DATING SPA MODULE-BASED

Walkthrough

Use one of the solutions below, along with the slides to gain a deeper understanding of the Dating SPA and modules.

Solution: `parcel-dating-spa`

MODULES

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

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

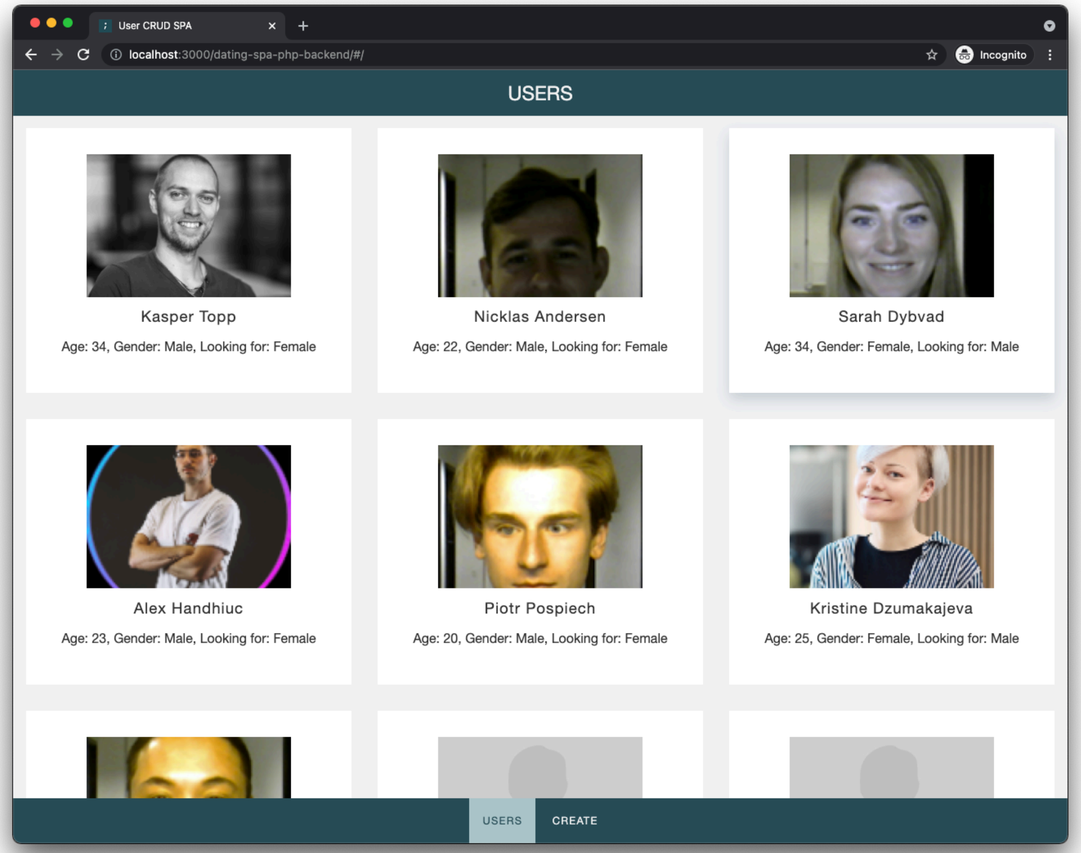
Modules can load each other and use special directives export and import to interchange functionality, call functions of one module from another one ...

