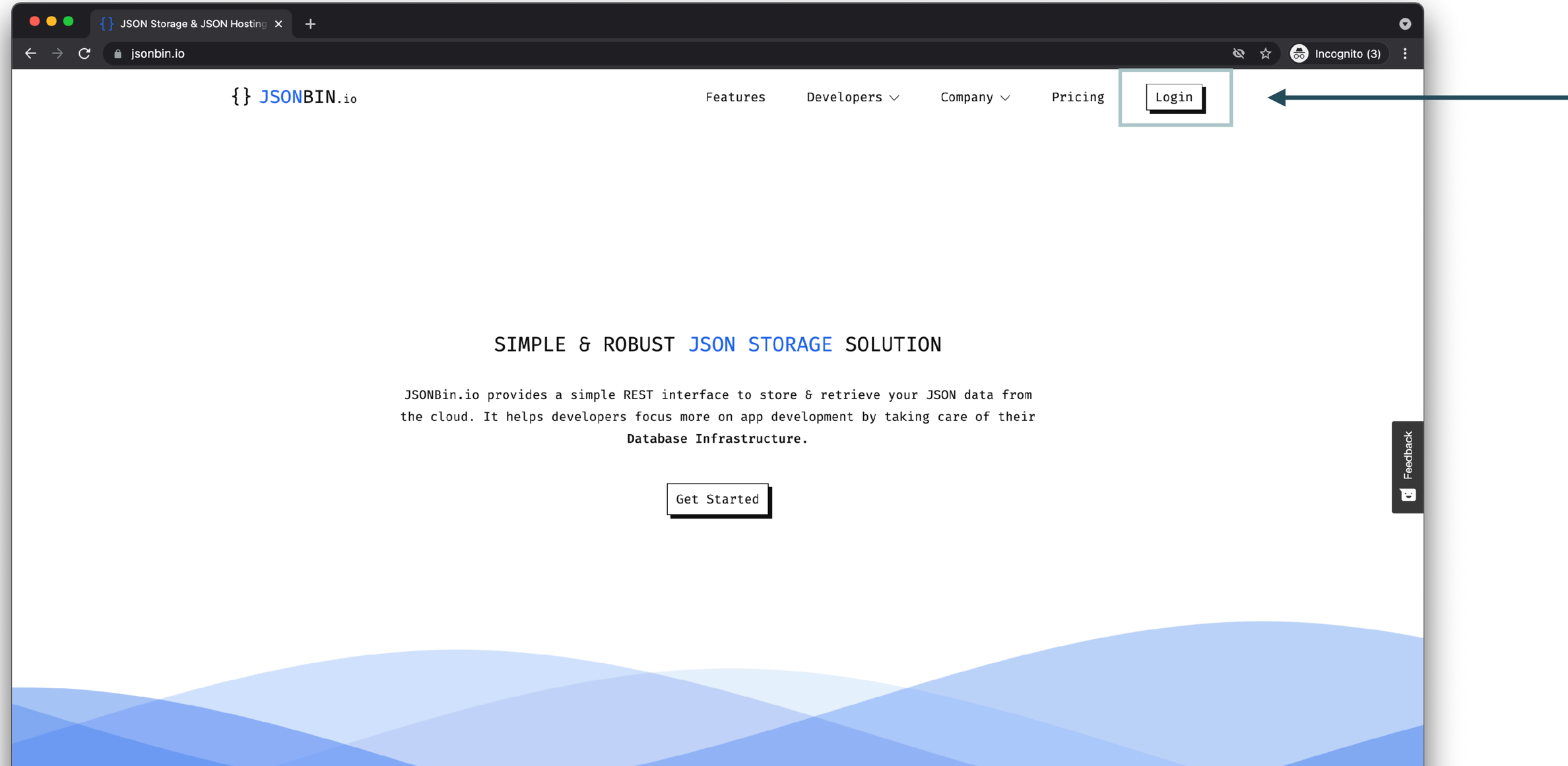
UPDATE - update an object, a person, user, movie, posts, product etc.

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

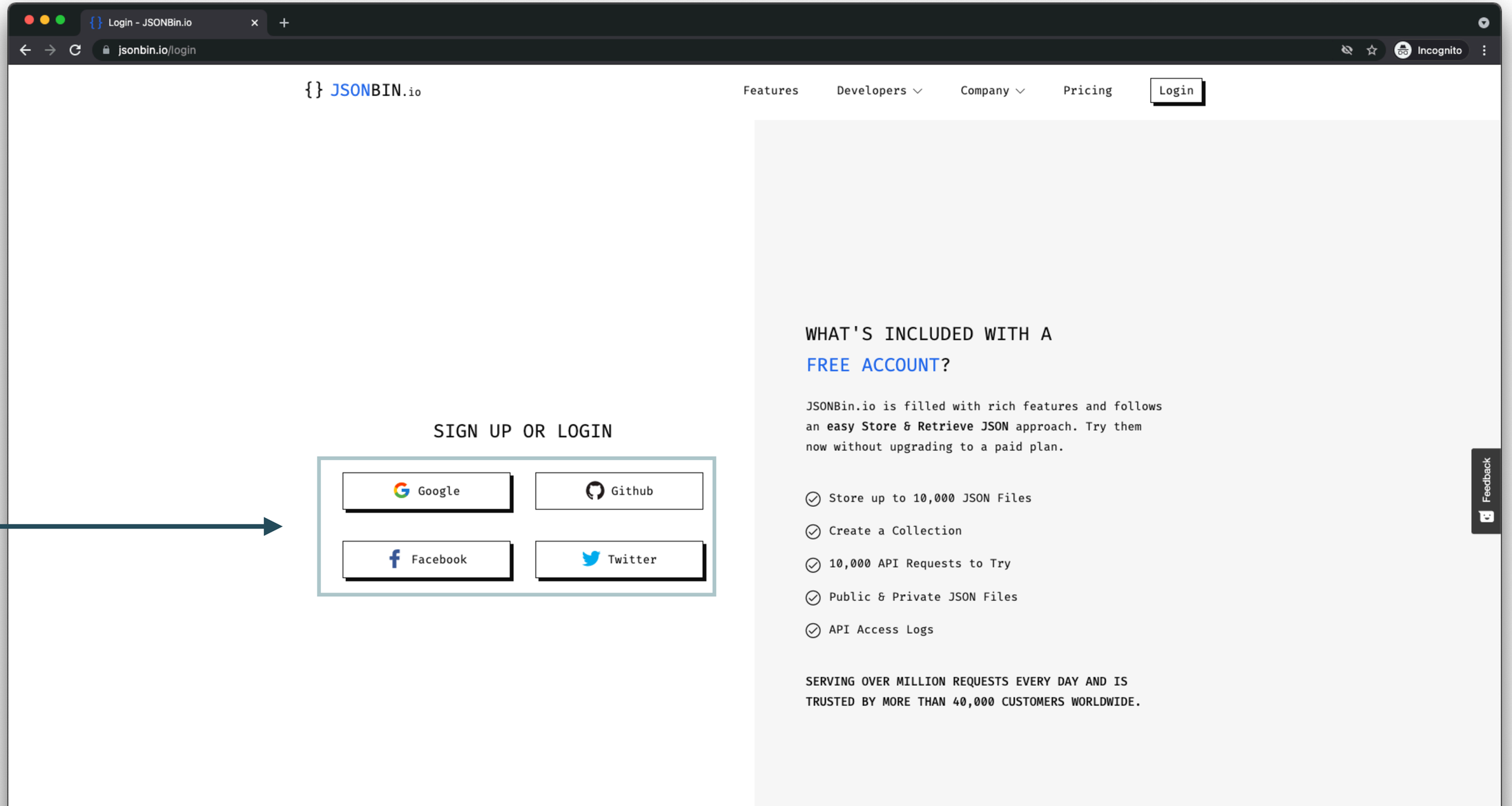
JSONBIN.IO


JSON Storage & JSON Hosting
jsonbin.io
Incognito (3)
{} 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

