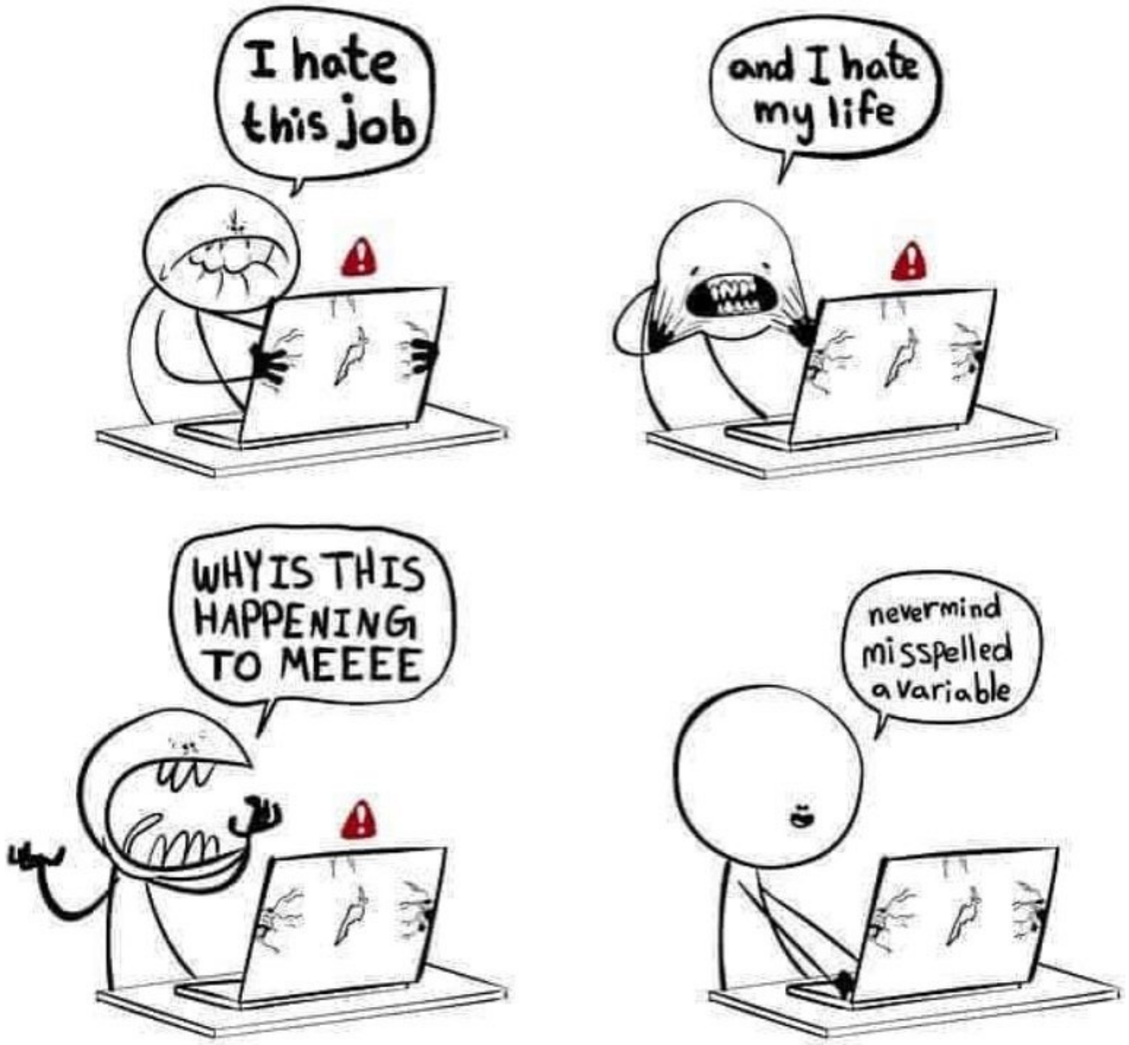
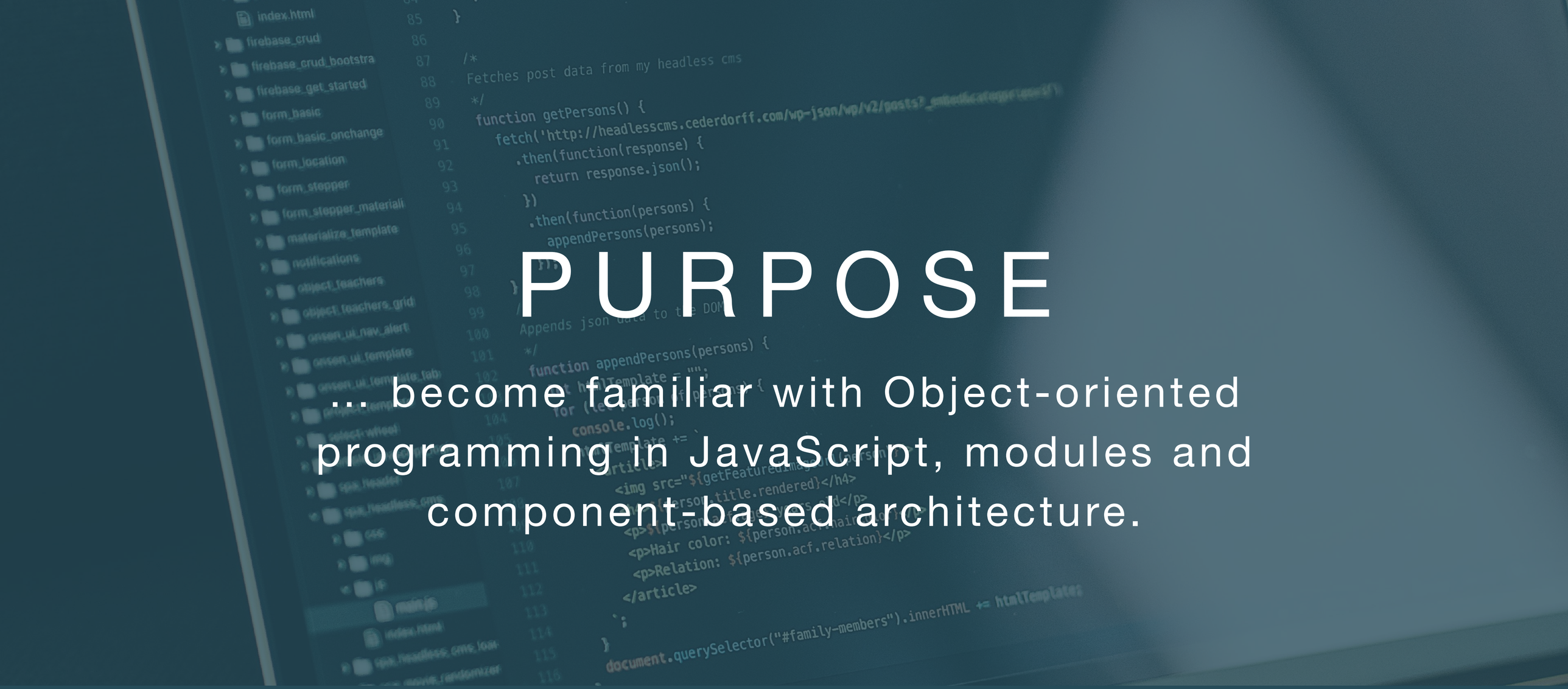


Well, programming can be hard 🤷

OBJECT-ORIENTED JS, MODULES & COMPONENTS

FRONTEND DEVELOPMENT

BUSINESS ACADEMY AARHUS
UNIVERSITY OF APPLIED SCIENCES



PURPOSE

... become familiar with Object-oriented programming in JavaScript, modules and component-based architecture.

AGENDA

PRACTICALS: 2ND SEMESTER

DATING SPA WRAP UP

FUNCTION VS OBJECT-ORIENTED JS

CLASSES & OBJECTS

MODULES, IMPORT & EXPORT

EXERCISES

Dating SPA Wrap up

Person Class & Teacher Class

Import-Export

TodoList Component & UserList Component

Object-oriented Canvas User CRUD

Dating SPA using Classes & Modules


ELECTIVES
2ND SEMESTER

2ND SEMESTER

ADVANCED WEB DEVELOPMENT (10 ECTS)

- FULLSTACK JAVASCRIPT (MERN STACK)

ELECTIVE MODULES (20 ECTS)

- CONTENT MANAGEMENT SYSTEMS (10 ECTS)
- XR DEVELOPMENT (10 ECTS)
- MOBILE DEVELOPMENT (10 ECTS)
- PROGRESSIVE WEB APPS (10 ECTS)

baaa.dk/programmes/bachelor-s-top-up-degree/web-development

2ND SEMESTER

CONTENT MANAGEMENT SYSTEMS (10 ECTS)

Techniques & tools for developing Content Management based web applications.

Different types of CMS (traditional, decoupled, headless).

Criteria for selection of a CMS.

How to use and extend a CMS.

Hands on with different CMS’. Ex. Drupal, Strapi, Wordpress, Umbraco, self-chosen

2ND SEMESTER

XR DEVELOPMENT (10 ECTS)

Techniques & tools for developing
contemporary X Reality (XR) applications,
including virtual reality (VR),
mixed reality (MR) & augmented reality (AR).

Programming paradigms in contemporary
XR tools & frameworks.

User Experience & XR - analyze users &
situations in addition to XR.

2ND SEMESTER

MOBILE DEVELOPMENT (10 ECTS)

Techniques & tools for developing applications
for mobile devices.

Programming techniques, UI development &
web integration that are necessary
to develop mobile app - from idea to build to distribution
on one and/or various platforms.

Hands on with frameworks & tools. Ex. Flutter, Expo,
Ionic, Unity, React Native & JS Framworks)

2ND SEMESTER

PROGRESSIVE WEB APPS (10 ECTS)

Techniques & tools for developing installable web apps.

How to use device capabilities.

Service workers, caching & web performance.

Offline vs online.

Relevant web technologies, software, libraries
& frameworks. Ex JS Frameworks, React PWA, Preact,
Ionic PWA, Module Bundlers.