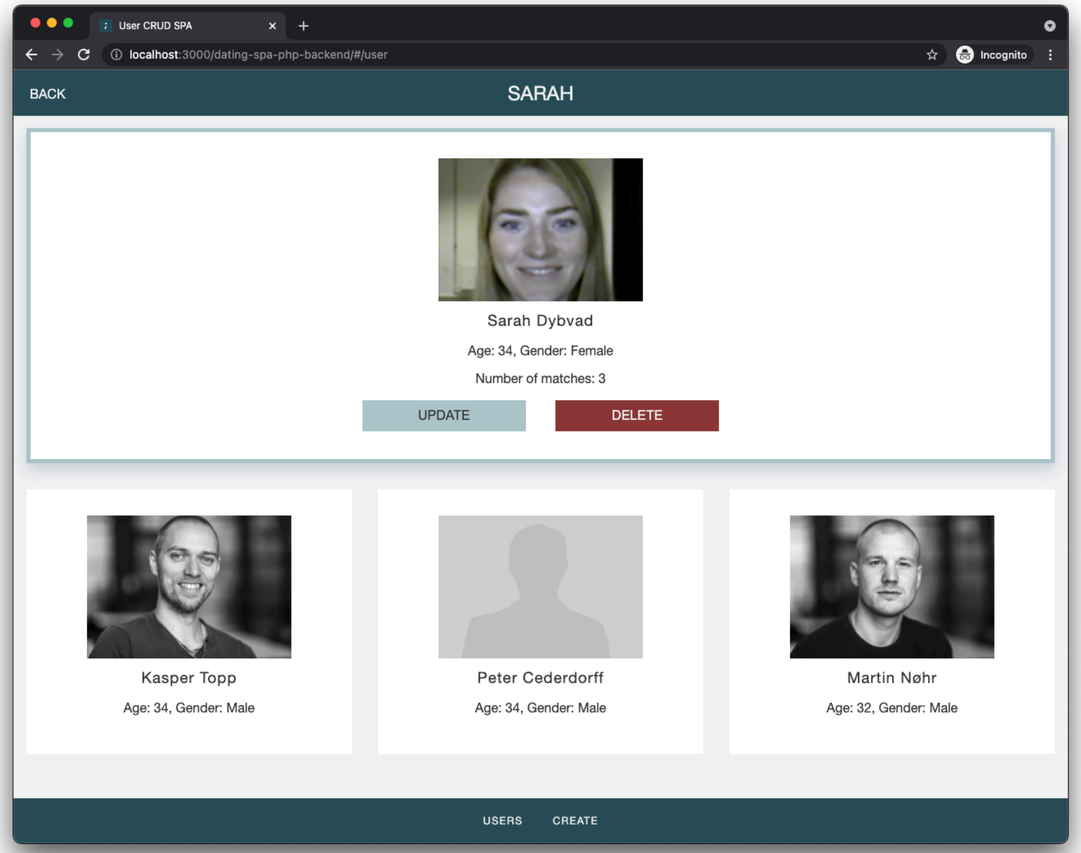
OBJECT-ORIENTED JS, MODULES & COMPONENTS