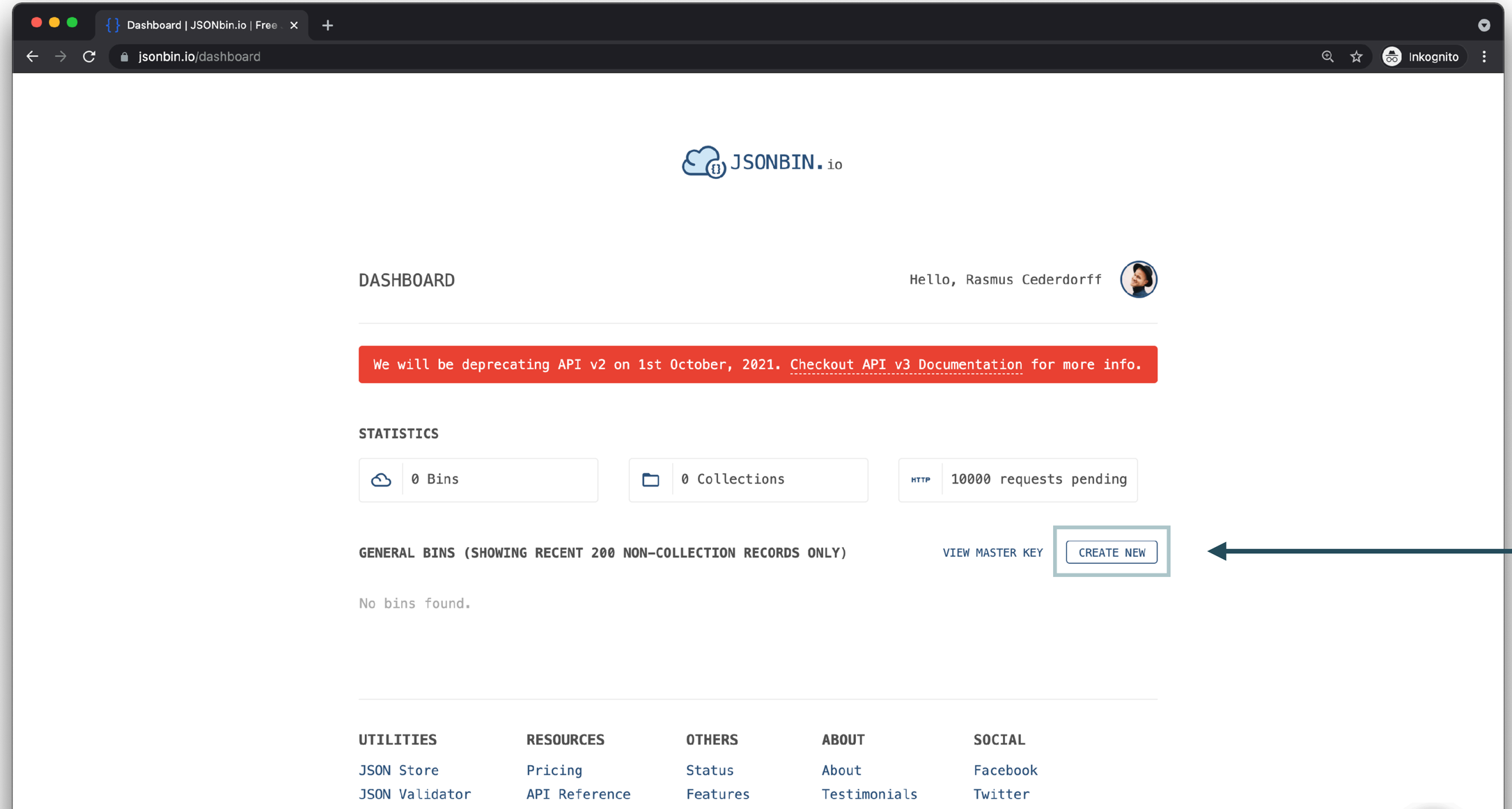
LOGIN / SIGN UP



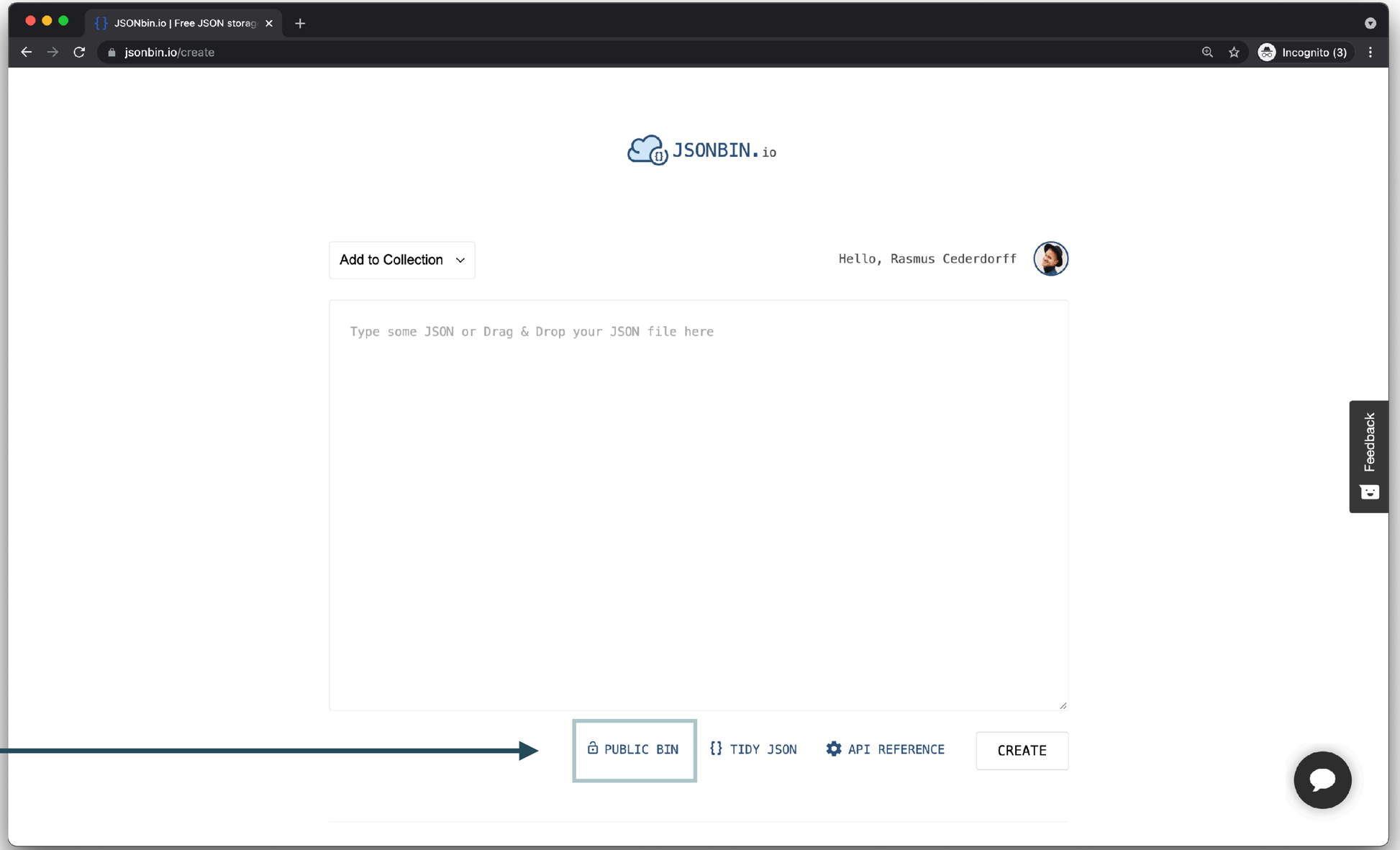
LOGIN / SIGN UP



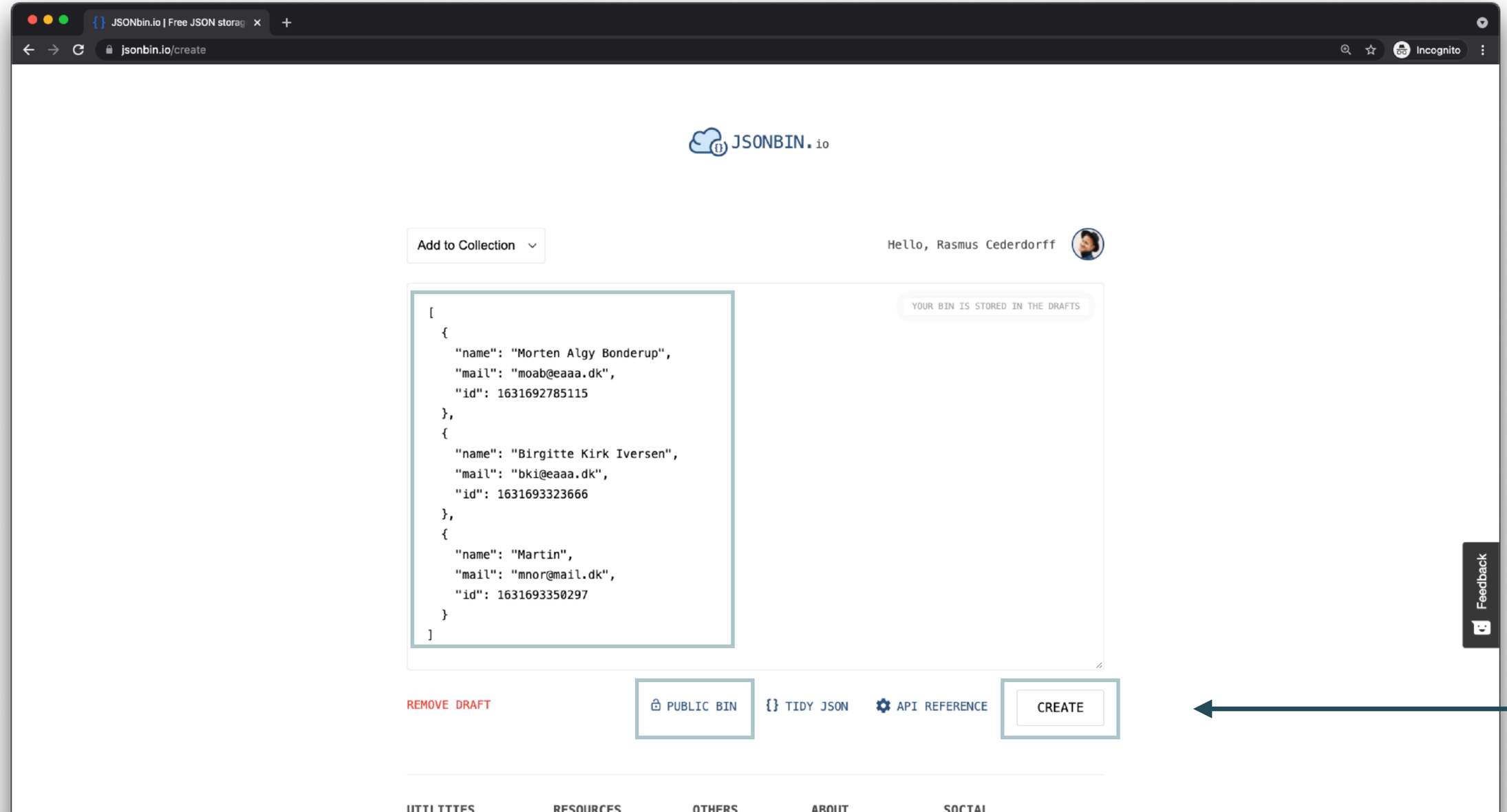
DASHBOARD —> CREATE NEW BIN



CREATE **PUBLIC** BIN



PASTE SOME JSON & SAVE



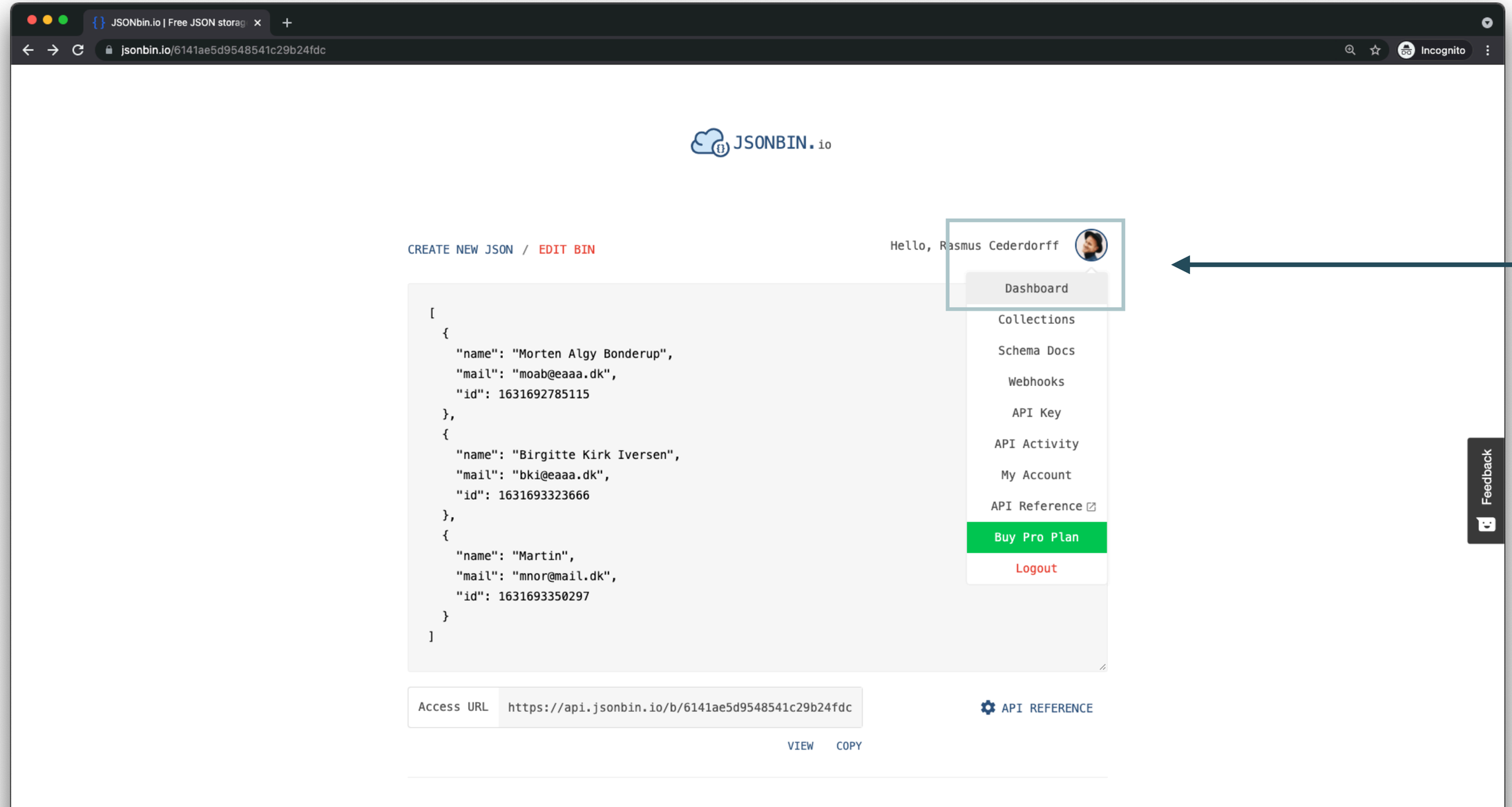
PASTE SOME INITIAL JSON

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

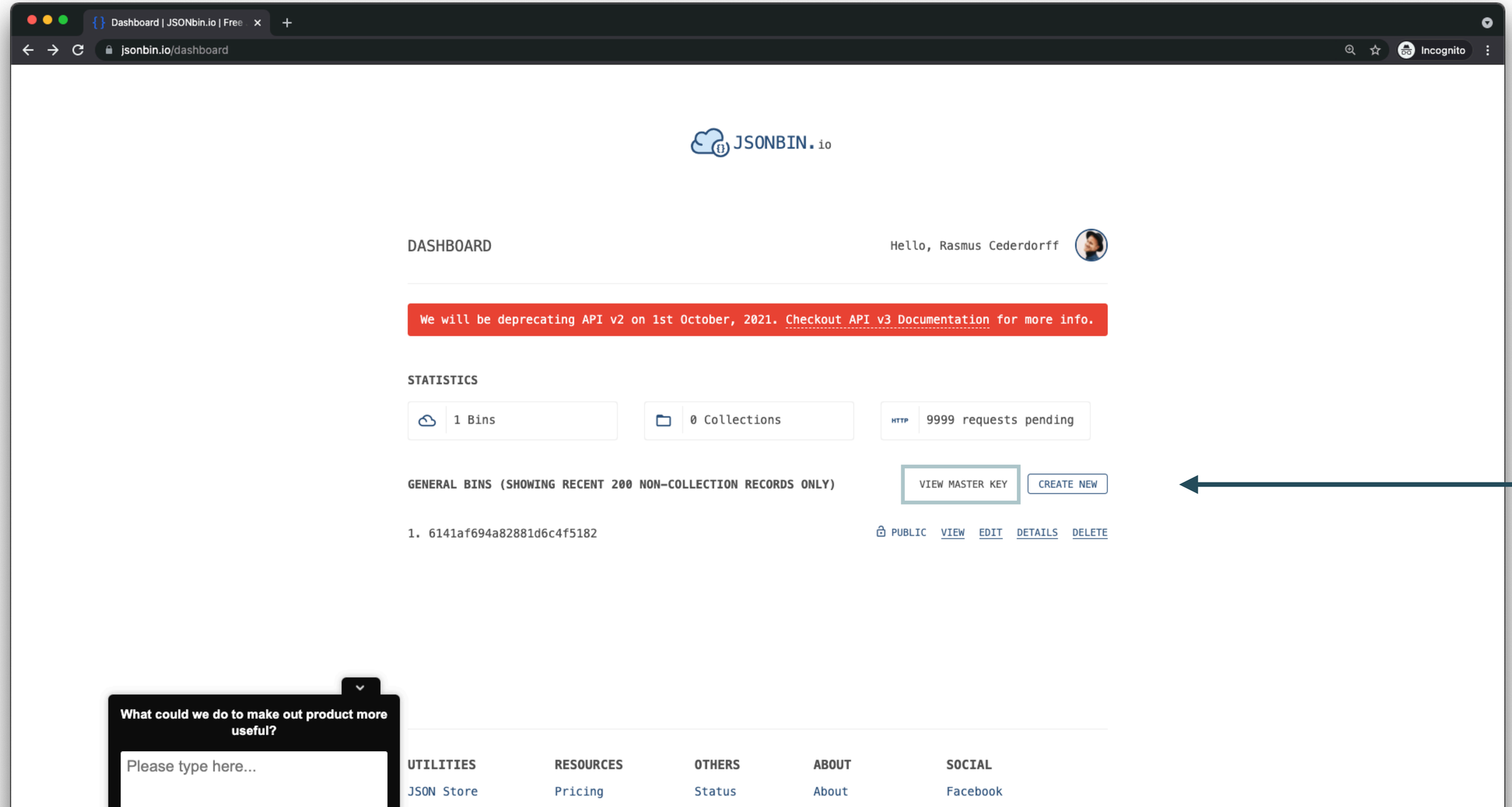
Array

Objects

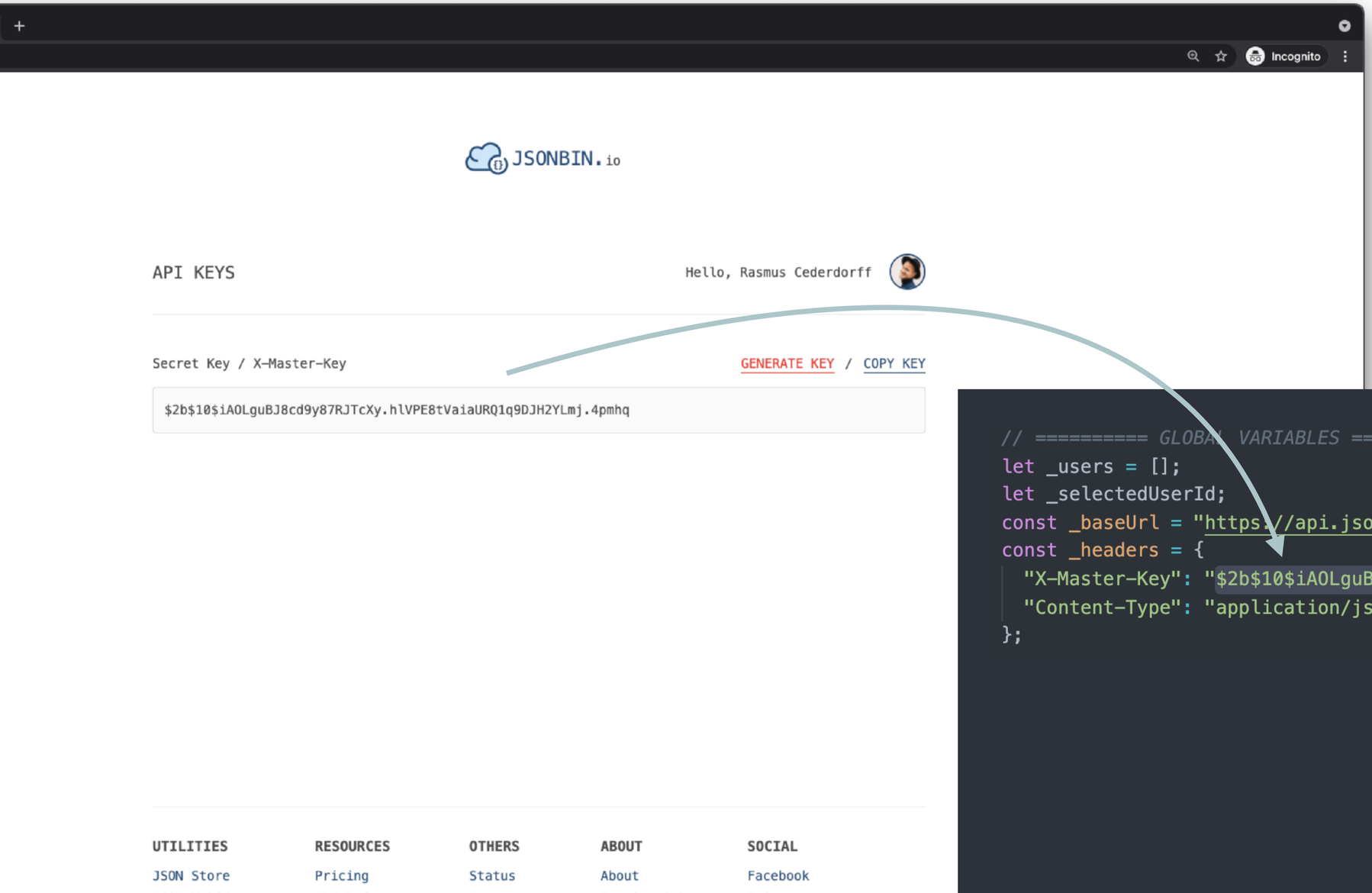
GO BACK TO DASHBOARD



VIEW MASTER KEY

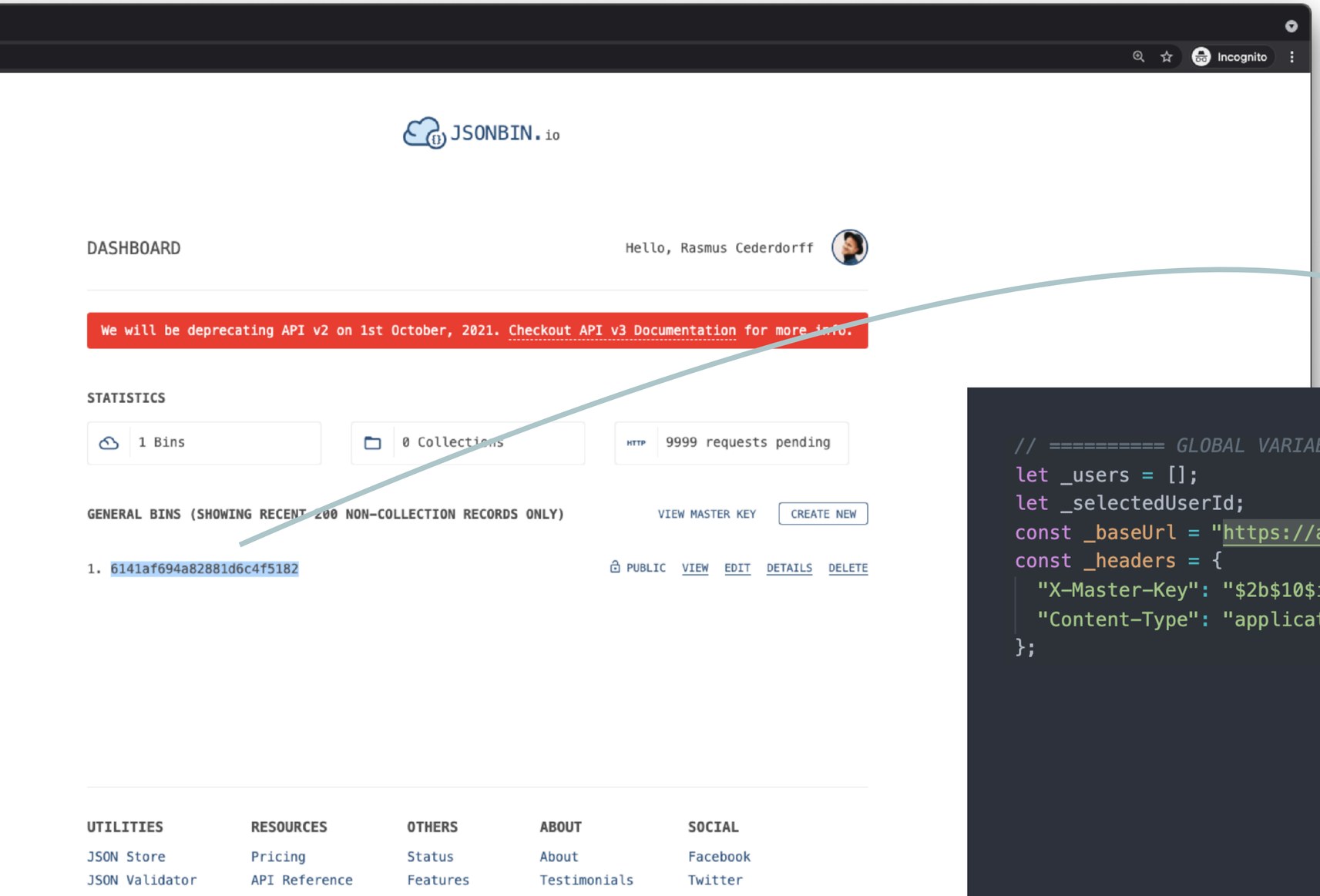


COPY AND PASTE KEY IN JS



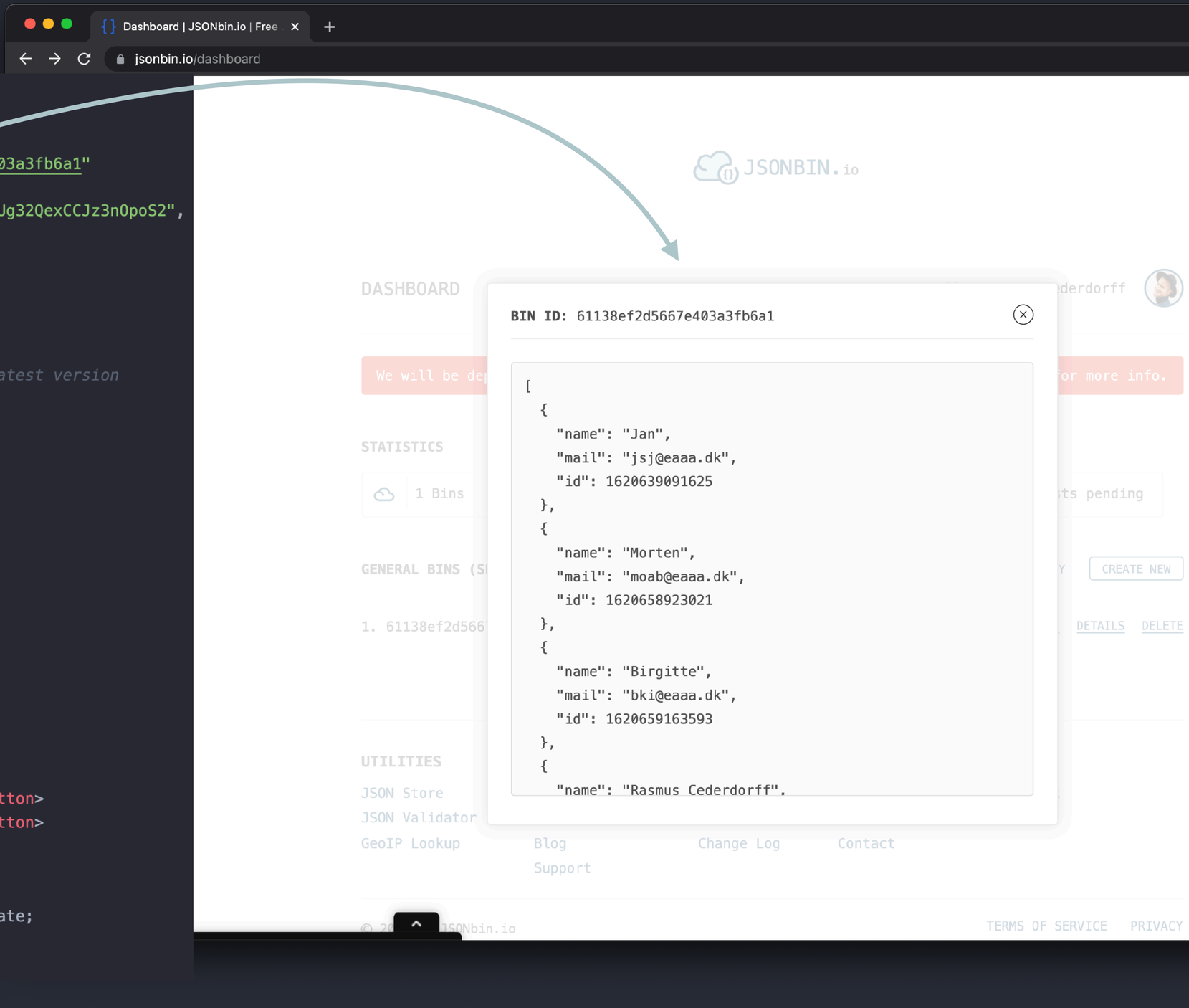

```
// ========== GLOBAL VARIABLES ==========
let _users = [];
let _selectedUserId;
const _baseUrl = "https://api.jsonbin.io/v3/b/6141af694a82881d6c4f5182";
const _headers = {
  "X-Master-Key": "$2b$10$iAOLguBJ8cd9y87RJTcXy.hlVPE8tVaiaURQ1q9DJH2YLmj.4pmhq",
  "Content-Type": "application/json"
};
```

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




```
// ========== 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"
};
```

```