DATING SPA

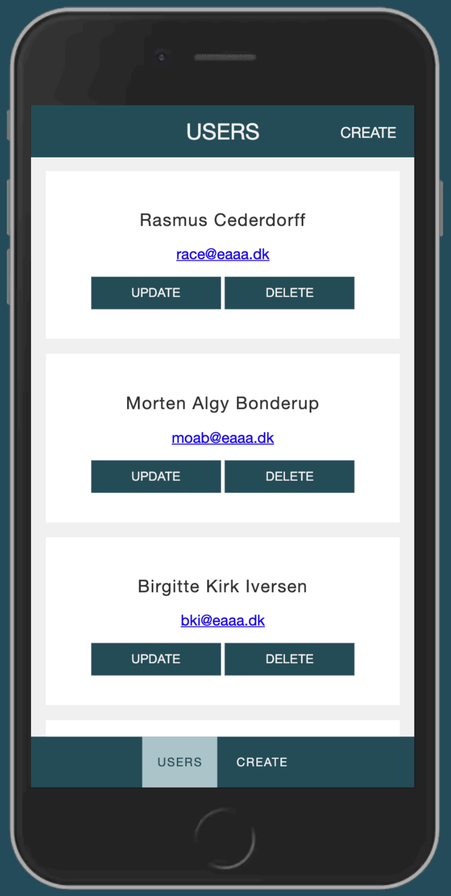
WRAP UP

FETCH, HTTP REQUEST & RESPONSE

WEB APP

API

DATA SOURCE



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

REQUEST

JSON DATA

RESPONSE

FETCH: HEADERS, BODY & HTTP METHODS

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 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"

```
const id = Date.now(); // dummy generated user id
const newUser = { // declaring a new js object with the form values
  id, firstname, lastname, age, haircolor, countryName, gender, lookingFor, image
};
console.log(newUser);
// 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
```

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();
```

FETCH WITH HEADERS & BODY

... AND HTTP METHOD "POST"

```
/**
 * Upload image to php backend
 */
async function uploadImage(imageFile) {
  let formData = new FormData();
  formData.append("fileToUpload", imageFile);

  const response = await fetch("backend/upload.php", {
    method: "POST",
    headers: { "Access-Control-Allow-Headers": "Content-Type" },
    body: formData
  });
  // waiting for the result
  const result = await response.json();
  console.log(result);
  return result;
}
```

FETCH WITH HEADERS & BODY

... AND HTTP METHOD "PUT"

```
async function updateUser(firstname, lastname, age, haircolor, countryName, gender, lookingFor, image) {
  const userToUpdate = { // declaring a new js object with the form values
    id: _selectedUserId, firstname, lastname, age, haircolor, countryName, gender, lookingFor, image
  };
  // put user to php userService using fetch(...)
  const response = await fetch(_baseUrl + "?action=updateUser", {
    method: "PUT",
    headers: { "Content-Type": "application/json; charset=utf-8" },
    body: JSON.stringify(userToUpdate) // 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
  _users = result;
  appendUsers(result); // update the DOM using appendUsers(...)
  showUserPage(); // navigating back to user page
}
```

FETCH WITH HEADERS

... AND HTTP METHOD "DELETE"