OUR CODE IS STRUCTURED IN OBJECTS & CLASSES

MODULES & COMPONENTS

DATING SPA

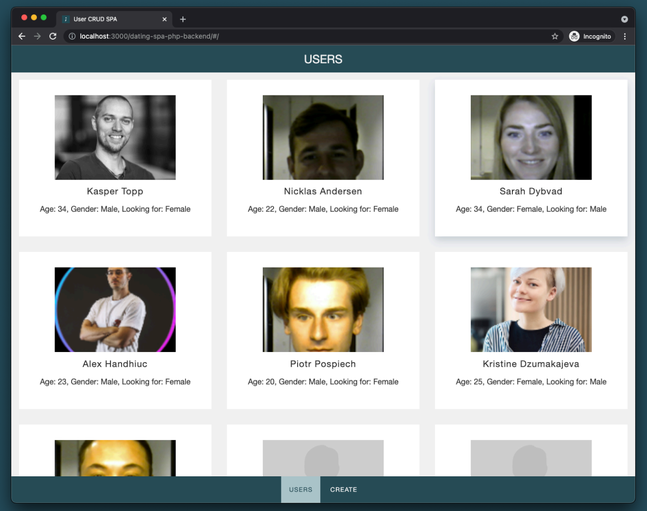




Solution: `parcel-dating-spa`

THE DATING SPA IS FETCHING DATA FROM AND TO USER SERVICE

DATING SPA



FETCH

REQUEST

JSON DATA

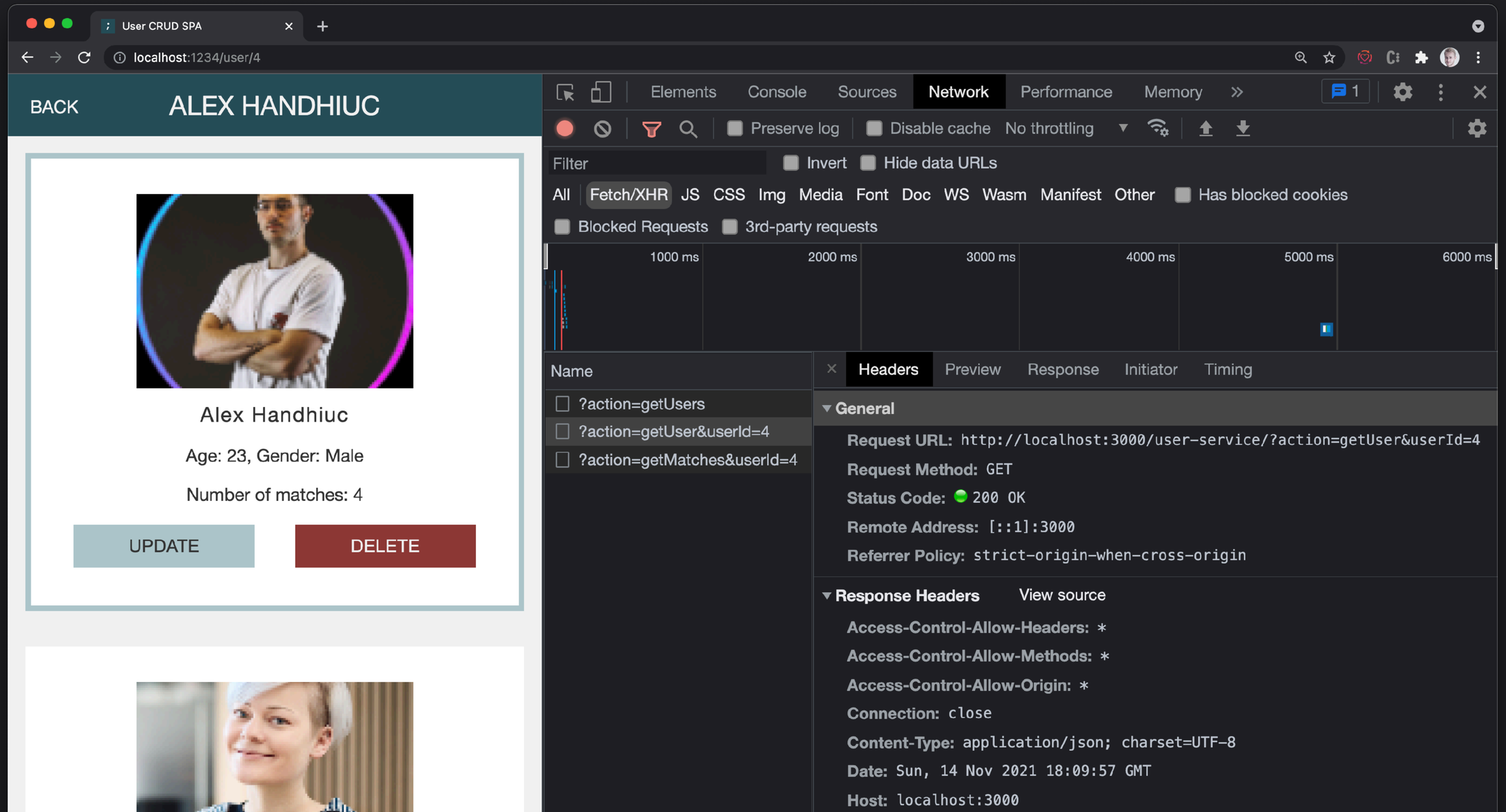
RESPONSE

USER SERVICE

`parcel-dating-spa`

`user-service` running on localhost or directly from https://web-frontend.cederdorff.com/user-service/