// ========== GLOBAL VARS ==========
let _users = [];
let _selectedUser;
const _baseUrl = "https://api.jsonbin.io/v3/b/61138ef2d5667e403a3fb6a1"
const _headers = {
  "X-Master-Key": "$2b$10$Uf1lbMtIPrrWeneN3Wz6JuDcyBuOz.1LbHiUg32QexCCJz3nOpoS2",
  "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*/ `
      <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);
}
```



BACK

READY TO CODE

READ USERS

BACK

- 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);
}
```

Docs: https://jsonbin.io/api-reference/v3/bins/read

CREATE USER

BACK

- 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

BACK

- 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 arra
}
```

Docs: https://jsonbin.io/api-reference/v3/bins/update

UPDATE USER

BACK

- 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);
  //...
}
```

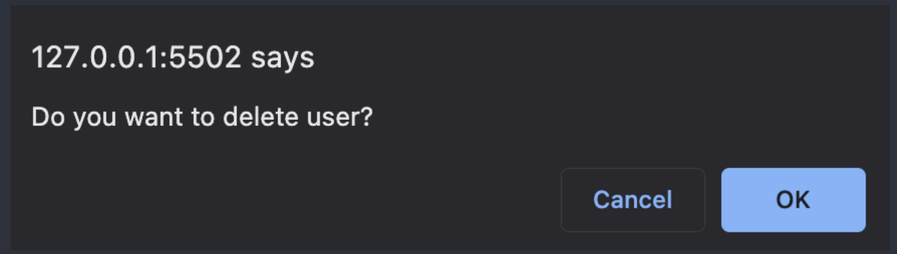
```
async function updateUser() {
  // references to input fields
  const nameInput = document.querySelector("#name-update");
  const mailInput = document.querySelector("#mail-update");
  // find user to update by given user id
  const userToUpdate = _users.find(user => user.id === _selectedUserId);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

DELETE USER

BACK

- Create som 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, matchting the styles of your SPA.

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



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}<
        <p>Age: ${data.selectedUser.age}, Gender: ${data.selectedUser.gen
        <p>Number of matches: ${data.matchCount}</p>
        <button type="button" class="update" onclick="showUserUpdate(${da
        <button type="button" class="delete" onclick="deleteUser(${data.s
    </article>
  `;
  for (const user of data.matches) {
    htmlTemplate += /*html*/ `
      <article onclick="showUser(${user.id})">
        <h3>${user.firstname} ${user.lastname}</h3>
        <p>Age: ${user.age}, Gender: ${user.gender}</p>
      </article>
      `;
  }
  document.querySelector("#guest-user").innerHTML = htmlTemplate;
  document.querySelector("#user .title").innerHTML = data.selectedUser.fi
  showLoader(false);
}