```
async function deleteUser() {
  const deleteUser = confirm("Do you want to delete user?");
  if (deleteUser && _selectedUserId) {
    // delete user using php userService and fetch(...)
    const response = await fetch(`${_baseUrl}?action=deleteUser&userid=${_selectedUserId}`, {
      method: "DELETE",
      headers: { "Content-Type": "application/json; charset=utf-8" }
    });
    // waiting for the result
    const result = await response.json();
    console.log(result); // the result is the new updated users array
    _users = result;
    appendUsers(result); // update the DOM using appendUsers(...)
    navigateTo("#/"); // navigating back to front page
  }
}
```

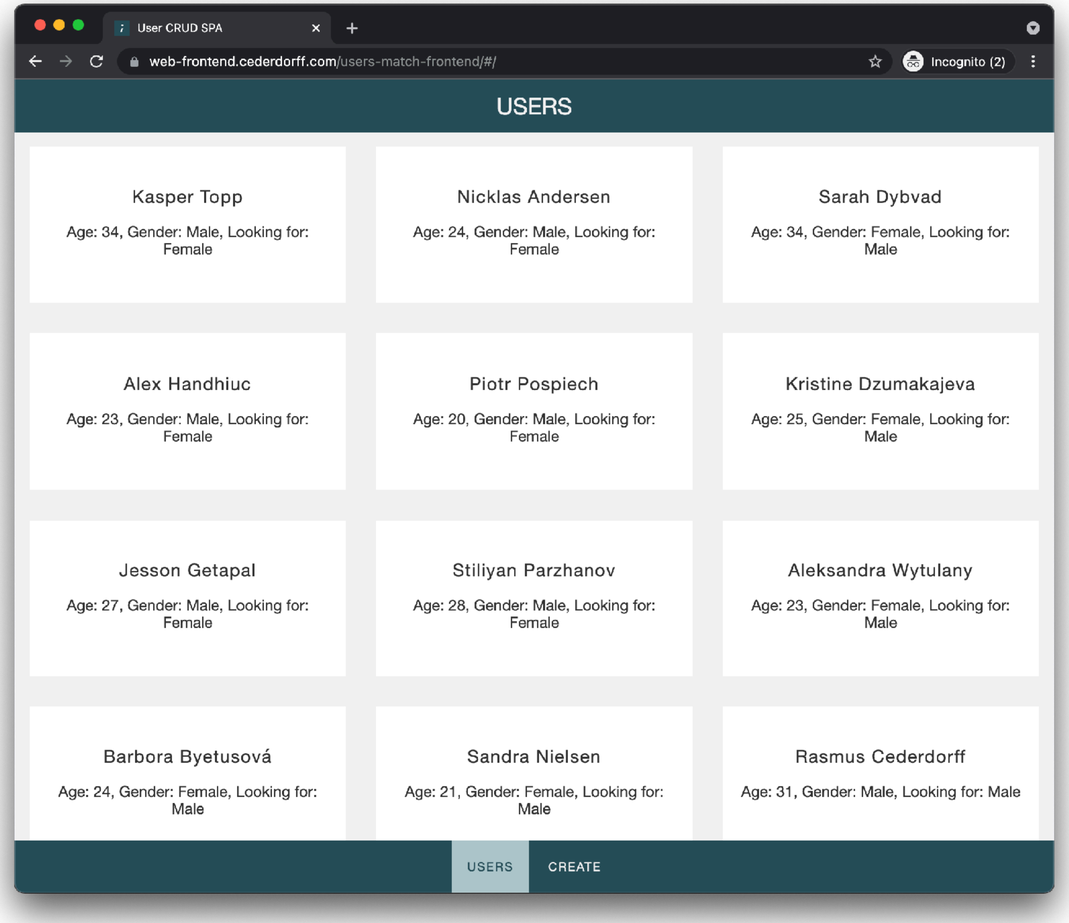
STEPS

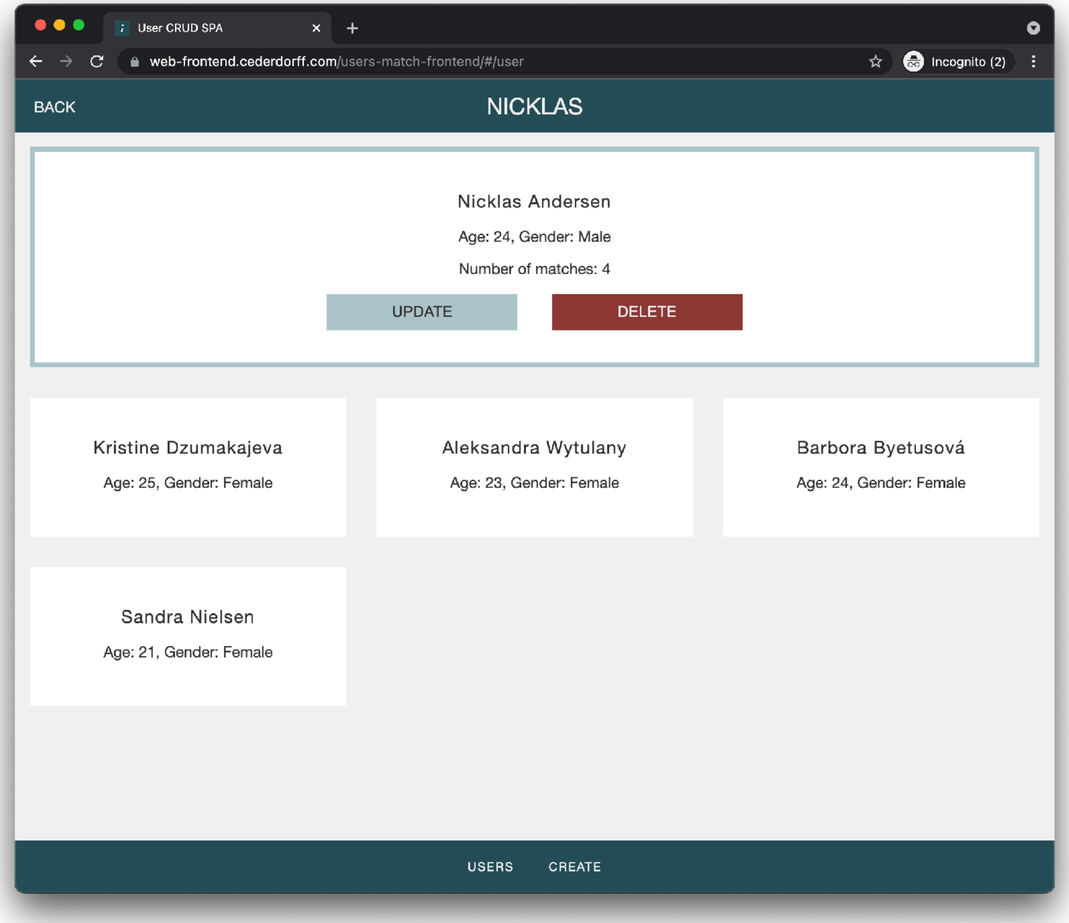
Template: `dating-spa-template`

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

DATING SPA JSONBIN

BACK

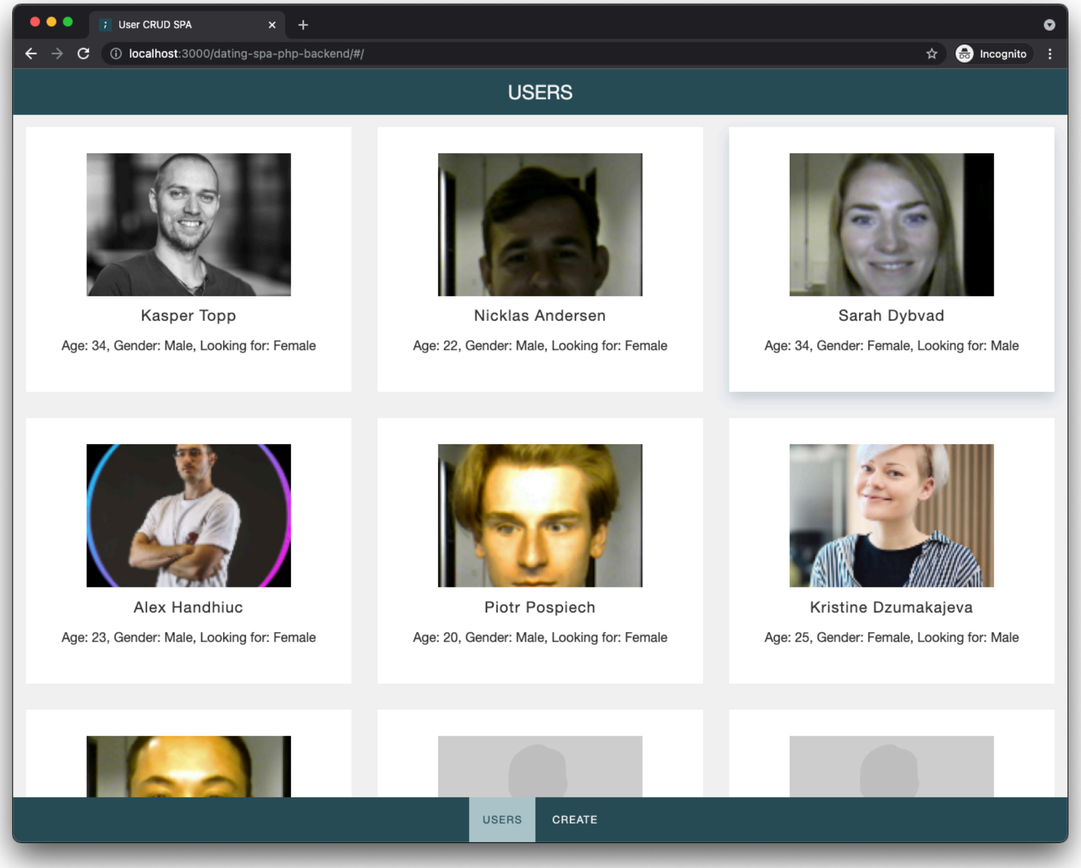


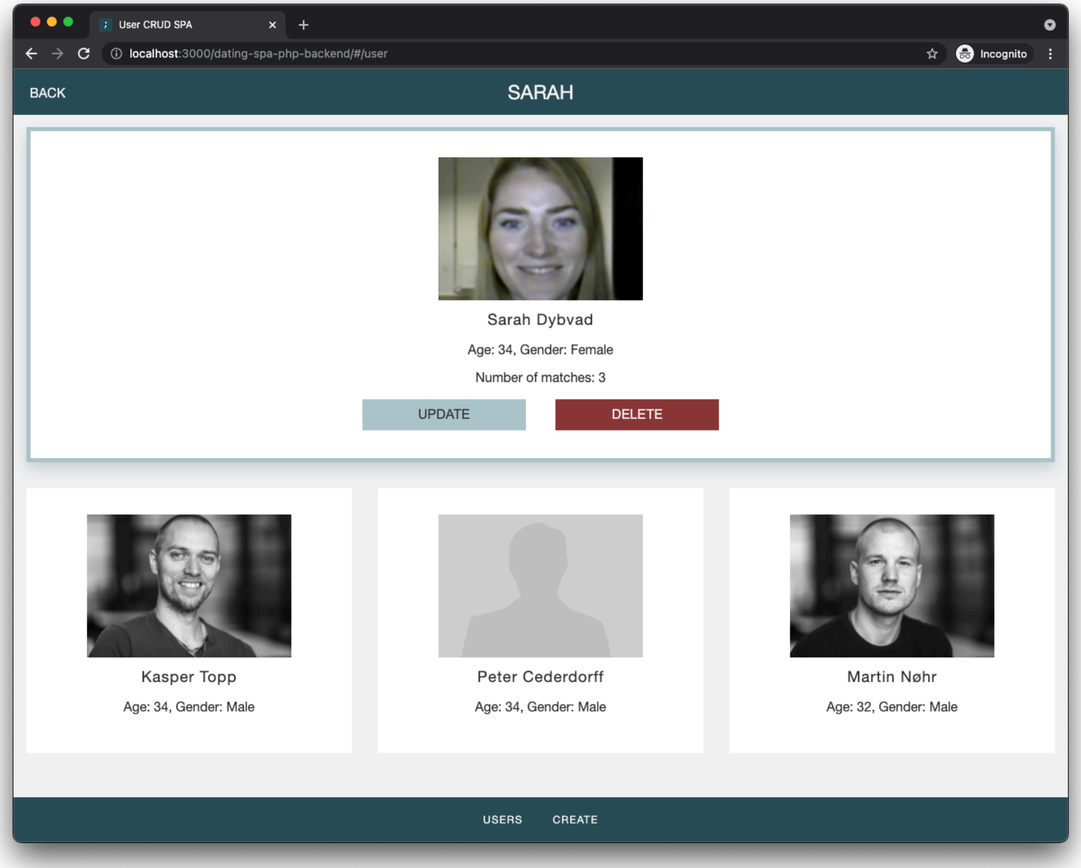


Solution: `dating-spa-jsonbin`

DATING SPA PHP BACKEND

BACK





Solution: `dating-spa-php-backend`

WRAP UP IN PAIRS

BACK

1. Use your own solution and/or `dating-spa-jsonbin`
2. Discuss and explain the implementation of:
    1. `_routes` and `router.js`
    2. Reading users from JSONBIN (`loadUsers()`)
    3. `updateJSONBIN(…)` in order to create, update & delete users + use of fetch, http methods, headers & body.
    4. Detail View/ user profile with matches. How did you implement the algorithm?
    5. Global and local variables
3. Compare your solution with `dating-spa-jsonbin`. What are the differences?
4. Any thoughts on the structure of `main.js` and the app overall?
5. If you havn't, finish your dating spa and/or do improvements.
6. Add themes, terms, questions and comments to the Padlet: eaaa.padlet.org/race/frontend_programming
7. `if(time){`implement PHP Backend Service`}`

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`

Solution: `dating-spa-php-backend`

WRAP UP BY RACE

1. How to run and test the solution.
2. Fetch: methods, headers and body
3. PHP user service and `['action']`s
4. How to calculate and get matches (client or server side?)
5. The use of `localStorage`
6. Create and image upload
7. Image preview and image
8. Init App (`init()` in `main.js`)

Solution: `dating-spa-php-backend`

THE STRUCTURE OF THE DATING SPA?

FUNCTION VS OBJECT-ORIENTED JAVASCRIPT

OBJECT-ORIENTED JAVASCRIPT

"The basic idea of OOP is that we use objects to model real world things that we want to represent inside our programs, and/or provide a simple way to access functionality that would otherwise be hard or impossible to make use of."

"Objects can contain related data and code, which represent information about the thing you are trying to model, and functionality or behavior that you want it to have."

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_JS


ONE WAY
DEPT OF TRANSPORTATION
ONE WAY
DEPT OF TRANSPORTATION

WHY?

DIFFERENT WAYS & PARADIGMS OF SOLVING ONE PROBLEM

NO PARADIGM SEEMS TO BE BETTER THAN ANOTHER

STRUCTURE, ORGANISATION & SCALABILITY

As our application grows bigger, we want to keep a good consistant structure & organisation of our code. We need different approaches.

DIFFERENT WAYS OF STRUCTURING & ORGANISING YOUR CODE

FUNCTION-ORIENTED - STRUCTURED IN FUNCTIONS

OBJECT-ORIENTED - STRUCTURED IN OBJECTS

OBJECT-ORIENTED JAVASCRIPT

We use classes to describe a generic structure (data and functionality) of an object.

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_JS

OBJECT-ORIENTED JS

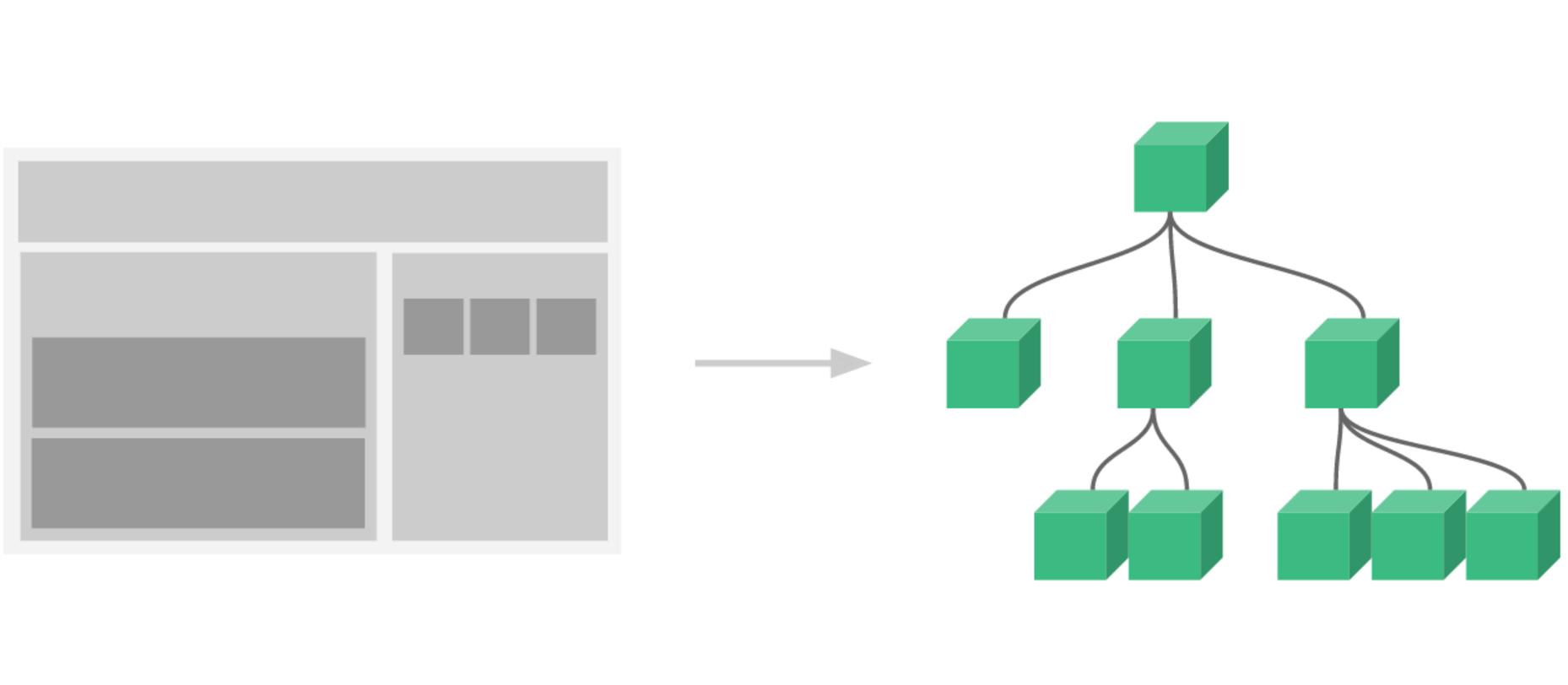
THE USE OF OBJECTS, CLASSES, ENCAPSULATION & INHERITANCE

OBJECT-ORIENTED JS

REUSABLE CODE

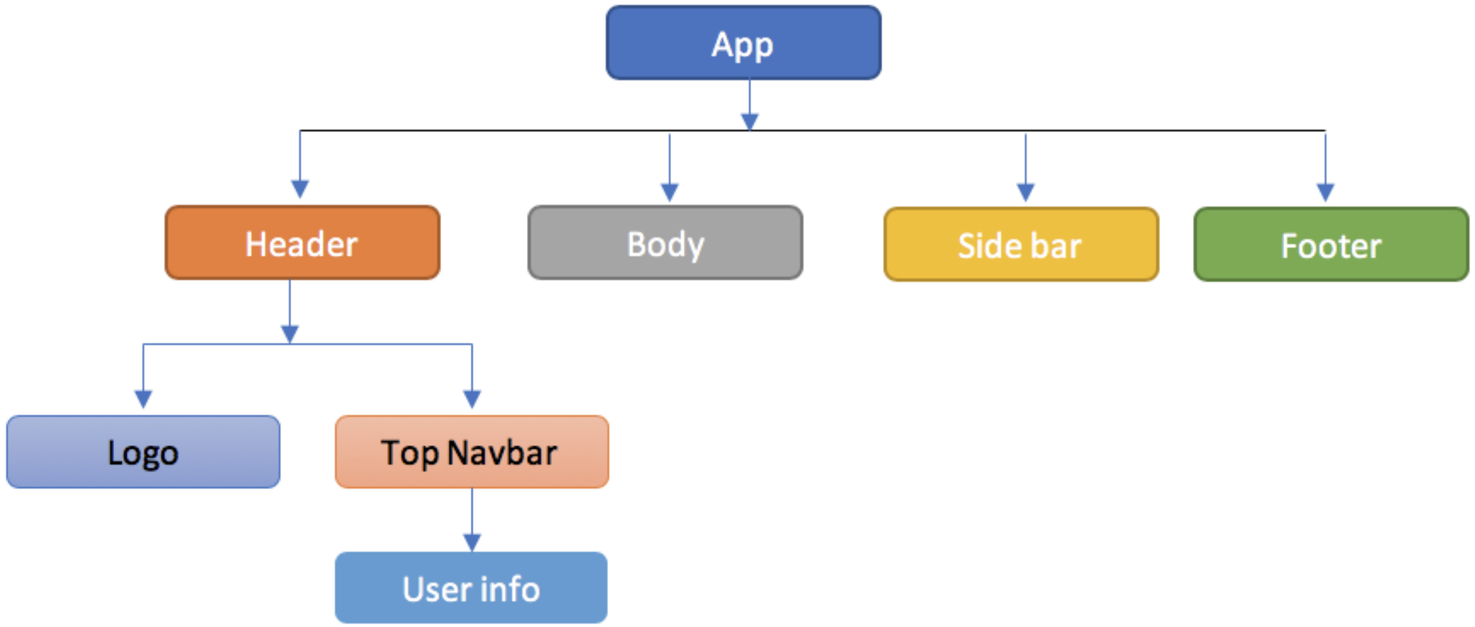
BUILD REUSABLE MODULES & COMPONENTS

COMPONENTS



APP STRUCTURED IN COMPONENTS

APP STRUCTURED IN COMPONENTS



FRAMEWORKS & LIBRARIES

Many of them have the Object-oriented approach

(or component-based approach)

IT'S ALL OBJECTS (& ARRAYS)

FUNCTIONAL VS. OBJECT-ORIENTED PROGRAMMING IN JAVASCRIPT

https://javascript.plainenglish.io/javascript-functional-vs-oop-fb5fbf15a35d

ES6+

MODERN JAVASCRIPT

MODERN JAVASCRIPT

LET & CONST
TEMPLATE STRING
ARROW FUNCTIONS
FETCH
PROMISES
ASYNC & AWAIT
FOR OF LOOP
ARRAY.FIND()
ARRAY.MAP()
ARRAY.REDUCE()
ARRAY.FILTER()
ARRAY.SORT()
ARRAY.CONCAT()
DEFAULT PARAMS
DESTRUCTURING OBJECTS
DESTRUCTURING ARRAYS
OBJECT LITERAL
SPREAD OPERATOR
CLASSES
MODULES
IMPORT & EXPORT

CLASSES

A class is an extensible program-code-template for creating objects, providing initial values for state (member variables) and implementations of behaviour (member functions or methods).

CLASSES

Classes are templates for objects.

CAR CLASS WITH CONSTRUCTOR