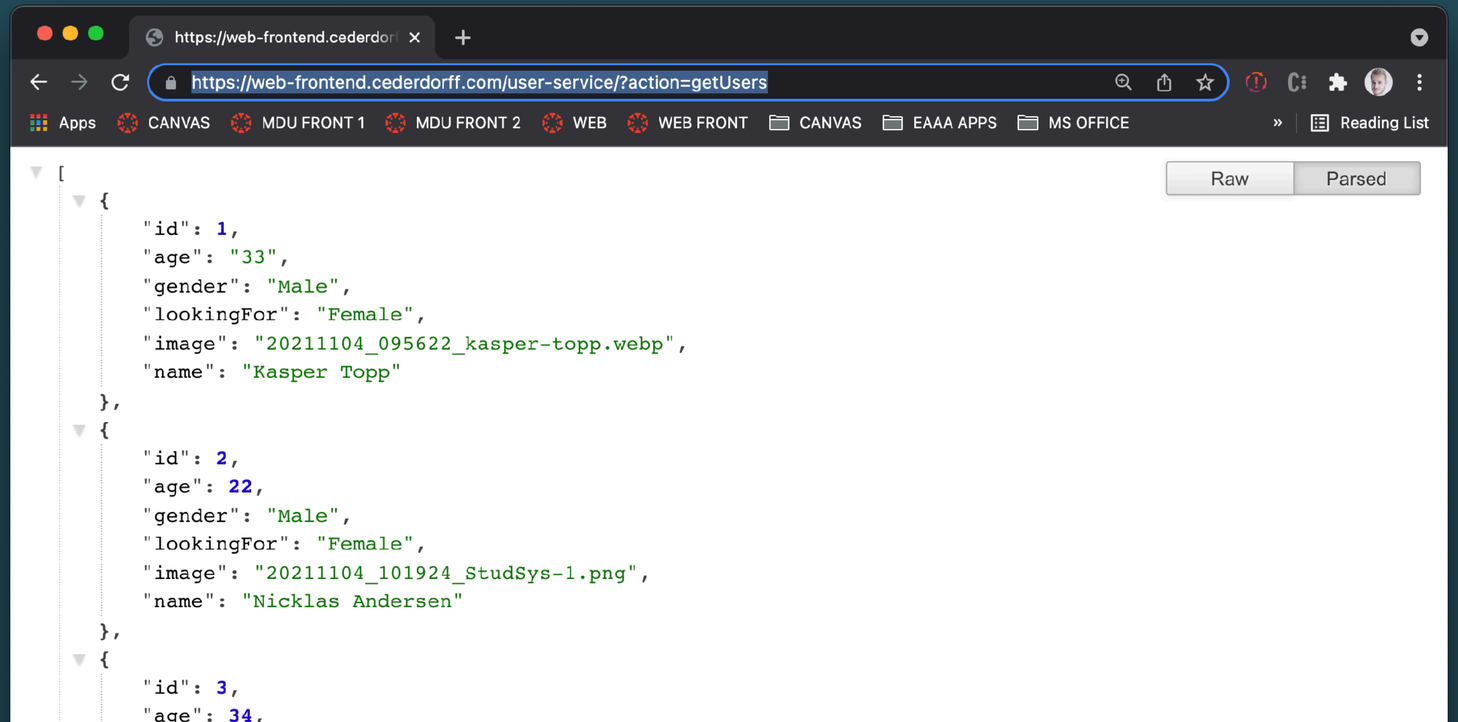
YOU CAN INSPECT FETCH CALLS IN DEV TOOL—> NETWORK



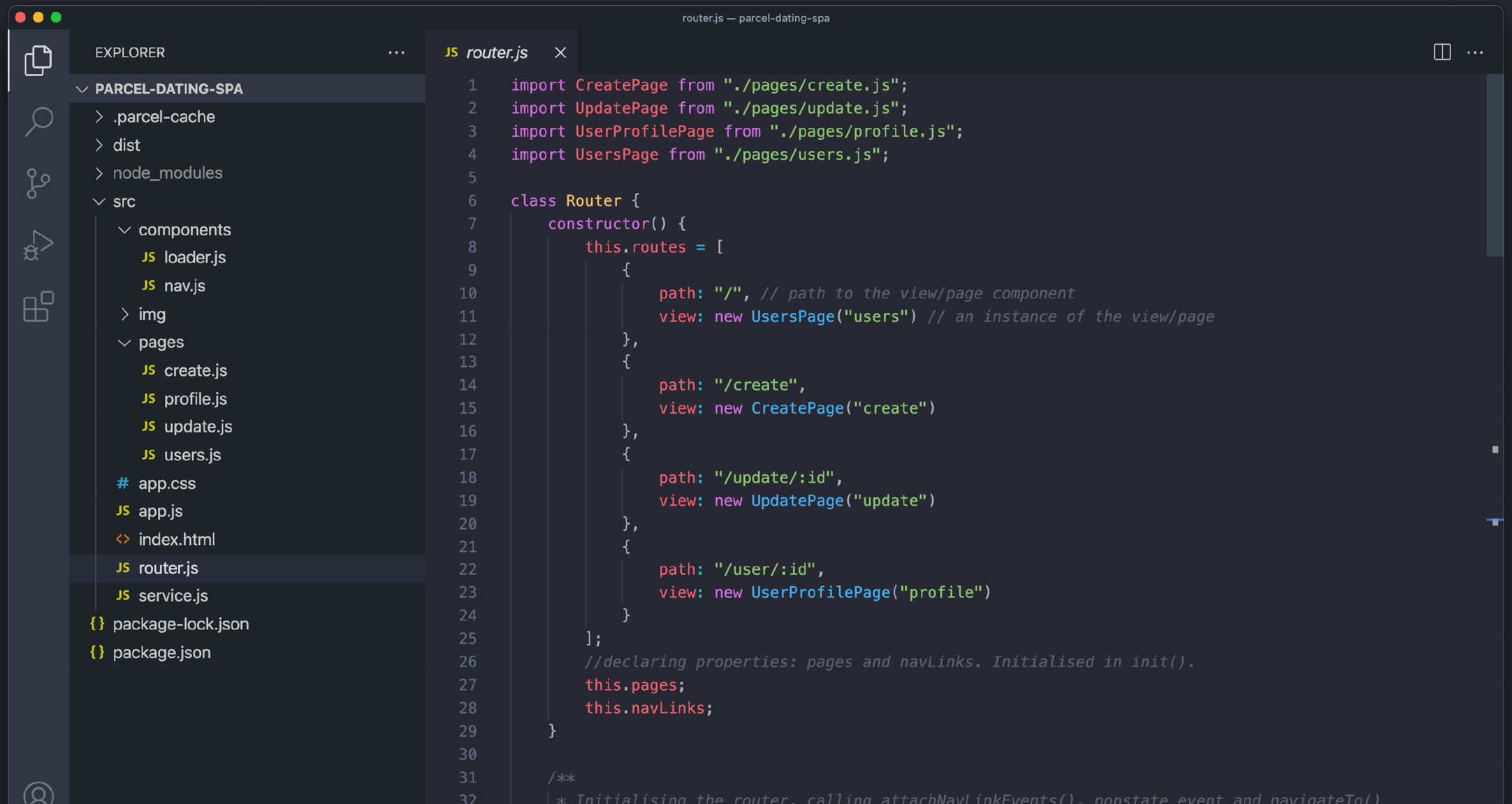
PHP USER SERVICE

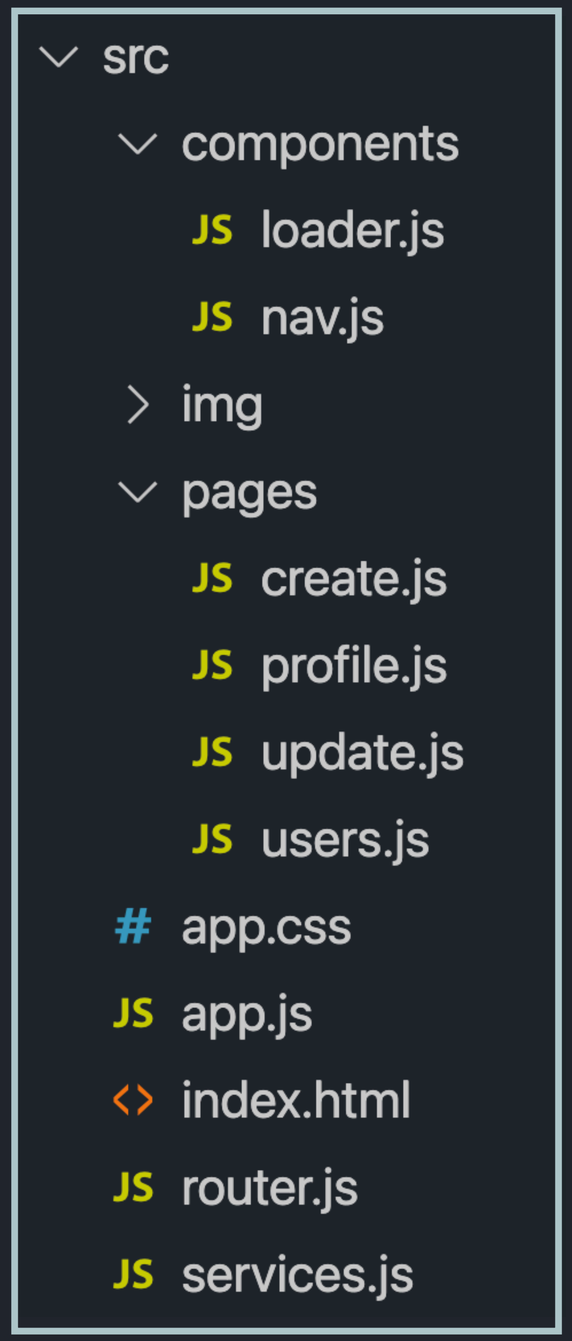
The Dating SPA is using the PHP User service. You don’t need to inspect the implementation of the User Service. All you need is a documentation, telling you how to interact with the service: https://race.notion.site/The-Docs-PHP-User-Service-55835841105f4ed79aa36c2f052a65d5

The communication between frontend and backend consists of fetch exchanging JSON data in order to create, read, update and delete data (users). Run for instance https://web-frontend.cederdorff.com/user-service/?action=getUsers in the browser:



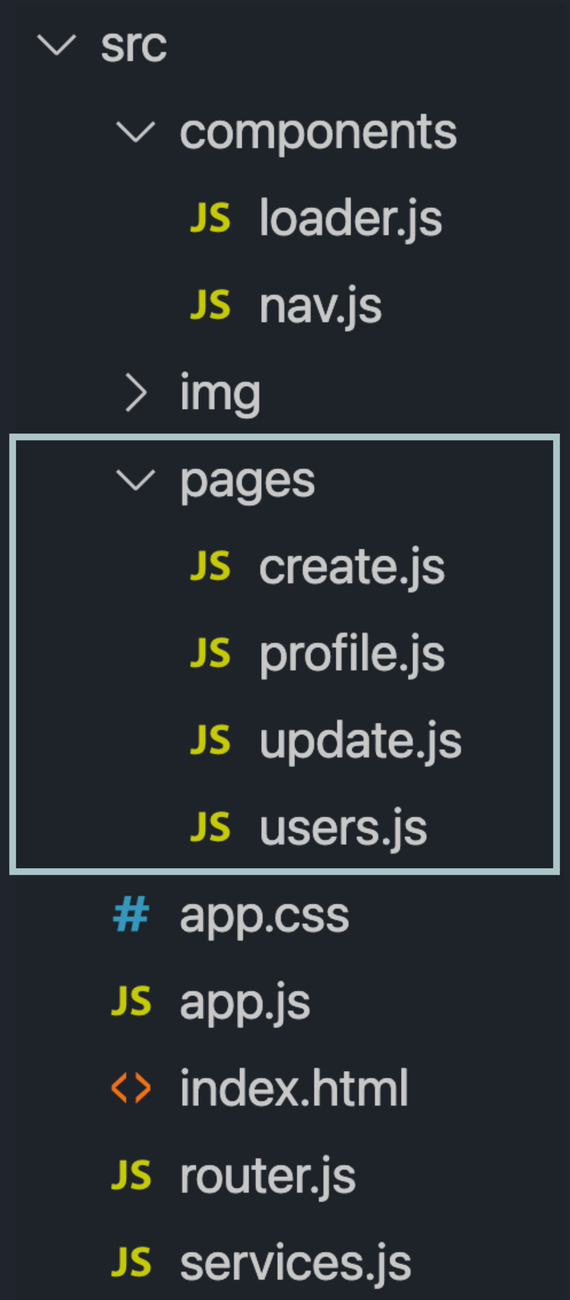
THE STRUCTURE OF THE SPA





STRUCTURE

- The Dating SPA consists of one `index.html` file and JS files.
- Some of the JS files are components or pages.
- `service.js` consists of services used across components and pages. The service communicate with the PHP backend service.
- `router.js` is a basic vanilla JS router. The router is initialising all needed functionality to make the app work like a SPA. The router imports and initialises the pages (components).
- The `index.html` is the entry file for the project. The `index.html` loads `app.css` with styles. Also `index.html` loads `app.js`
- `app.js` imports and renders the navbar component and imports and initialising the router.



PAGES

- The JS files inside of the folder pages determinate the structure and functionality of every page (users page, create page, profile page and update page).
- All pages are defined as classes and must have the functions `render()`, `attachEvents()` and `beforeShow()` (see next slide).
- The individual pages can have other functions, properties and functionality.
- I have added comments in the solution, explaining the functions, etc. Read it carefully.

THE STRUCTURE OF A PAGE

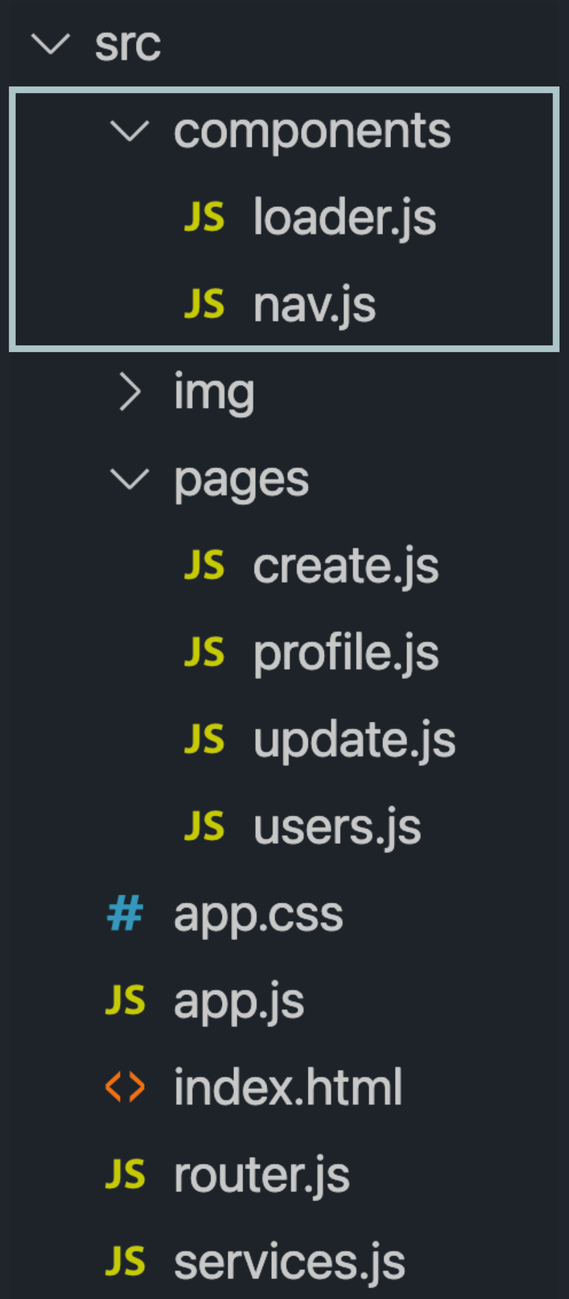
```
export default class Page {
    constructor(id) {…
    }

    render() {…
    }

    attachEvents() {…
    }

    beforeShow(props) {…
    }
}
```