// ========== CREATE ==========
/**
 * creates a new user and saving in JSON file using the php backend servi
 */
async function createUser(firstname, lastname, age, haircolor, countryNam
```

You are welcome to get inspired by the existing project on GitHub and reuse the algorithm from `users-match-backend` or your mandatory assignment in Backend.

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

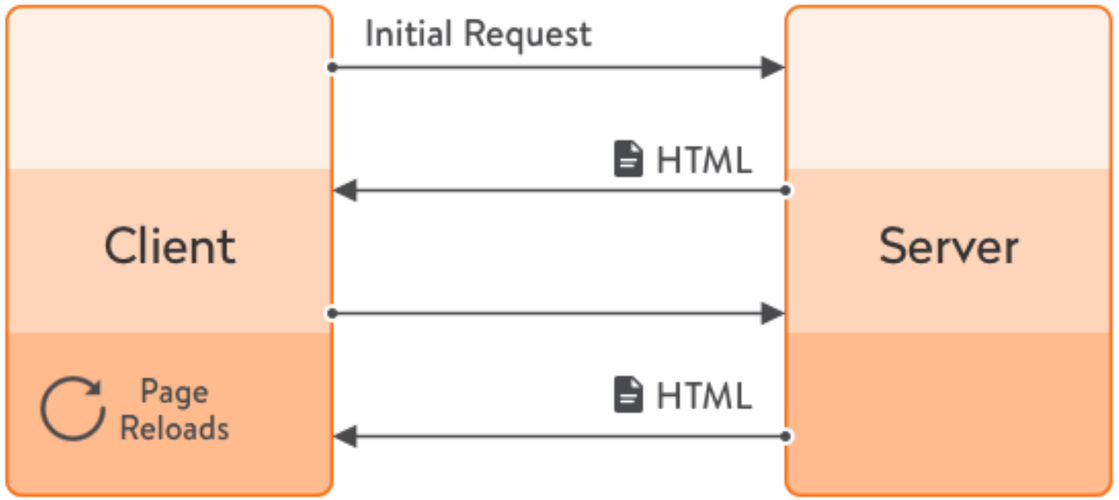
SINGLE PAGE APPLICATION & ROUTER

SINGLE PAGE APPLICATION

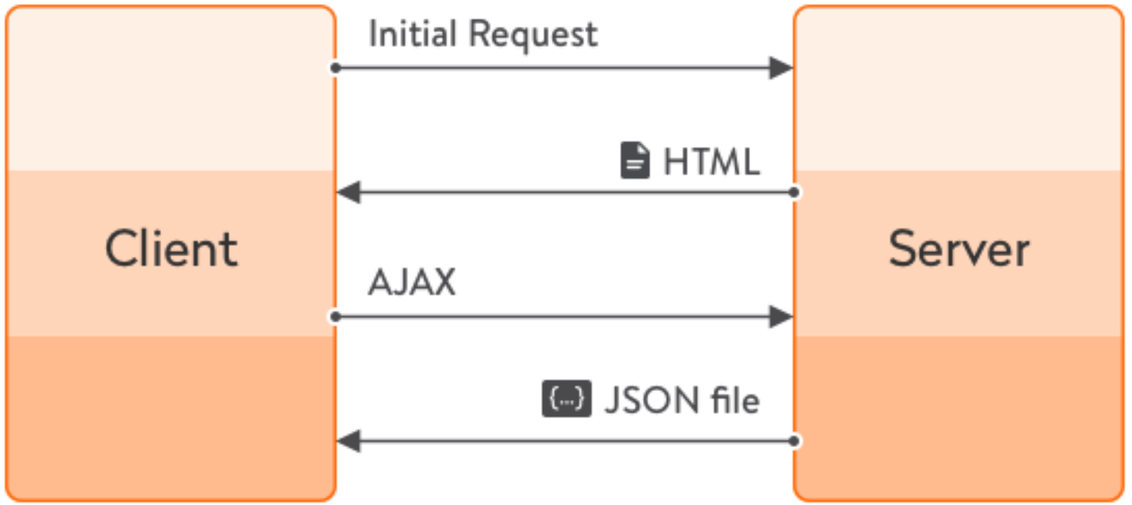
“A single-page application is an application that loads a single HTML page and all the necessary assets (such as JavaScript and CSS) required for the application to run. Any interactions with the page or subsequent pages do not require a round trip to the server which means the page is not reloaded.”

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

Multi-page app lifecycle

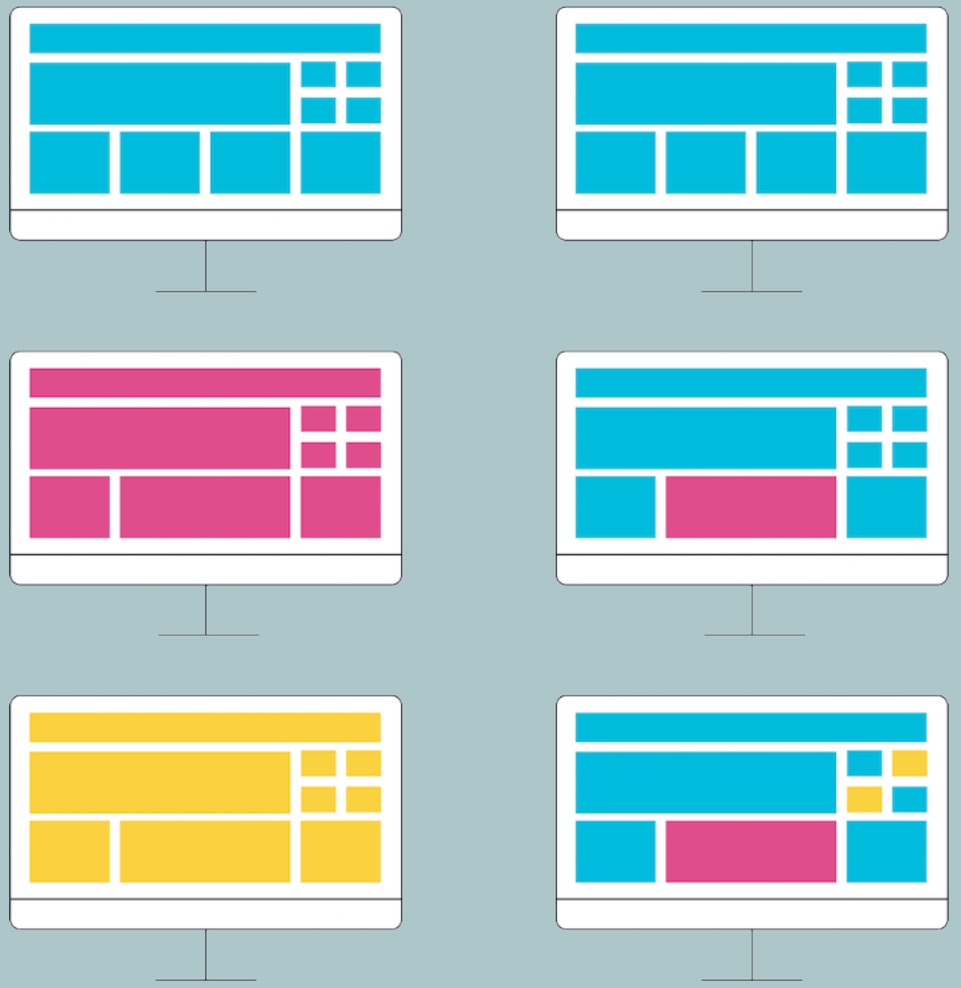


Single-page app lifecycle



https://yalantis.com/blog/single-page-apps-vs-multiple-page-apps/

MPA VS SPA




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.

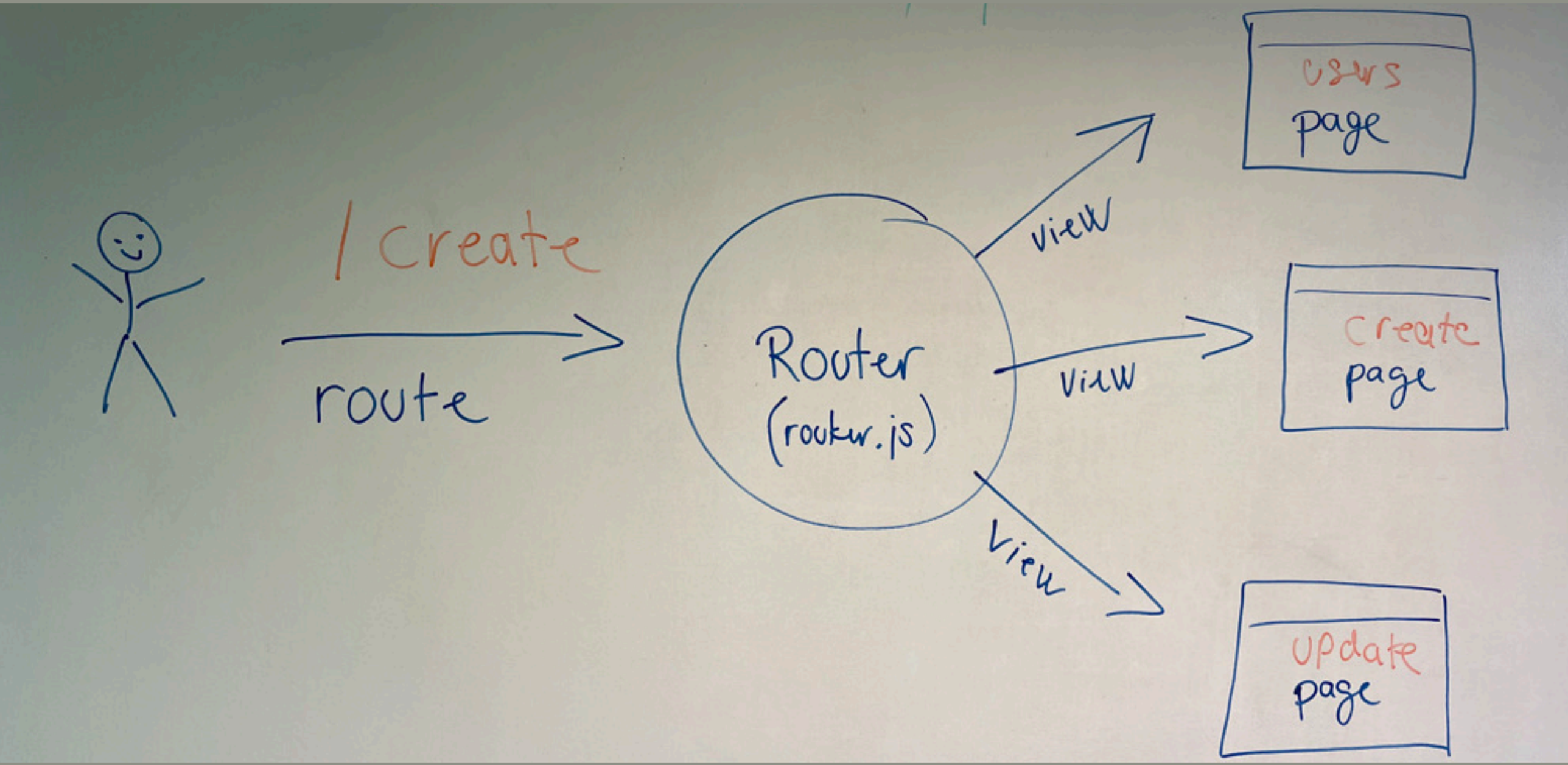
https://medium.com/@fro_g/routing-in-javascript-d552ff4d2921

S P A & R O U T E R

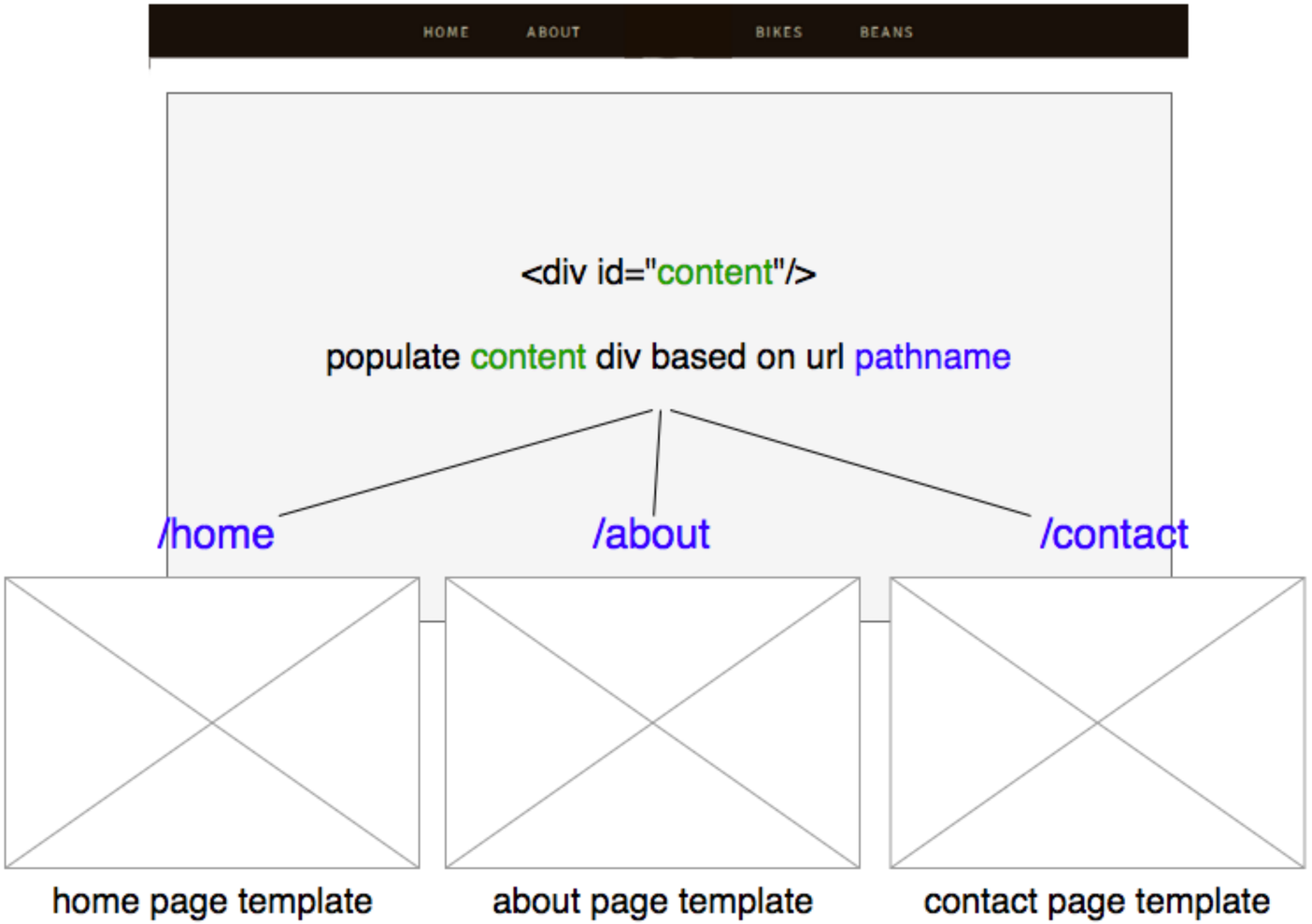
In an SPA the router is in charge of displaying the right view by watching changes on the URL.

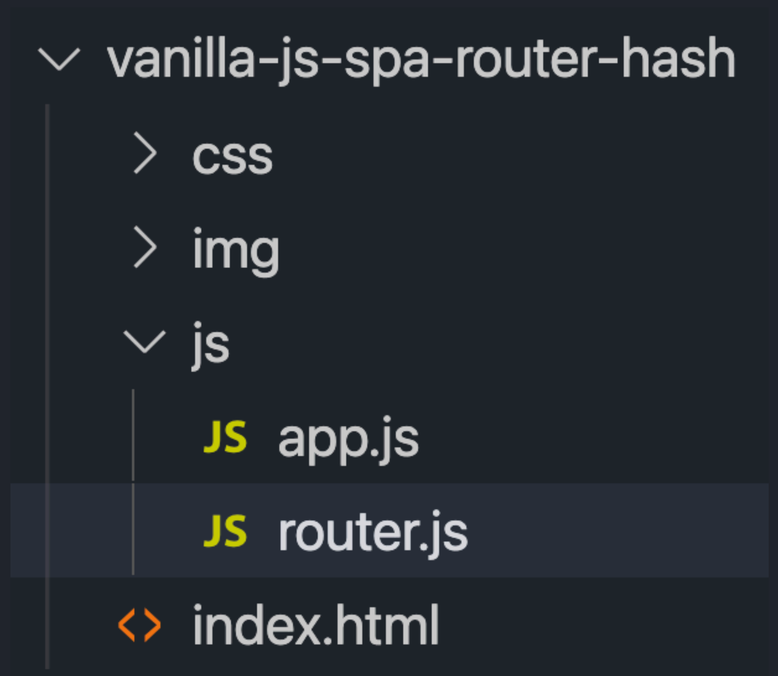
The router makes it looks like the web app has multiple pages and HTML files. But it’s just changing the view, what is displayed, in side of the single `index.html` file. The page transition and change of content, is done dynamically by JavaScript (`router.js`).

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

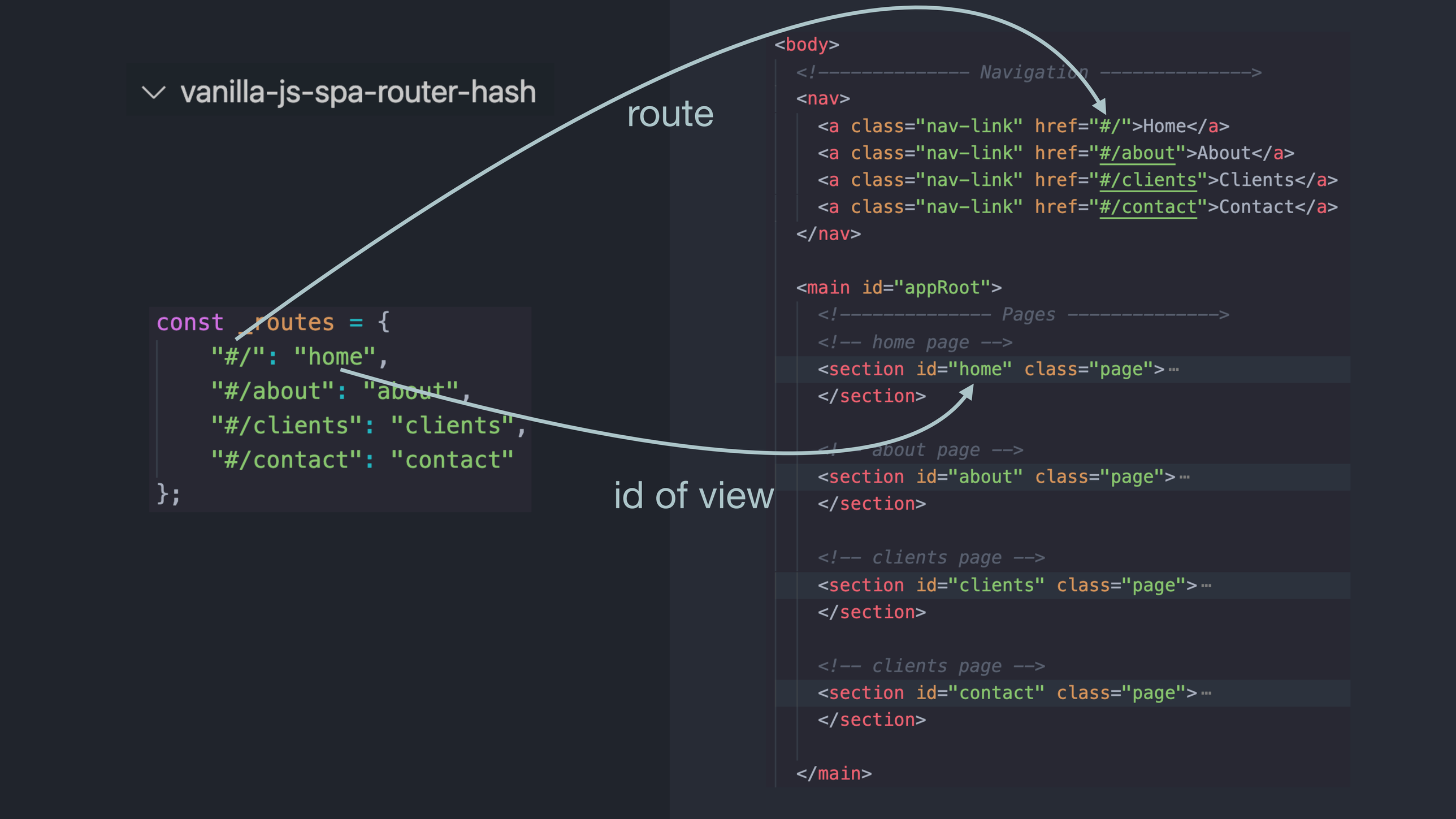

/create
route
Router
(router.js)
view
view
view
users
page
create
page
update
page

www.webpage.com/pathname