```
class Car {
    constructor(name, year) {
        this.name = name;
        this.year = year;
    }
}
```

OBJECT

```
let myCar1 = {
    name: "Ford",
    year: 2014
};
```

GENERIC TEMPLATE

```
class Car {
    constructor(name, year) {
        this.name = name;
        this.year = year;
    }
}
```

```
class MyClass {
  // class methods
  constructor() { ... }
  method1() { ... }
  method2() { ... }
  method3() { ... }
  ...
}
```

CREATING OBJECT USING A CLASS

```
class Car {
    constructor(name, year) {
        this.name = name;
        this.year = year;
    }
}

let myCar1 = new Car("Ford", 2014);
let myCar2 = new Car("Audi", 2019);
```

CLASSES CAN CONTAIN PROPERTIES & FUNCTIONS

```
class Car {
    constructor(name, year) {
        this.name = name;
        this.year = year;
    }
    age() {
        let date = new Date();
        return date.getFullYear() - this.year;
    }
}

let myCar1 = new Car("Ford", 2014);
let myCar2 = new Car("Audi", 2019);

console.log(myCar1.age());
console.log(myCar2.age());
```

CLASSES CAN CONTAIN PROPERTIES & FUNCTIONS

```
class Car {
    constructor(name, year) {
        this.name = name;
        this.year = year;
    }
    age() {
        let date = new Date();
        return date.getFullYear() - this.year;
    }
}

let myCar1 = new Car("Ford", 2014);
let myCar2 = new Car("Audi", 2019);

console.log(myCar1.age());
console.log(myCar2.age());
```

properties

method/function

instantiation

call method on object

WITHOUT A CLASS

```
let myCar1 = {
    name: "Ford",
    year: 2014
};

let myCar2 = {
    name:"Audi",
    year: 2019
};

function age(car){
    let date = new Date();
    return date.getFullYear() - car.year;
}

console.log(age(myCar1));
console.log(age(myCar2));
```

FUNCTION-BASED

```
let myCar1 = {
    name: "Ford",
    year: 2014
};

let myCar2 = {
    name:"Audi",
    year: 2019
};

function age(car){
    let date = new Date();
    return date.getFullYear() - car.year;
}

console.log(age(myCar1));
console.log(age(myCar2));
```

OBJECT-ORIENTED

```
class Car {
    constructor(name, year) {
        this.name = name;
        this.year = year;
    }
    age() {
        let date = new Date();
        return date.getFullYear() - this.year;
    }
}

let myCar1 = new Car("Ford", 2014);
let myCar2 = new Car("Audi", 2019);

console.log(myCar1.age());
console.log(myCar2.age());
```

USER CLASS

```
class User {
  constructor(name) {
    this.name = name;
  }
  sayHi() {
    alert(this.name);
  }
}

// Usage:
let user = new User("John");
user.sayHi();
```

FUNCTION-BASED

```
let user = {
  name: "John"
};

function sayHi(user) {
  alert(user.name);
}

sayHi(user);
```

OBJECT-ORIENTED

```
class User {
  constructor(name) {
    this.name = name;
  }
  sayHi() {
    alert(this.name);
  }
}

