

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

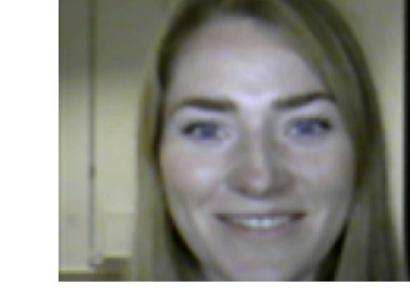
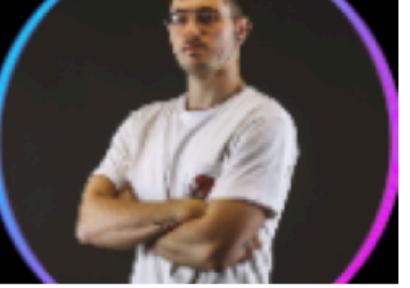
OUR CODE IS STRUCTURED IN OBJECTS &
CLASSES
MODULES & COMPONENTS

DATING SPA

User CRUD SPA

localhost:3000/dating-spa-php-backend/#/

USERS

 Kasper Topp Age: 34, Gender: Male, Looking for: Female	 Nicklas Andersen Age: 22, Gender: Male, Looking for: Female	 Sarah Dybvad Age: 34, Gender: Female, Looking for: Male
 Alex Handhiuc Age: 23, Gender: Male, Looking for: Female	 Piotr Pospiech Age: 20, Gender: Male, Looking for: Female	 Kristine Dzumakajeva Age: 25, Gender: Female, Looking for: Male
		

USERS CREATE

User CRUD SPA

localhost:3000/dating-spa-php-backend/#/user

SARAH

BACK


Sarah Dybvad
Age: 34, Gender: Female
Number of matches: 3

UPDATE DELETE

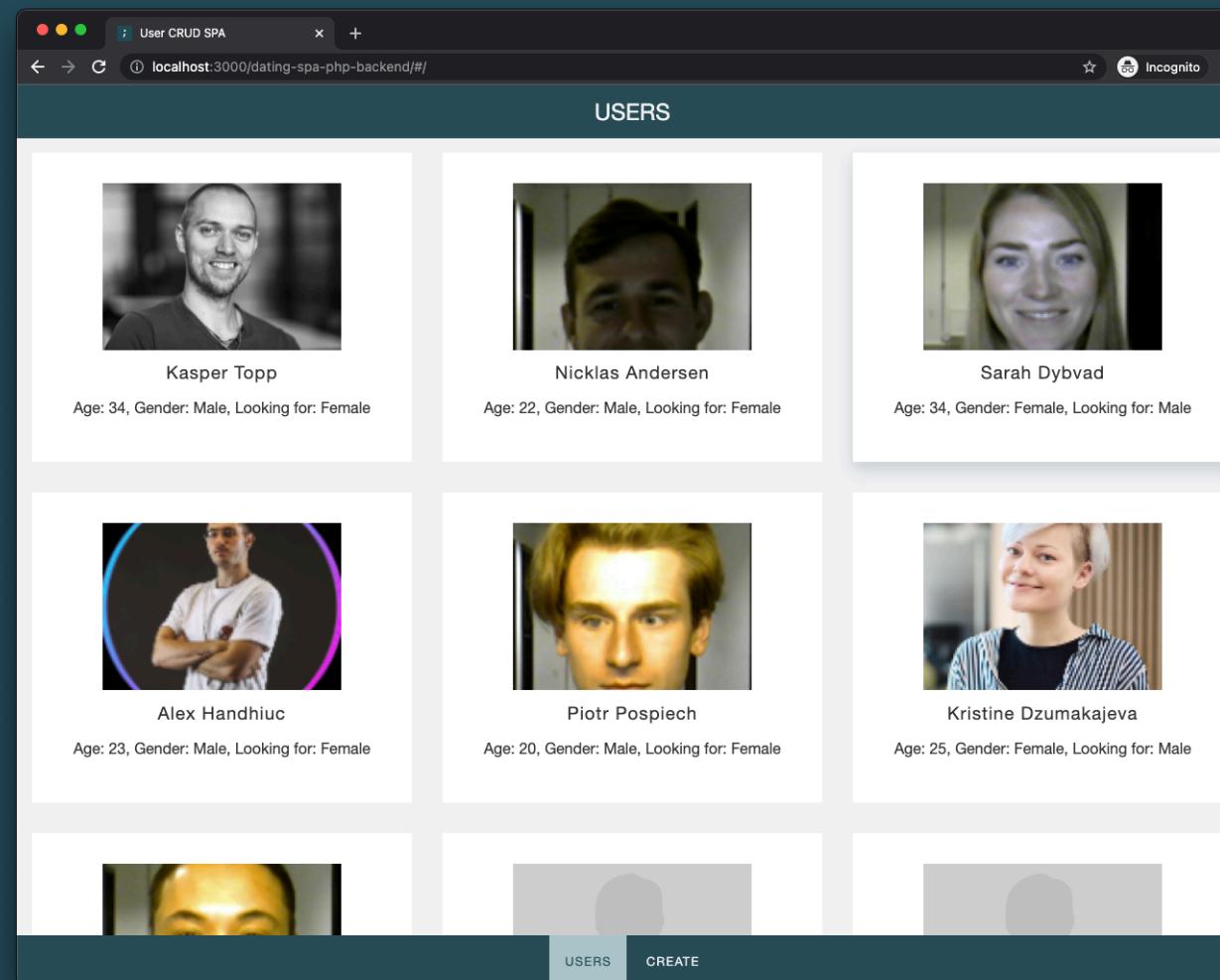
 Kasper Topp Age: 34, Gender: Male	 Peter Cederdorff Age: 34, Gender: Male	 Martin Nøhr Age: 32, Gender: Male
---	--	---