```
 * 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", ""); // remove
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
 */
```


vanilla-js-spa-router-hash
route
id of view
const routes = {
"#/": "home",
"#/about": "about",
"#/clients": "clients",
"#/contact": "contact"
};
<body>
<!-------------- Navigation -------------->
<nav>
Home
About
Clients
Contact
</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>
<!-- clients page -->
<section id="contact" class="page">…
</section>
</main>

change default event for all `.nav-link` items

```html
<!-------------- 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>
```

```js
/**
 * 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`

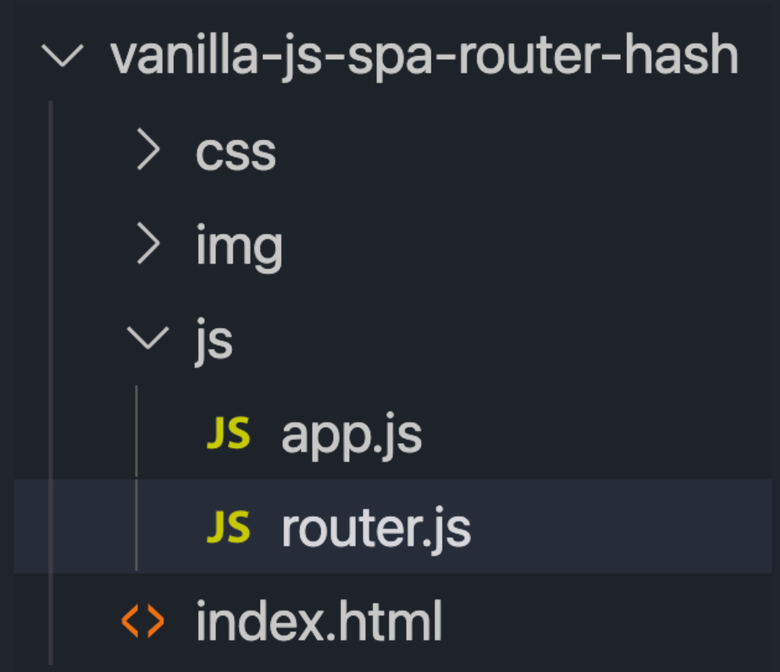
```html
<!-------------- Navigation -------------->
<nav>
    <a class="nav-link" href="#/">Home</a>
    <a class="nav-link" href="#/about">About</a>
    <a class="nav-link" href="#/clients">Clients</a>
    <a class="nav-link" href="#/contact">Contact</a>
</nav>
```