// Usage:
let user = new User("John");
user.sayHi();
```

CLASSES

A class is a way to describe a generic structure and functionality of an object. It is a template for creating an object.

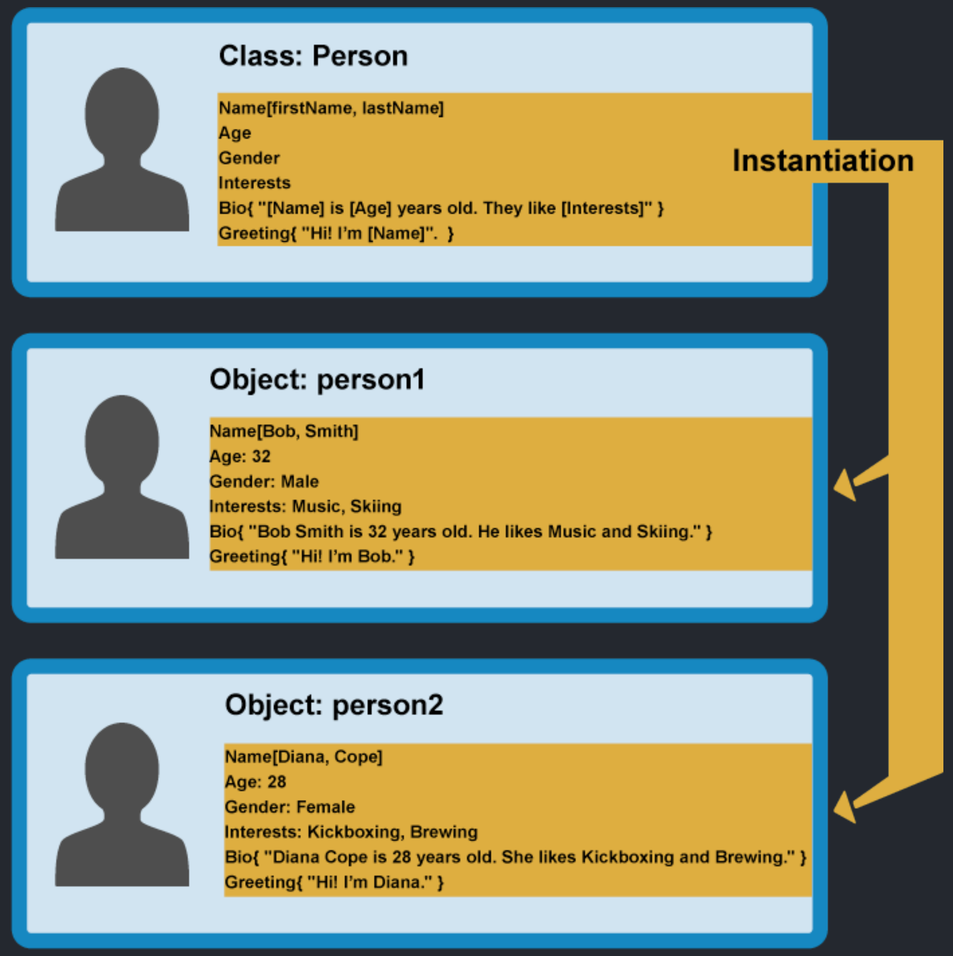
A class will define properties and functions of an object.

With a class we can declare instances (objects) with values based on the given template (class structure).

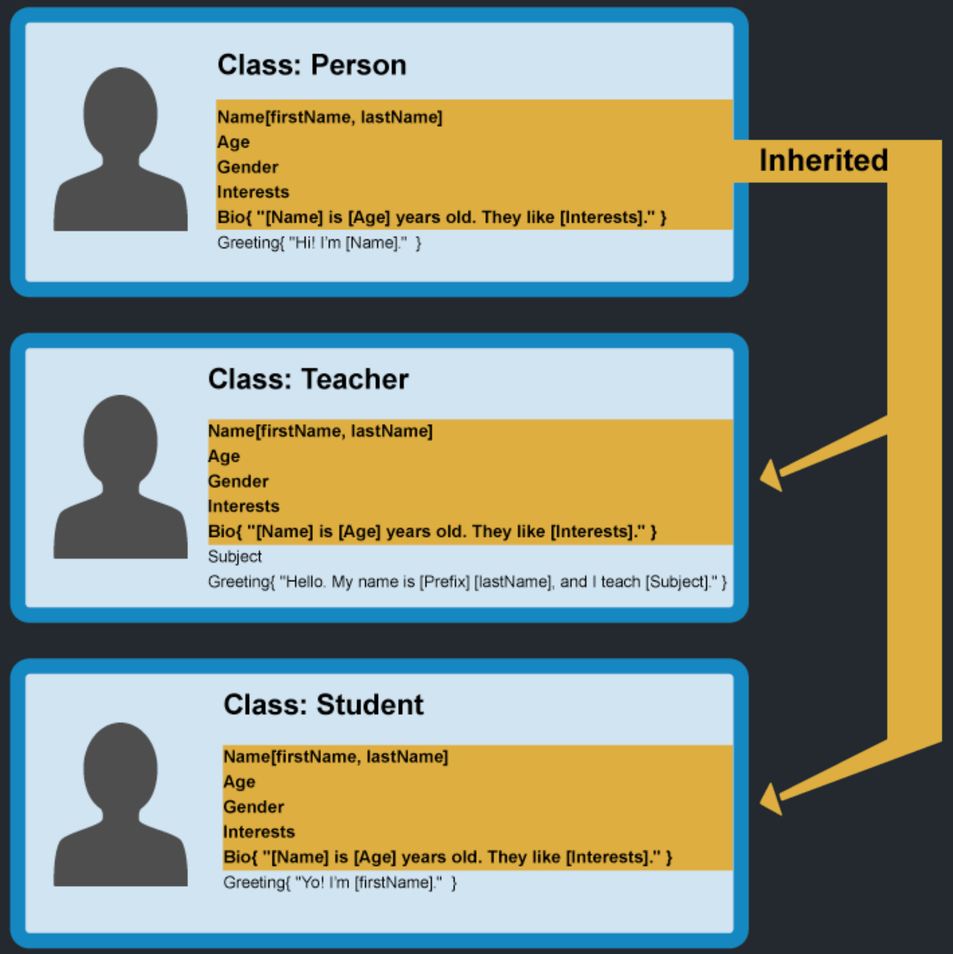
CLASSES

In practice, we often create many objects of the same kind, like users, products, movies etc.

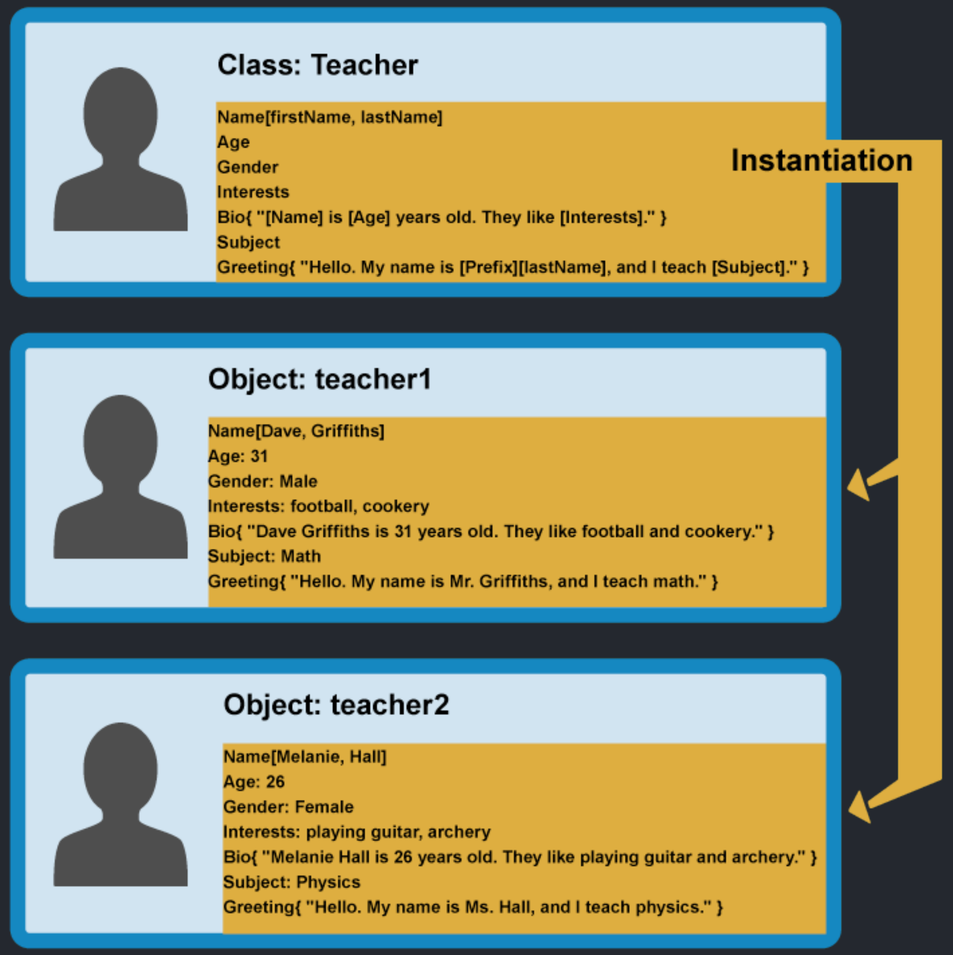
INSTANTIATION



INHERITANCE



INSTANTIATION



PERSON CLASS EXERCISE

BACK

1. Use the `project-template` and delete all inside `main.js`
2. Create a class called `Person` inside `main.js`
3. Create a `constructor`
4. Define the properties `name`, `mail`, `birthDate` & `img`
5. Define two `Person` objects based on your newly defined class.
6. Inside of `Person` class, create a function named `log()` logging all properties about the person.
7. Call `log()` on your two person objects.
8. Inside of `Person` class, create a function named `getAge()` retuning the age of the person. Calculate the age based on the property `birthDate`.
9. Call `getAge()` on your two person objects.
10. Inside of `Person` class, create a function called `getHtmlTemplate()`. The function must return a html template string. The string must include `name`, age (`getAge()`), `mail` & `img`
11. Use `getHtmlTemplate()` to add the persons to the DOM.

instantiation inside an Array

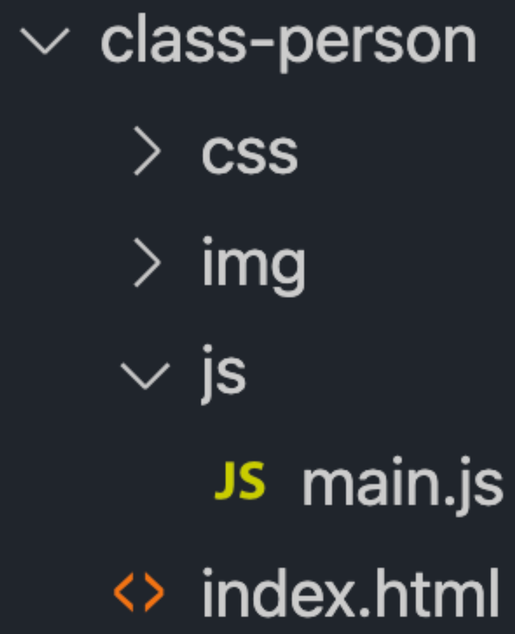
```
const persons = [
  new Person("Birgitte", "bki@mail.dk", "1966-01-14", "https://www.eaaa.dk/media/u4gorzsd/birgit
  new Person("Martin", "mnor@mail.dk", "1989-05-02", "https://media-exp1.licdn.com/dms/image/C4D
  new Person("Rasmus", "race@mail.dk", "1990-09-15", "https://www.eaaa.dk/media/devlvvgj/rasmus-
];