USERS CREATE

Solution: parcel-dating-spa

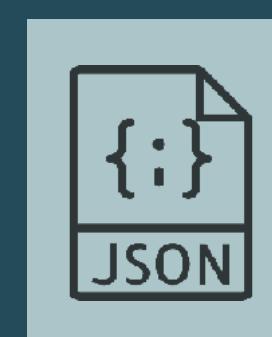
THE DATING SPA IS FETCHING DATA FROM AND TO USER SERVICE

DATING SPA



FETCH
REQUEST

JSON DATA
RESPONSE



USER SERVICE

A screenshot of a browser window showing the network tab with a request to "localhost:3000/user-service/?action=getUsers". The response is a JSON array containing five user objects. Each object has properties: id, age, gender, lookingFor, image, and name. The data is as follows:

```
[{"id": 1, "age": "35", "gender": "Male", "lookingFor": "Female", "image": "20211104_095622_kasper-topp.webp", "name": "Kasper Topp"}, {"id": 2, "age": "22", "gender": "Male", "lookingFor": "Female", "image": "20211104_101924_StudSys-1.png", "name": "Nicklas Andersen"}, {"id": 3, "age": "34", "gender": "Female", "lookingFor": "Male", "image": "20211104_095710_StudSys.png", "name": "Sarah Dybvad"}, {"id": 4, "age": "23", "gender": "Male", "lookingFor": "Female", "image": "20211104_102429_8fc636b-76d9-4f56-8f18-3d6faaf5859e.png", "name": "Alex Handhiuc"}, {"id": 5, "age": "25", "gender": "Female", "lookingFor": "Male", "image": "20211104_102429_8fc636b-76d9-4f56-8f18-3d6faaf5859e.png", "name": "Kristine Dzumakajeva"}]
```

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

The screenshot shows a User CRUD SPA application running at `localhost:1234/user/4`. The main view displays a user profile for "ALEX HANDHIUC" with a circular photo, name, age (23), gender (Male), and 4 matches. Below the photo are "UPDATE" and "DELETE" buttons. In the bottom left corner, there's a small thumbnail of another user's profile picture.

The right side of the screenshot shows the Chrome DevTools Network tab. The "Fetch/XHR" tab is selected, showing a list of requests. One request is highlighted: `?action=getUser&userId=4`. The details panel on the right shows the following information:

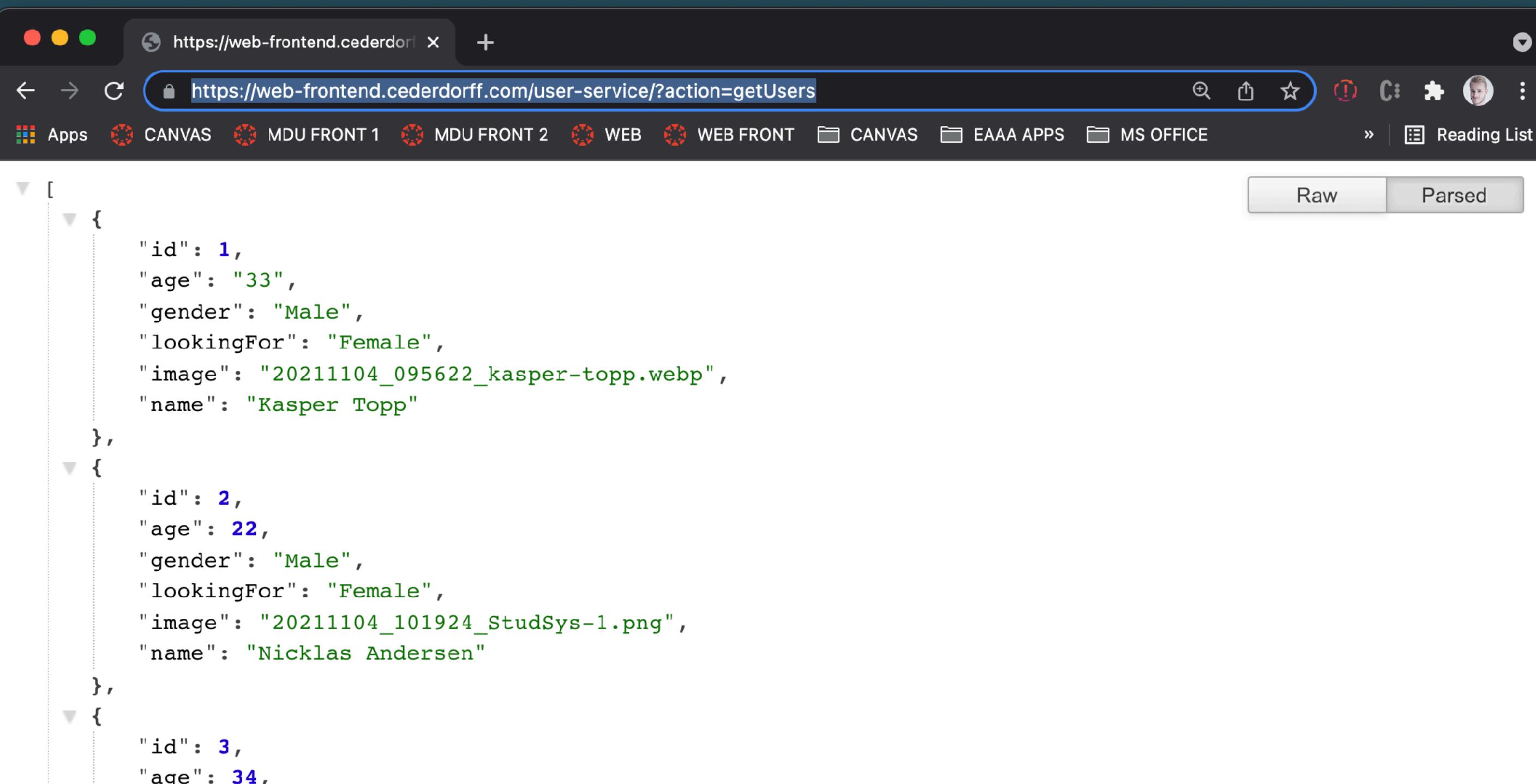
- General**
 - Request URL: `http://localhost:3000/user-service/?action=getUser&userId=4`
 - Request Method: GET
 - Status Code: 200 OK
 - Remote Address: `[::1]:3000`
 - Referrer Policy: strict-origin-when-cross-origin
- Response Headers**
 - Access-Control-Allow-Headers: *
 - Access-Control-Allow-Methods: *
 - Access-Control-Allow-Origin: *
 - Connection: close
 - Content-Type: application/json; charset=UTF-8
 - Date: Sun, 14 Nov 2021 18:09:57 GMT
 - Host: localhost:3000