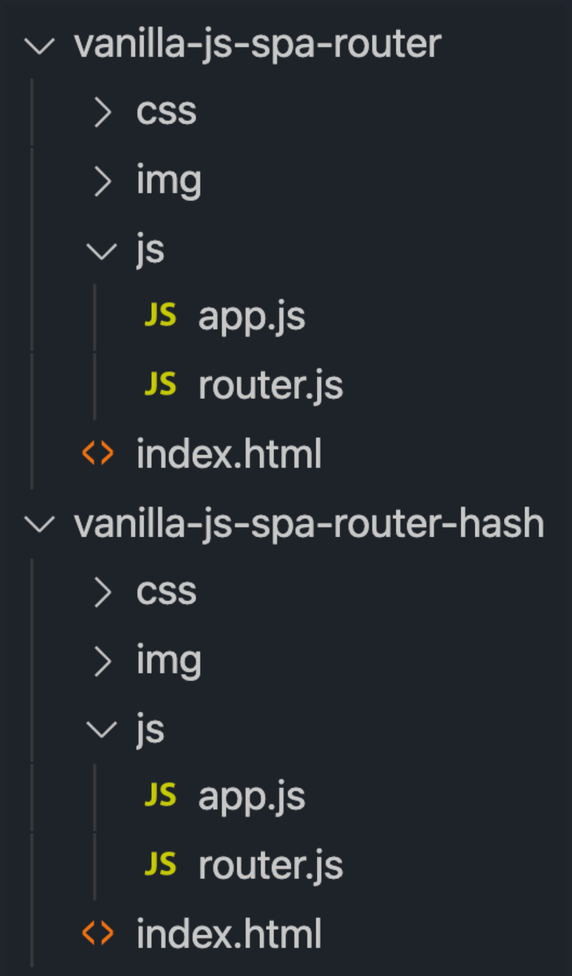
```js
/**
 * Navigating SPA to specific page by given path
 */
function navigateTo(path) {
    window.history.pushState({}, path, _basePath + path);
    showPage(path);
}

/**
 * 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

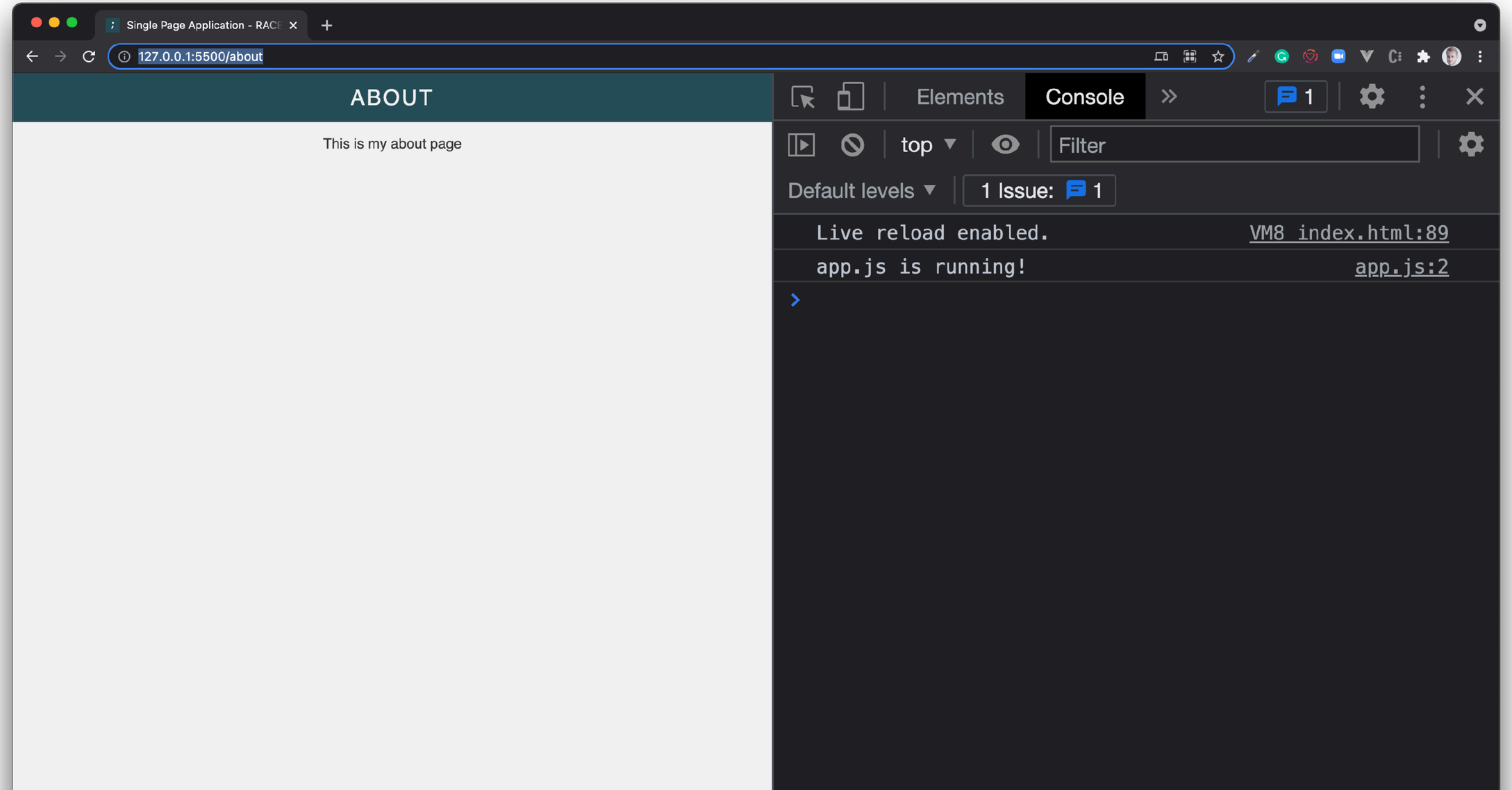
- Inspect the `index.html` & `router.js` in `vanilla-js-spa-router-hash`
- Add another page called users.
- Remember to add a new section for users page, a nav-link and the route in `router.js`.
- Test your are able to navigate to the users page.
- Change the clients page to a posts page. Remember to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



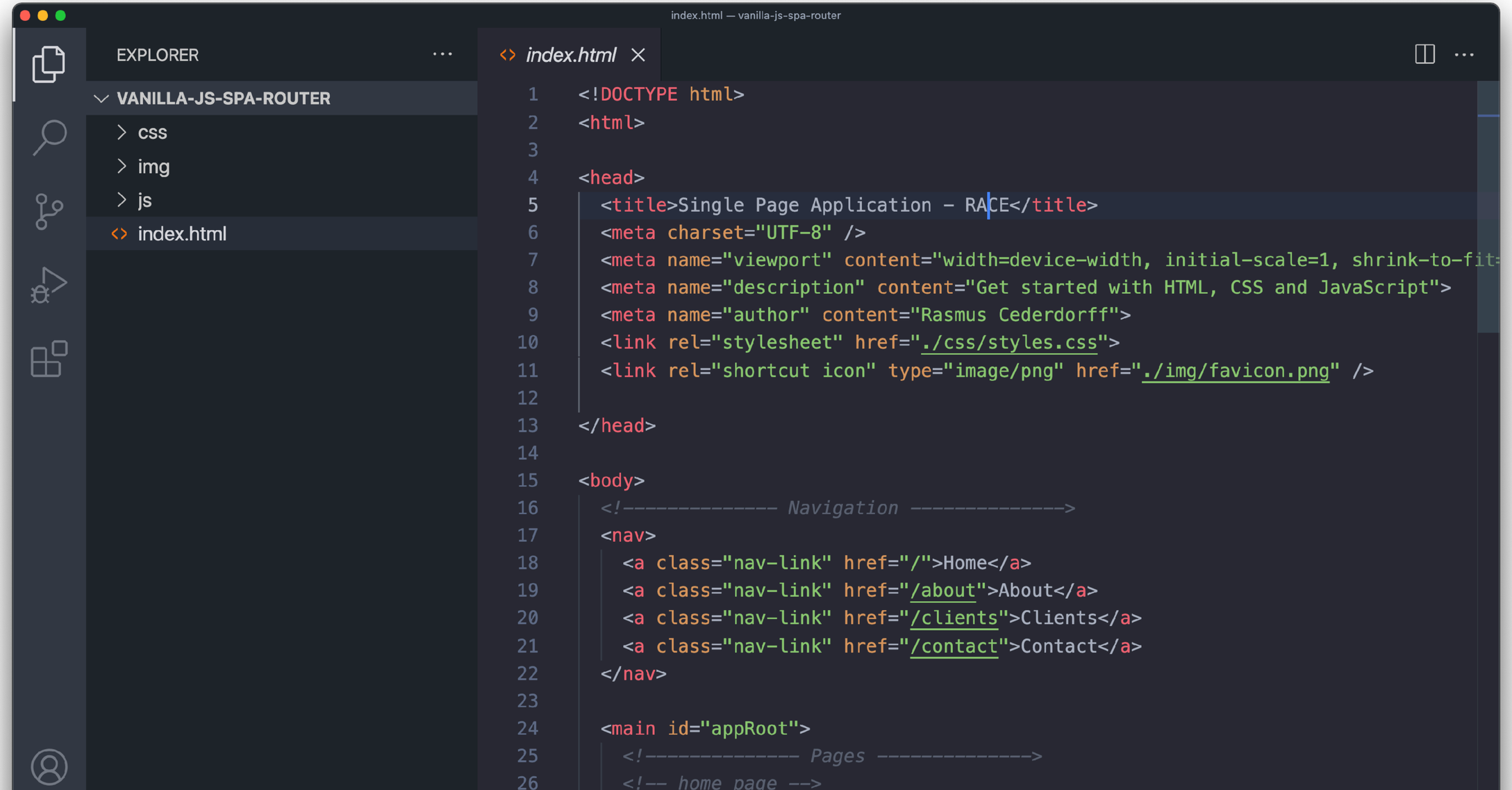


WHAT THE “#/HASH”?

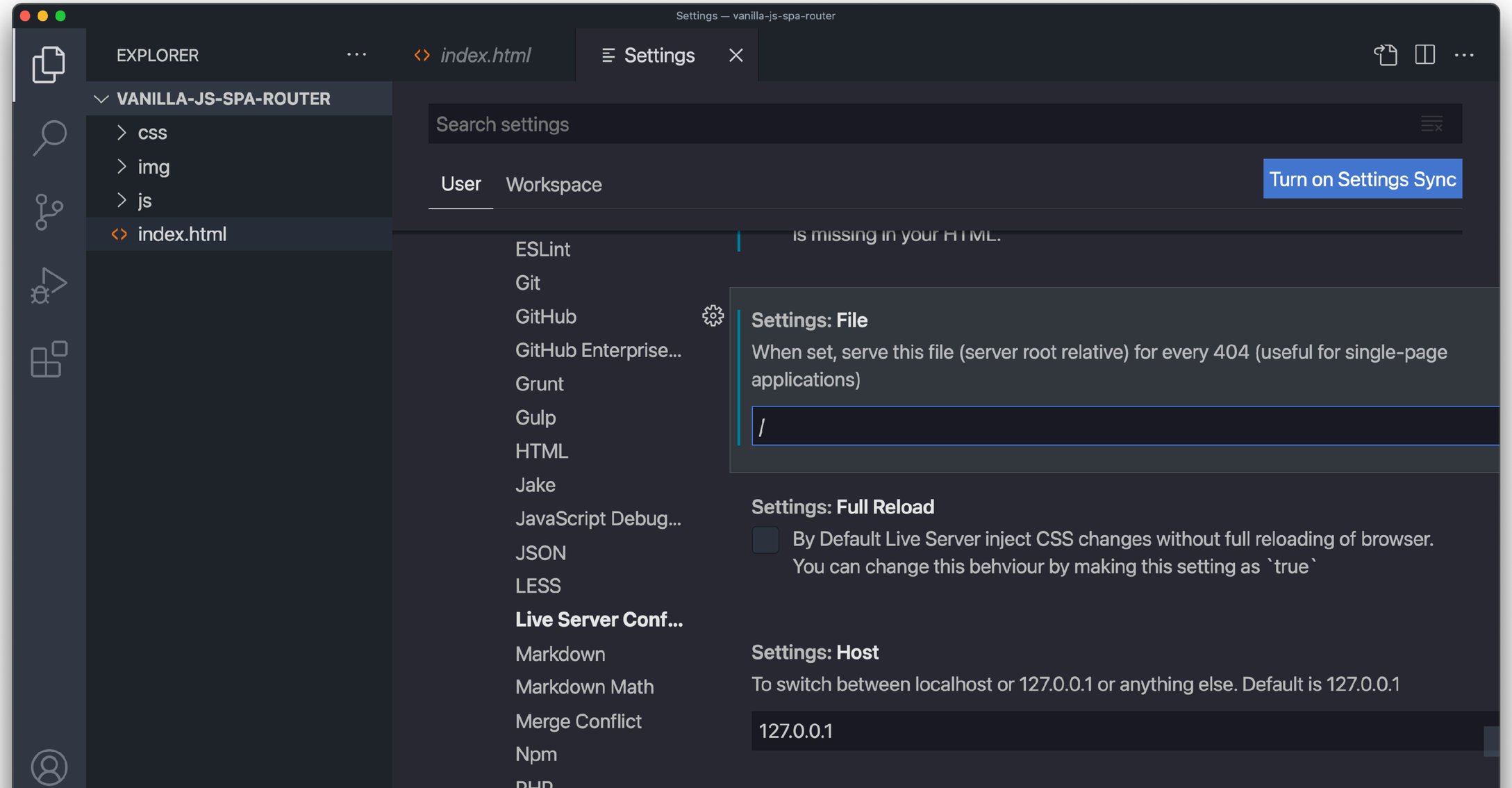
WITHOUT HASH



OPEN & RUN THIS PROJECT ONLY

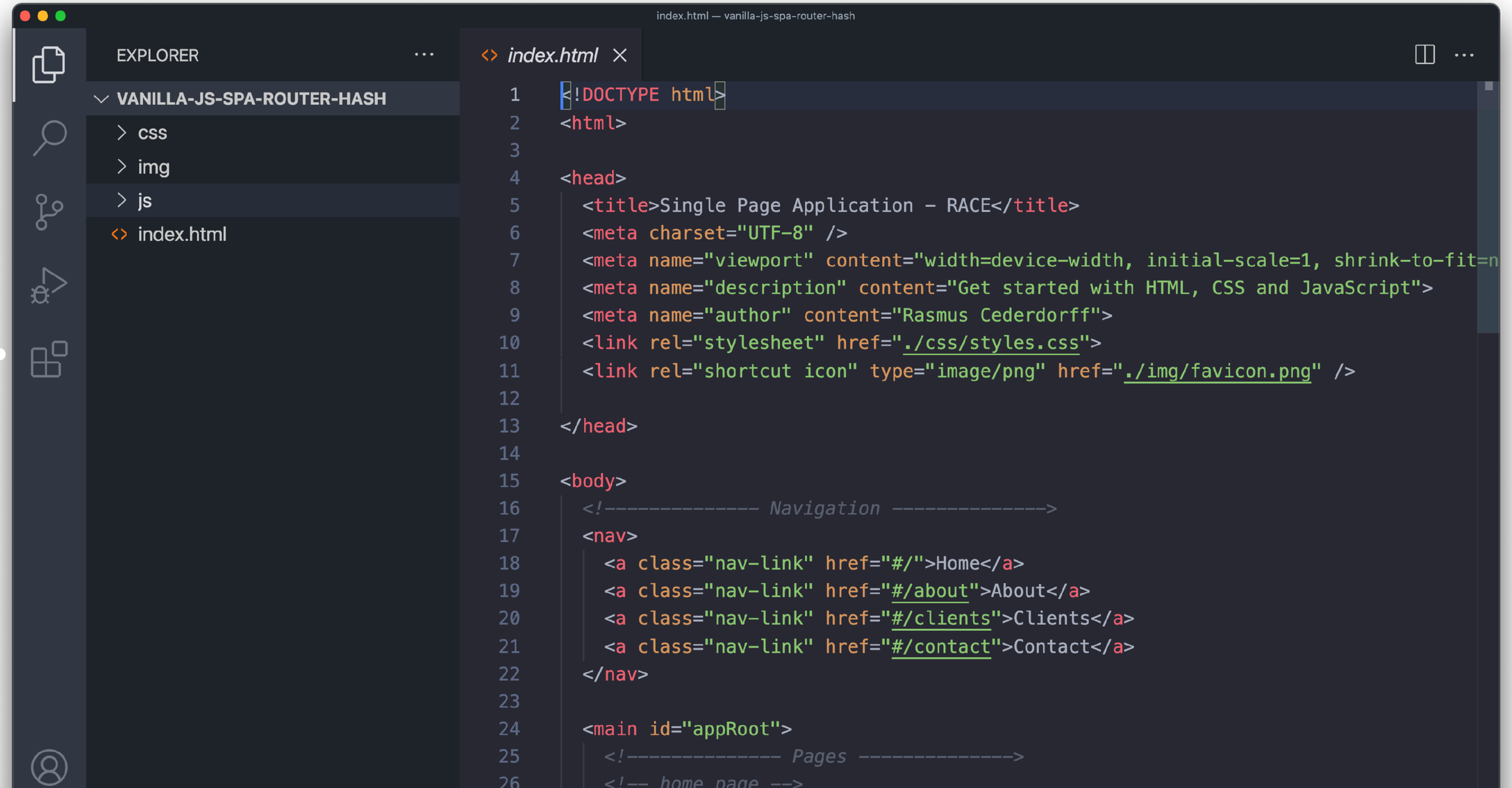


CHANGE LIVE SERVER ROOT



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

OR USE HASH 🤷


index.html — vanilla-js-spa-router-hash
EXPLORER
VANILLA-JS-SPA-ROUTER-HASH
css
img
js
index.html
index.html
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5 <title>Single Page Application - RACE</title>
6 <meta charset="UTF-8" />
7 <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=n
8 <meta name="description" content="Get started with HTML, CSS and JavaScript">
9 <meta name="author" content="Rasmus Cederdorff">
10 <link rel="stylesheet" href="./css/styles.css">
11 <link rel="shortcut icon" type="image/png" href="./img/favicon.png" />
12
13 </head>
14
15 <body>
16 <!-------------- Navigation -------------->
17 <nav>
18 Home
19 About
20 Clients
21 Contact
22 </nav>
23
24 <main id="appRoot">
25 <!-------------- Pages -------------->
26 <!-- 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

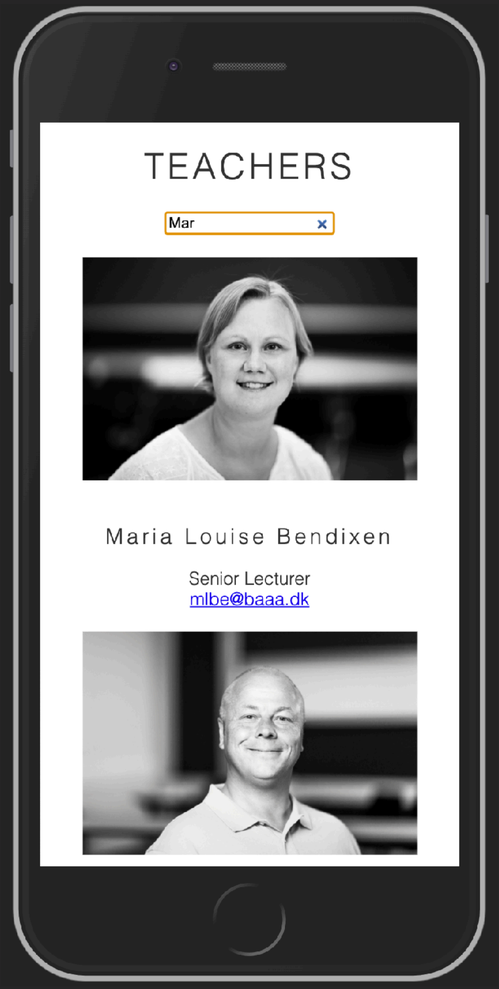
https://javascript.info/array#loops

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);
}
```