console.log(persons);

for (const person of persons) {
  document.querySelector("#content").innerHTML += person.getHtmlTemplate();
}
```

call method on object

PERSON CLASS EXERCISE #2

BACK

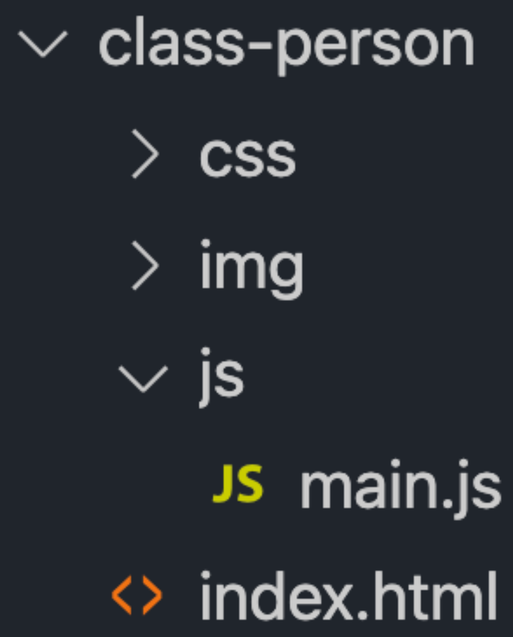
- Use your person class solution.
- Create an array with at least three instances (objects) of your person class.
- Log tha array.
- Use a for of loop to append all persons to the DOM. Make use of getHtmlTemplate to append the person.



PERSON CLASS EXERCISE #3

BACK

- Modify the person class to match the data structure of your objects in your Canvas User CRUD with JSONBIN. (properties: `id`, `name`, `avaTarUrl`, `course`, `email`, `enrollmentType`, `loginId`, & `sortableName`)
- Create some instances with your modified person class.
- Fetch the data from JSONBIN.
- Loop through the data (users array) and create instances for each object in your fetched data.
- Use getHtmlTemplate to append the users.
- Consider more useful methods/functions your could define in your Person class.



TEACHER CLASS EXERCISE

BACK

1. Use the `project-template` and delete all inside `main.js`
2. Create a class called Teacher inside main.js
3. Create a `constructor`
4. Define the properties `name`, `initials`, `department` & `img`
5. Create a function named `getMail()` using `this.initials` to define the mail.
6. Define two new `Teacher` objects.
7. Create a function named `log()` to log all information about the teacher.
8. Create a function named `sayHi()` to create an alert displaying “Hi `this.name`”.
9. Create a function named `appendPerson()` to append the teacher to the DOM.
10. Create another object (new `Teacher`).Test the class and the functions.
11. Extra: instead of defining a new (abstract) `Teacher` class, use inheritance and inherit `Teacher` class from Person class. Add the distinguishing functionality.

MODULES

A MODULE IS JUST A FILE (OR A SCRIPT).

ONE SCRIPT IS ONE MODULE.

```
// 📁 sayHi.js
export function sayHi(user) {
  alert(`Hello, ${user}!`);
}
```

A MODULE (SCRIPT)

```
// 📁 main.js
import {sayHi} from './sayHi.js';

alert(sayHi); // function...
sayHi('John'); // Hello, John!
```

ANOTHER MODULE (SCRIPT)

MODULES

As our application grows bigger, we want to split it into multiple files, so called “modules”. A module may contain a class or a library of functions for a specific purpose.

MODULES

Modules can load each other and use special directives export and import to interchange functionality, call functions of one module from another one ...

EXPORT => IMPORT

export keyword labels variables and functions that should be accessible from outside the current module.

import allows to import functionality from other modules.

```
// 📁 sayHi.js
export function sayHi(user) {
  alert(`Hello, ${user}!`);
}
```

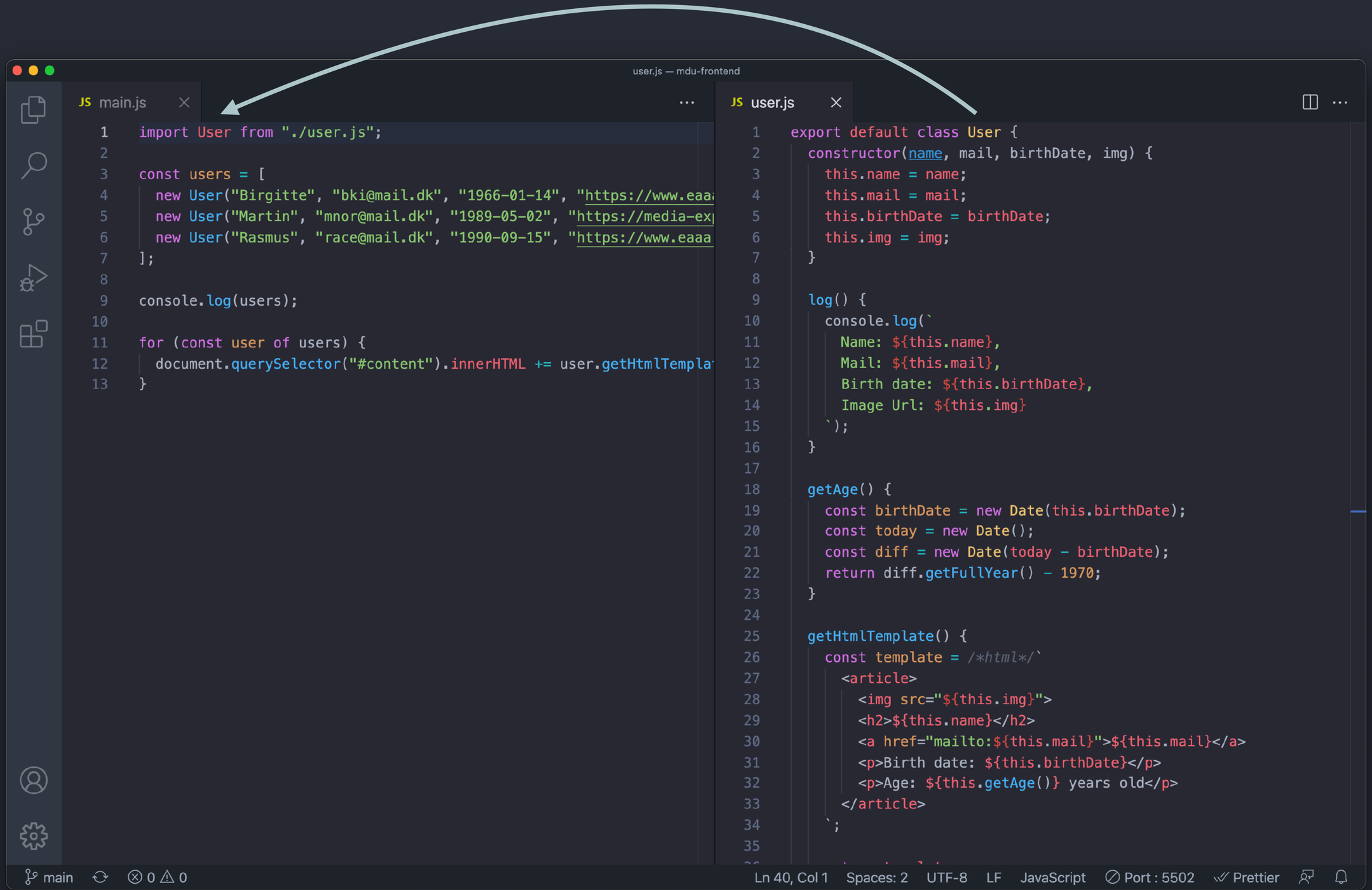
A MODULE (SCRIPT)

```
// 📁 main.js
import {sayHi} from './sayHi.js';

alert(sayHi); // function...
sayHi('John'); // Hello, John!
```

ANOTHER MODULE (SCRIPT)

```
// 📁 user.js
class User { // just add "default"
  constructor(name) {
    this.name = name;
  }
}
export default User;


// 📁 main.js
import User from './user.js'; // not {User}, just User

new User('John');
```

```
// 📁 user.js
export default class User { // just add "default"
  constructor(name) {
    this.name = name;
  }
}


// 📁 main.js
import User from './user.js'; // not {User}, just User

new User('John');
```


user.js — mdu-frontend
main.js
import User from "./user.js";
const users = [
new User("Birgitte", "bki@mail.dk", "1966-01-14", "https://www.eaa
new User("Martin", "mnor@mail.dk", "1989-05-02", "https://media-ex
new User("Rasmus", "race@mail.dk", "1990-09-15", "https://www.eaaa
];
console.log(users);
for (const user of users) {
document.querySelector("#content").innerHTML += user.getHtmlTempla
}
user.js
export default class User {
constructor(name, mail, birthDate, img) {
this.name = name;
this.mail = mail;
this.birthDate = birthDate;
this.img = img;
}
log() {
console.log(`
Name: ${this.name},
Mail: ${this.mail},
Birth date: ${this.birthDate},
Image Url: ${this.img}
`);
}
getAge() {
const birthDate = new Date(this.birthDate);
const today = new Date();
const diff = new Date(today - birthDate);
return diff.getFullYear() - 1970;
}
getHtmlTemplate() {
const template = /*html*/`
<article>
<img src="${this.img}">
<h2>${this.name}</h2>
${this.mail}
Birth date: ${this.birthDate}
Age: ${this.getAge()} years old
</article>
`;
main
0 0
Ln 40, Col 1
Spaces: 2
UTF-8
LF
JavaScript
Port : 5502
Prettier

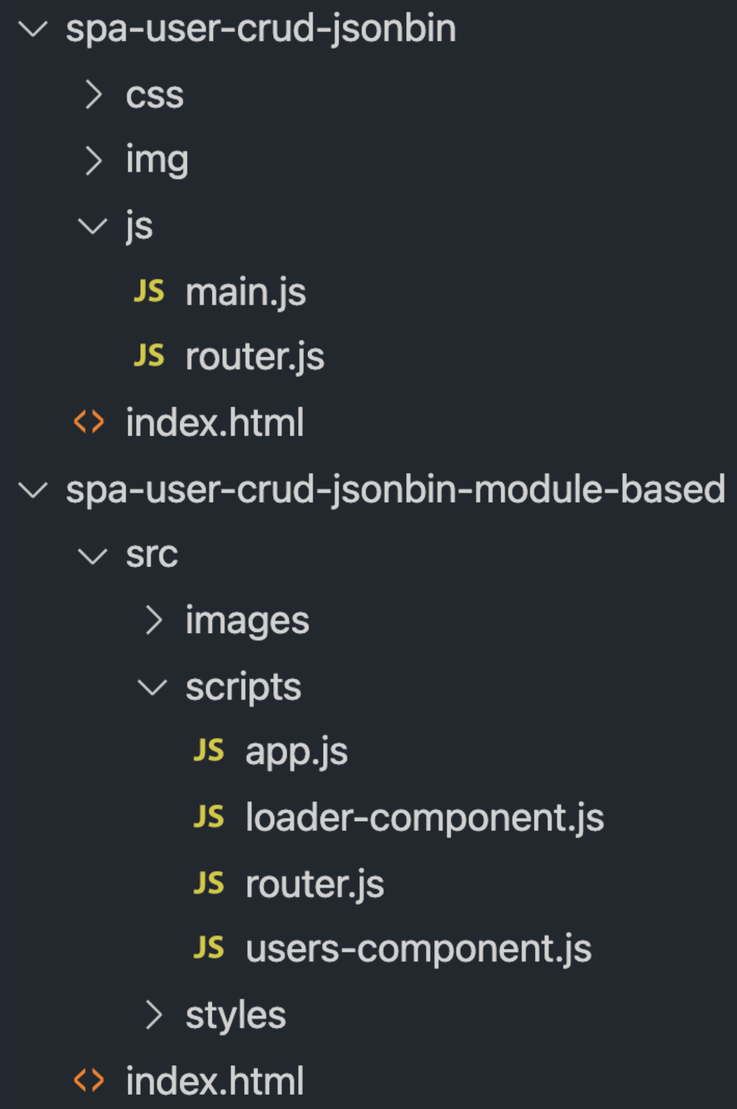
TELL THE BROWSER IT'S A MODULE