Initialising page with need properties. Required: `id`

Render the initial page template

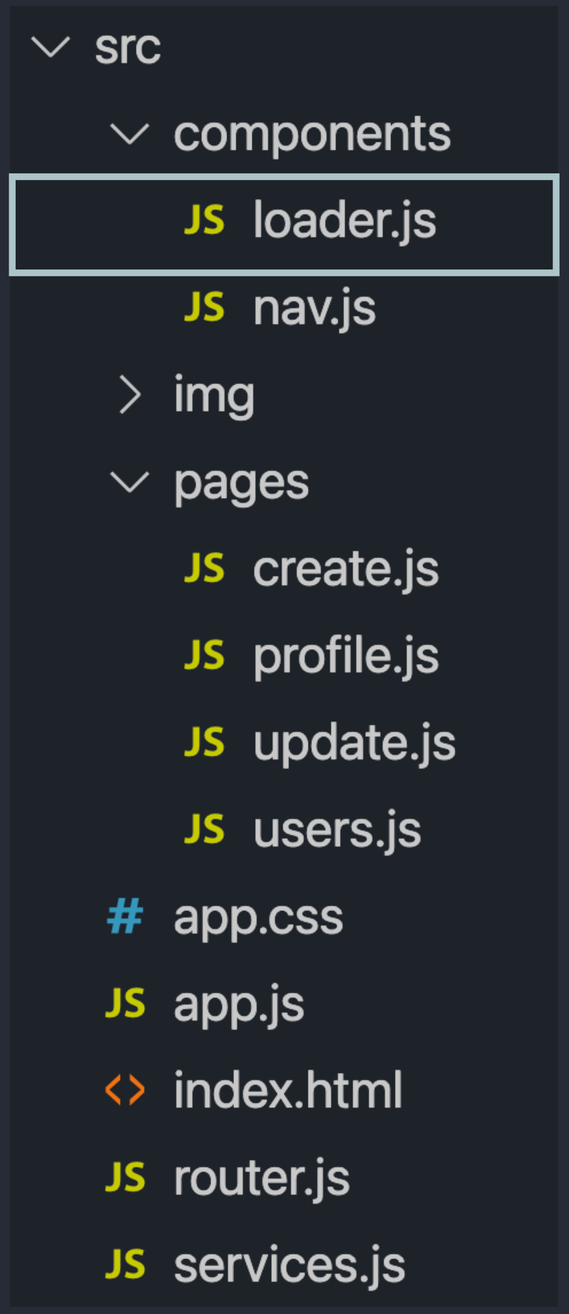
Attach events need for the page

Execute functions we need before the page is displayed



COMPONENTS

- The JS files inside of the folder components consists of components you can use (`import`) across other components.
- Pages are also components but placed in a separate folder (to make a better and more readable structure).
- In this case the `nav.js` component is imported and used by `app.js` to display the navbar.
- The `loader.js` is imported and used across different page components to show and hide the loader.