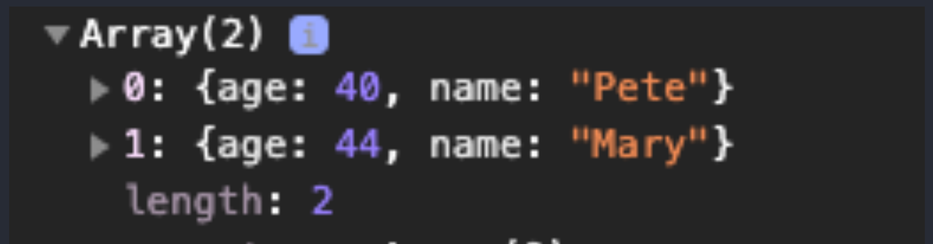


array-teachers-search

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 METHODS

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

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:

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

For instance:

```
let users = [
  {id: 1, name: "John"},
  {id: 2, name: "Pete"},
  {id: 3, name: "Mary"}
];

// returns array of the first two users
let someUsers = users.filter(item => item.id < 3);

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

PRODUCTS | ADD PRODUCT

Elements | Console | Sources | 1

```
<section id="products" class="page" style="display: block;">
  <header class="topbar">…</header>
  <section class="tools-grid">  grid
    <label for="sortBy">
      "Order by: "
      <select id="sortBy" onchange="orderBy(this.value)">…
      </select> == $0
    </label>
    <label for="outOfStock" class="checkmark-container">…
    </label>
    <input type="search" placeholder="Search" onkeyup="search
    (this.value)" onsearch="search('')">
  </section>
  <section id="products-container" class="grid-container">
  grid
```

html body section#products.page section.tools-grid label select …

Styles | Computed | Layout | Event Listeners | DOM Breakpoints

Filter :hov .cls

```
element.style {
}
select {                                              main.css:366
  background-color: var(--green);
  border: none;
  color: var(--text-color-light);
  padding: 1em;
  margin: 1em;
  max-width: 350px;
}
select:not(:-internal-list-box) {          user agent stylesheet
  overflow: visible !important;
}
```

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 178.09×72

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

PRODUCTS ADD PRODUCT

Elements Console Sources 1

```
<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="showHid
    eOfStock(this.checked)" checked> == $0
    <span class="checkmark">…</span>
  </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 {                    main.css:313
  position: absolute;
  opacity: 0;
  cursor: pointer;
  height: 0;
  width: 0;
}
input {                                         main.css:247
  margin: ▸ 1em auto;
  width: 100%;
  max-width: 350px;
```

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

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:

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

For instance:

```
let users = [
  {id: 1, name: "John"},
  {id: 2, name: "Pete"},
  {id: 3, name: "Mary"}
];

// returns array of the first two users
let someUsers = users.filter(item => item.id < 3);

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);
}
```

Table of contents

Related Topics

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