```
<script src="js/main.js" type="module"></script>
```

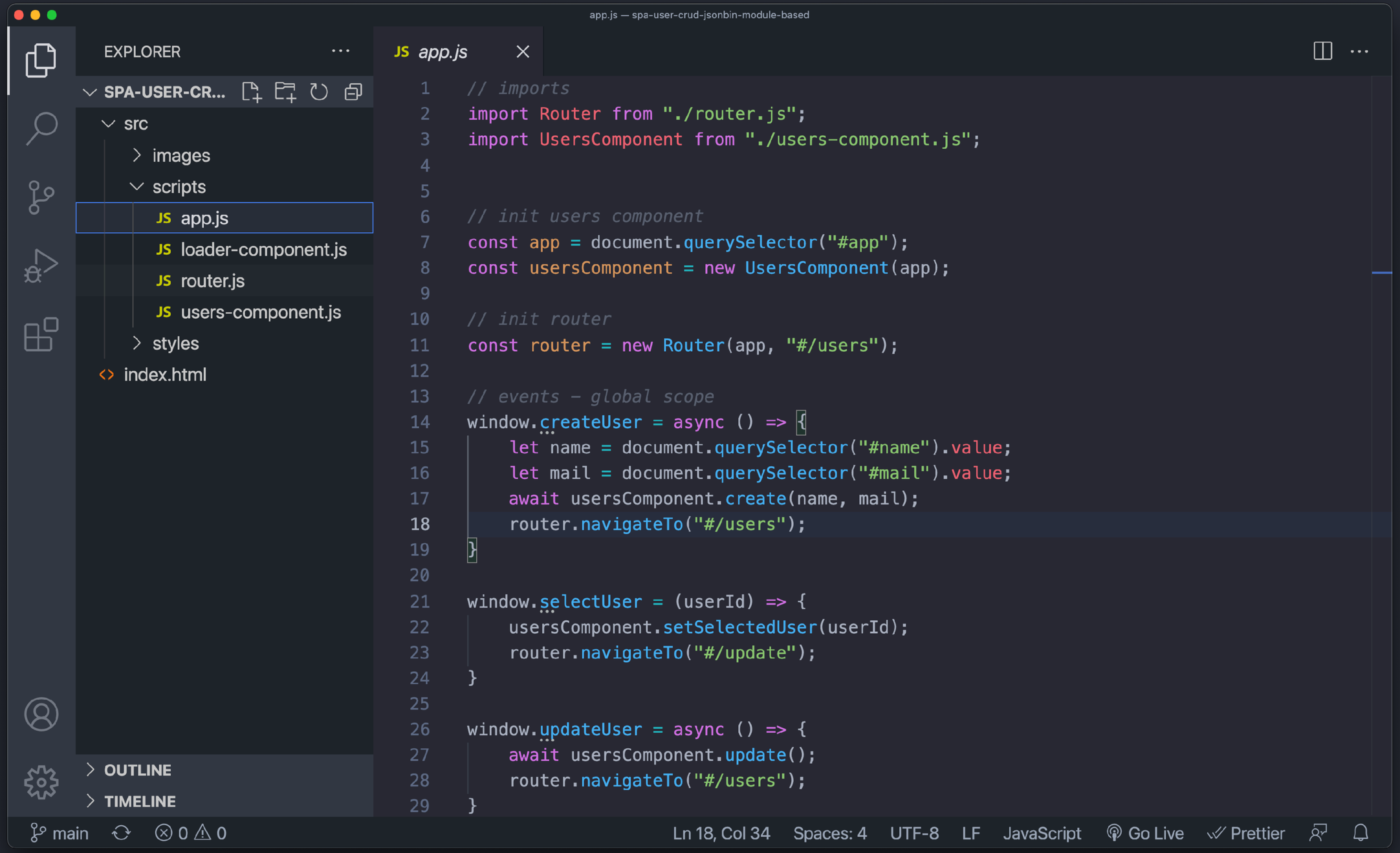
MODULES, EXPORT/IMPORT, CLASSES & OBJECT-ORIENTED JS?

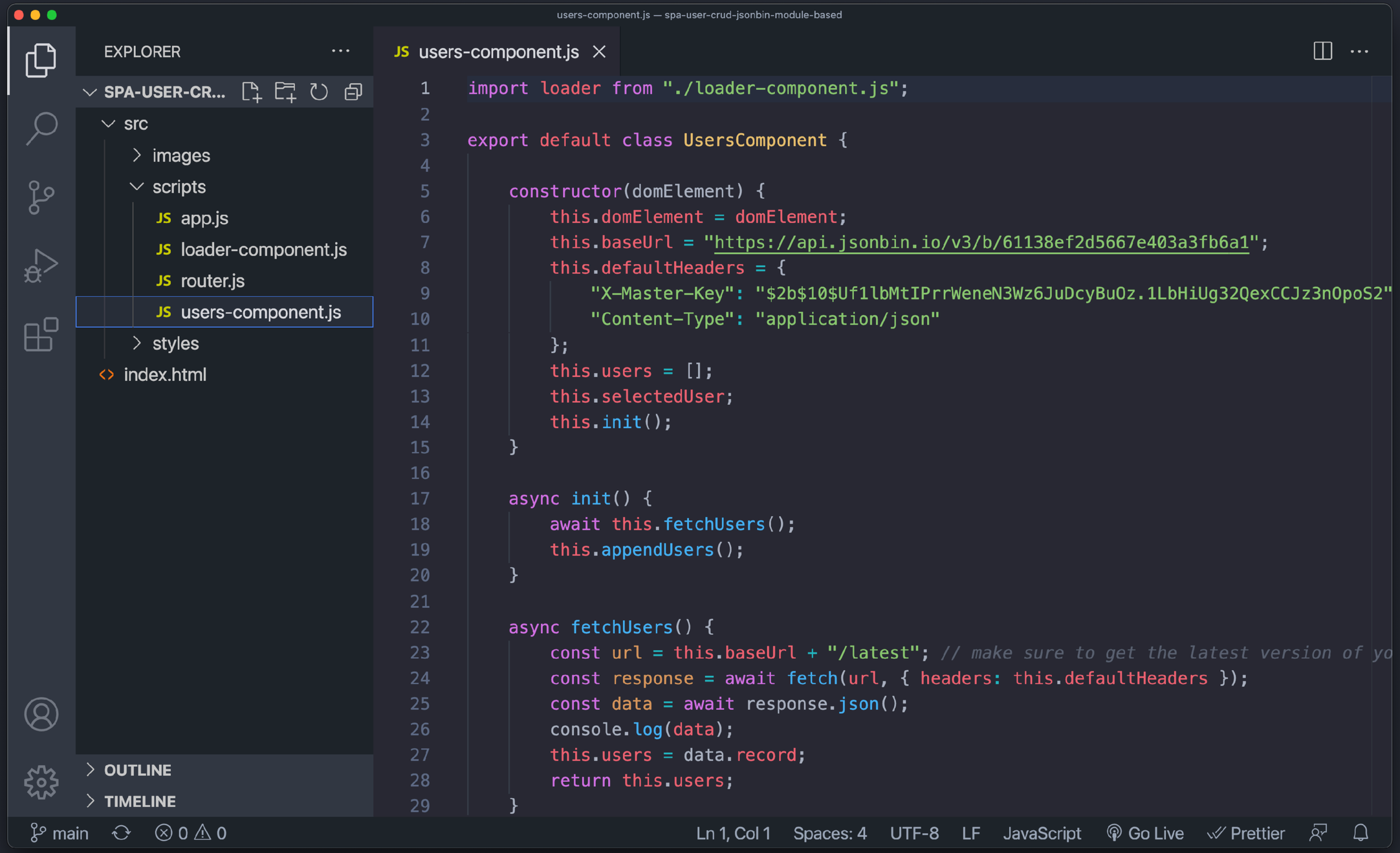
FUNCTION-BASED

OBJECT-ORIENTED

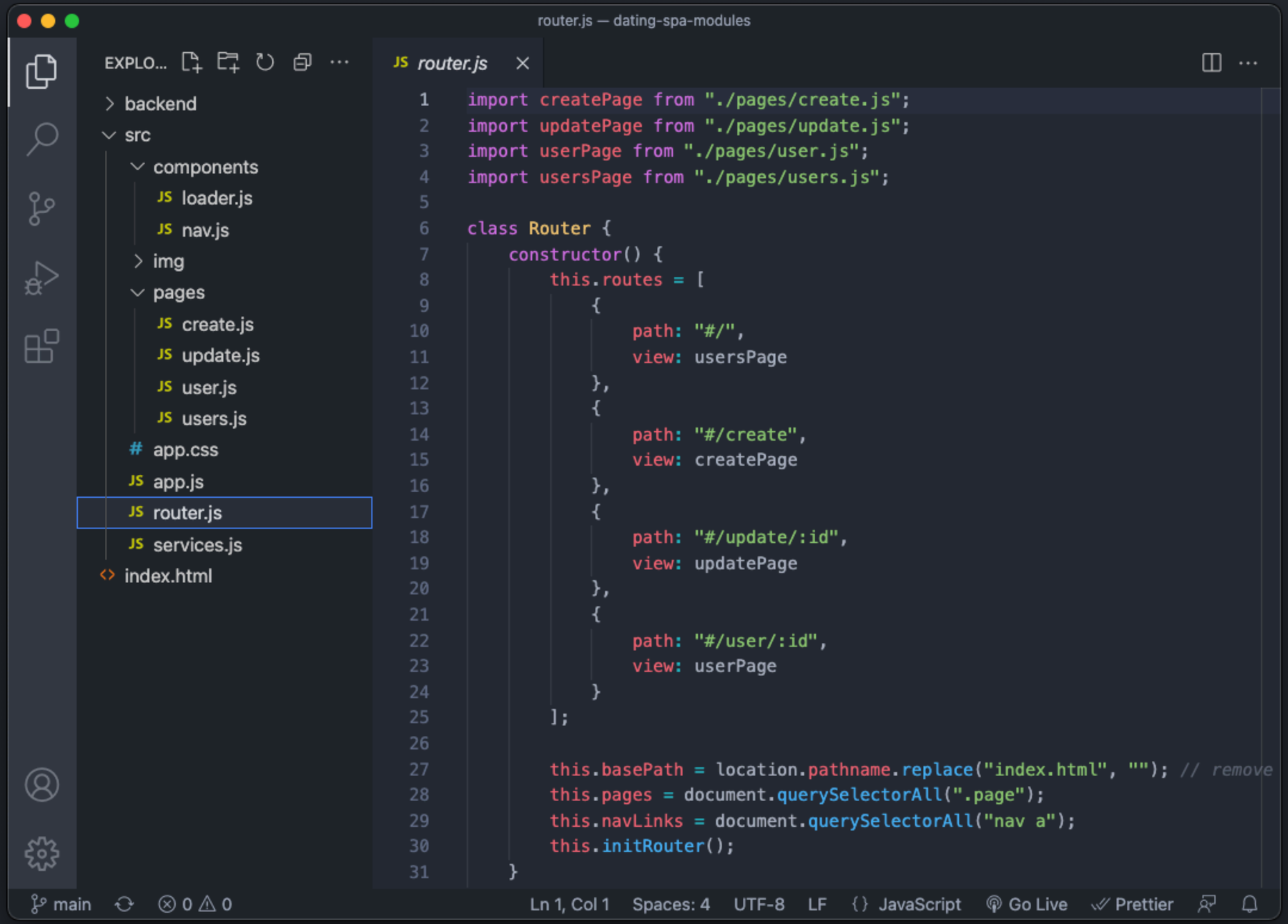