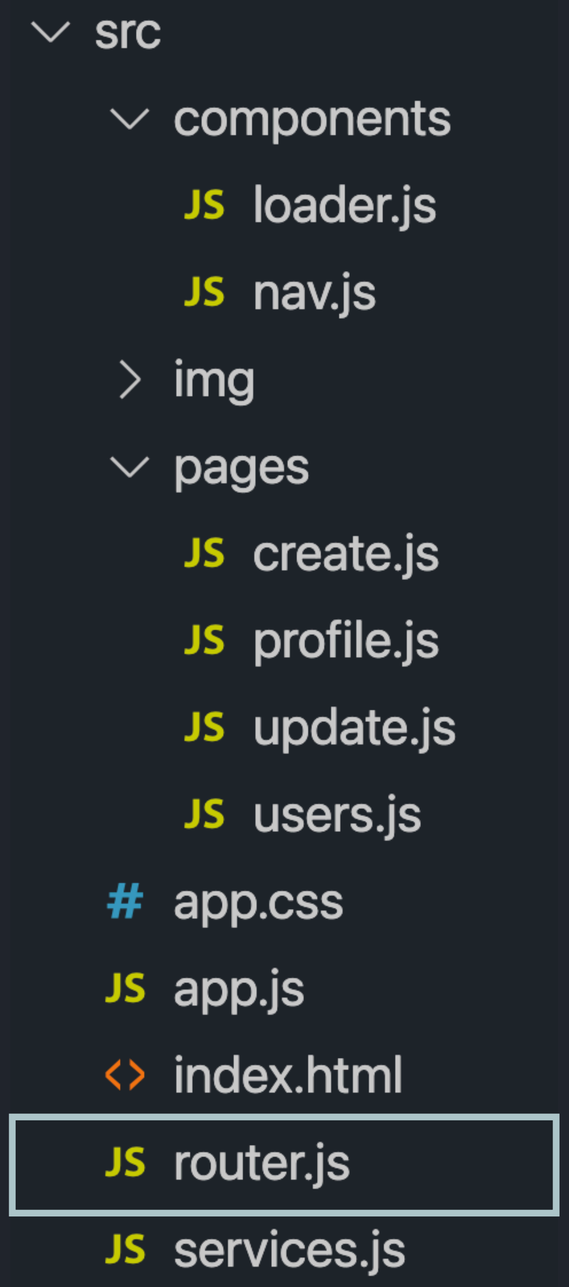


LOADER.JS

BACK

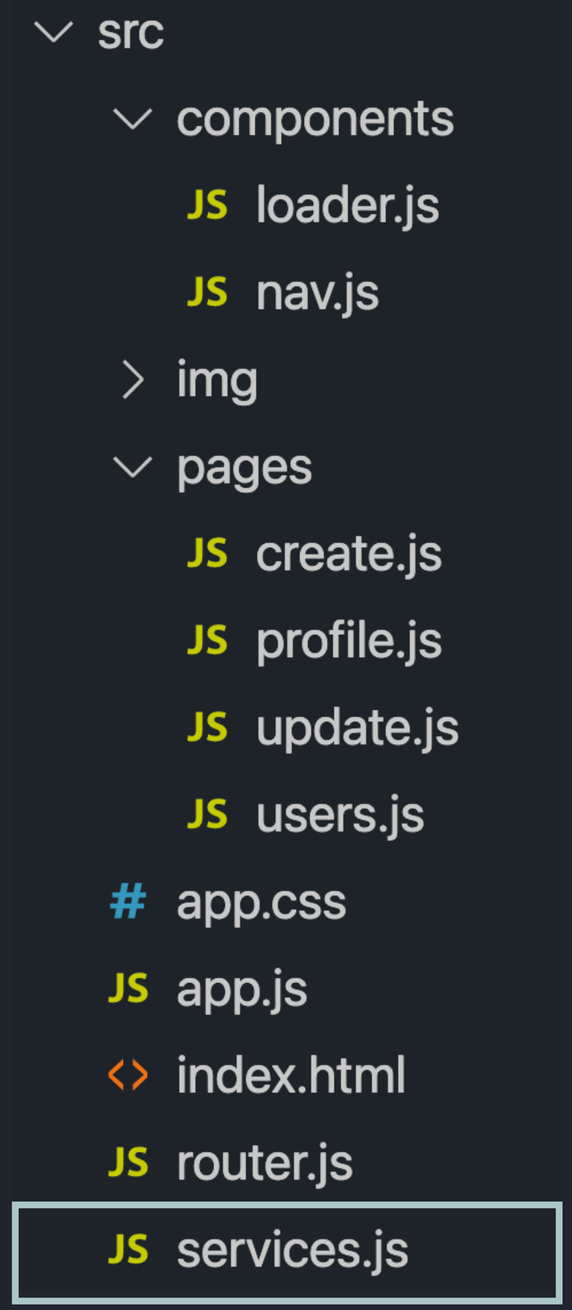
```
class Loader {
    constructor() {
        this.render();
        this.loader = document.querySelector(".loader");
    }

    render() {
        document.querySelector("#root").insertAdjacentHTML(
            "beforeend",
            /*html*/ `
            <section class="loader">
                <section class="spinner"></section>
            </section>
            `
        );
    }

    show() {
        this.loader.classList.remove("hide");
    }

    hide() {
        this.loader.classList.add("hide");
    }
}

const loader = new Loader();
export default loader;
```

- The `loader` is yet another component and can be imported by other components (pages).
- In the end of the `loader` module a new instance of the `Loader` class is declared and exported.



ROUTER

- The router is initialised by `app.js`
- In the constructor, `this.routes` is declared as an array containing route objects. Every route object consists of the property `path` and `view`.
  - `path` is the path of the page/view you would like to display.
  - In the `view` property the exact view/ page components is stored.
- You don't have to understand every single line of the router script. What you need is a basic understanding of what a router is
- In addition to the DATING SPA you need an understanding of `navigateTo(…)` and `showPage(…)`
  - Read the comment inside of the `router.js` script.
  - With `router.navigateTo(…)`, `props` and `page.beforeShow(…)` we are able to pass properties from one component (page) to another. This approach is based on Components and Props from React which we are going to work a lot with next semester 🎉



SERVICES

- The `Service Class` contains all needed functions and services used across all components (pages) in the SPA.
- Make sure to point to the backend service (`this.baseUrl = "http://localhost:3000/user-service/"` or `"https://web-frontend.cederdorff.com/user-service/"`).
- The service class defines all need functions in order to interact with the backend service.
- The functions consists of a fetch call with different request methods (`GET`, `POST`, `PUT`, `DELETE`).
- The service could have other useful methods used across components. It is one way to make the code more reusable and readable.
- I've added comments in `service.js` as well 😊

CODE EVERY DAY!