```
import loader from "./loader-component.js";

export default class UsersComponent {

    constructor(domElement) {
        this.domElement = domElement;
        this.baseUrl = "https://api.jsonbin.io/v3/b/61138ef2d5667e403a3fb6a1"
        this.defaultHeaders = {
            "X-Master-Key": "$2b$10$Uf1lbMtIPrrWeneN3Wz6JuDcyBuOz.1LbHiUg32Qe
            "Content-Type": "application/json"
        };
        this.users = [];
        this.selectedUser;
        this.init();
    }

    async init() {
        await this.fetchUsers();
        this.appendUsers();
    }

    async fetchUsers() {
        const url = this.baseUrl + "/latest"; // make sure to get the latest
        const response = await fetch(url, { headers: this.defaultHeaders });
        const data = await response.json();
        console.log(data);
        this.users = data.record;
        return this.users;
    }

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


app.js — spa-user-crud-jsonbin-module-based
EXPLORER
SPA-USER-CR...
src
images
scripts
app.js
loader-component.js
router.js
users-component.js
styles
index.html
OUTLINE
TIMELINE
app.js
// imports
import Router from "./router.js";
import UsersComponent from "./users-component.js";
// init users component
const app = document.querySelector("#app");
const usersComponent = new UsersComponent(app);
// init router
const router = new Router(app, "#/users");
// events - global scope
window.createUser = async () => {
let name = document.querySelector("#name").value;
let mail = document.querySelector("#mail").value;
await usersComponent.create(name, mail);
router.navigateTo("#/users");
}
window.selectUser = (userId) => {
usersComponent.setSelectedUser(userId);
router.navigateTo("#/update");
}
window.updateUser = async () => {
await usersComponent.update();
router.navigateTo("#/users");
}
main
0 0
Ln 18, Col 34
Spaces: 4
UTF-8
LF
JavaScript
Go Live
Prettier


users-component.js — spa-user-crud-jsonbin-module-based
EXPLORER
SPA-USER-CR...
src
images
scripts
app.js
loader-component.js
router.js
users-component.js
styles
index.html
OUTLINE
TIMELINE
users-component.js
import loader from "./loader-component.js";
export default class UsersComponent {
constructor(domElement) {
this.domElement = domElement;
this.baseUrl = "https://api.jsonbin.io/v3/b/61138ef2d5667e403a3fb6a1";
this.defaultHeaders = {
"X-Master-Key": "$2b$10$Uf1lbMtIPrrWeneN3Wz6JuDcyBuOz.1LbHiUg32QexCCJz3nOpoS2"
"Content-Type": "application/json"
};
this.users = [];
this.selectedUser;
this.init();
}
async init() {
await this.fetchUsers();
this.appendUsers();
}
async fetchUsers() {
const url = this.baseUrl + "/latest"; // make sure to get the latest version of yo
const response = await fetch(url, { headers: this.defaultHeaders });
const data = await response.json();
console.log(data);
this.users = data.record;
return this.users;
}
main
0 0
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript
Go Live
Prettier


dating-spa-jsonbin
css
img
js
main.js
router.js
index.html
dating-spa-modules
backend
src
components
loader.js
nav.js
img
pages
create.js
update.js
user.js
users.js
app.css
app.js
router.js
services.js
index.html


router.js — dating-spa-modules
EXPLO...
backend
src
components
loader.js
nav.js
img
pages
create.js
update.js
user.js
users.js
app.css
app.js
router.js
services.js
index.html
router.js
import createPage from "./pages/create.js";
import updatePage from "./pages/update.js";
import userPage from "./pages/user.js";
import usersPage from "./pages/users.js";
class Router {
constructor() {
this.routes = [
{
path: "#/",
view: usersPage
},
{
path: "#/create",
view: createPage
},
{
path: "#/update/:id",
view: updatePage
},
{
path: "#/user/:id",
view: userPage
}
];
this.basePath = location.pathname.replace("index.html", ""); // remove
this.pages = document.querySelectorAll(".page");
this.navLinks = document.querySelectorAll("nav a");
this.initRouter();
}
main
0 0
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript
Go Live
Prettier

IT’S ALL ABOUT

STRUCTURE

IMPORT-EXPORT EXERCISE

BACK

1. Use the `project-template` and delete all inside `main.js`
2. Add `type="module"` to the script tag.
3. Create a module (a new file) named `user.js`
4. Inside of `user.js` create a function called `hello()` with the following body:

   ```
   console.log("Hello from user.js module");
   ```
5. Export the `hello` function, import it in `main.js` and call `hello()` from `main.js`
6. Test in browser.
7. Copy the `Person` class from previous exercise and paste it in `user.js`
8. Rename the `Person` class to `User`.
9. Export the `User` class using `export default`
10. Import the `User` class in `main.js`, create several instances (objects) and test the `User` class.

TODO LIST COMPONENT EXERCISE

BACK

1. Use your knowlege about modules, import, export and classes to create a todo list component. Use the `project-template` and delete all inside `main.js`
2. Review the project `todo-app` and implement the same functionality with an object-oriented structure.
3. Create a new module and a `TodoList` class handling all the Todo list functionality.
4. Remember to make it reusable.

USER LIST COMPONENT EXERCISE

BACK

1. Use the `project-template` and delete all inside `main.js`
2. Use your knowlege about modules, import, export and classes to create a user list component.
3. Create a new class called `UserList` inside a new module.
4. Inside of `UserList`, define a constructor and implement the following functions: `init()`, `render()`, `fetchUsers()` & `appendUsers()`
5. Use `export defaults` to export the `UserList` class.
6. Import `UserList` in `main.js` and create an instance (object) of `UserList`

OBJECT-ORIENTED CANVAS USER CRUD

BACK

1. Use your knowledge about classes, modules, import & export in order to turn your Canvas User CRUD with JSONBIN into an object-oriented solution.
2. Use `spa-user-crud-jsonbin-module-based` as your inspiration.
3. Make a copy of your Canvas User CRUD project and rename to `spa-canvas-users-object-oriented`. In your `js` folder, create the following scripts: `app.js`, `loader-component.js`, `router.js` & `users-component.js`.
4. Start implementing the different script using an object-oriented approach (based on a class with a constructor, properties og methods). Remember to separate the functionality using the different scripts in a meaningful encapsulation (seperation of concerns).
5. User `export` and `import` to export the functionality from the scripts and import it in other script.
6. `app.js` must import an initialise the `Router` & `UserComponent`.
7. The `index.html` file must run the `app.js` only. Remember to use the type `module`.
8. Run and test your solution.
9. Extra: Consider an implement improvements in perspective of the object-oriented approach. Ex. map fetched users into user objects using a user class.

DATING SPA USING CLASSES & MODULES

BACK

1. Restructure your Dating SPA using modules and classes.
2. Start by creating the structure with needed folders and files.
    1. Create a folder called pages and define all your pages as an object.
3. … to be continued next Monday 🎉

NEXT UP

RECAP ON (eaaa.padlet.org/race/frontend_programming)??

MORE OBJECT ORIENTED JS, MODULES & COMPONENTS

CLI'S, BUILD TOOLS & PACKAGE MANAGERS

CODE EVERY DAY!