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>

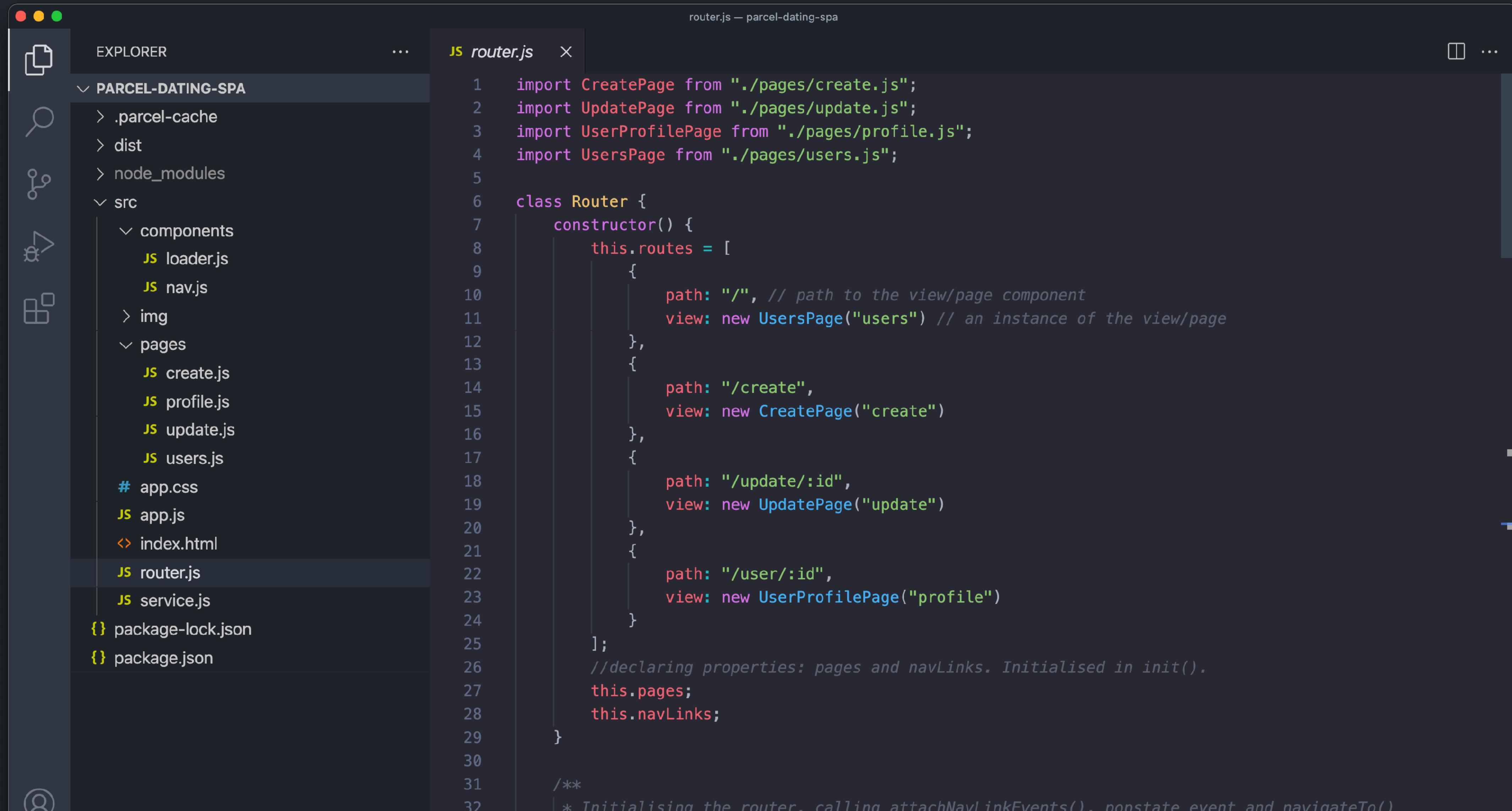
The communication between frontend and backend consists of fetch exchanging JSON data in order to create, read, update and delete data (users). Run for instance <https://web-frontend.cederdorff.com/user-service/?action=getUsers> in the browser:



A screenshot of a Mac OS X desktop environment showing a web browser window. The browser's title bar says "https://web-frontend.cederdorff.com". The address bar also displays the same URL. Below the address bar is a menu bar with various icons. The main content area of the browser shows a JSON array of user objects. Each object has properties: id, age, gender, lookingFor, image, and name. The first user object has an id of 1, age of 33, male gender, female lookingFor, an image named "20211104_095622_kasper-topp.webp", and a name of "Kasper Topp". The second user object has an id of 2, age of 22, male gender, female lookingFor, an image named "20211104_101924_StudSys-1.png", and a name of "Nicklas Andersen". The third user object has an id of 3 and an age of 34.

```
[{"id": 1, "age": "33", "gender": "Male", "lookingFor": "Female", "image": "20211104_095622_kasper-topp.webp", "name": "Kasper Topp"}, {"id": 2, "age": "22", "gender": "Male", "lookingFor": "Female", "image": "20211104_101924_StudSys-1.png", "name": "Nicklas Andersen"}, {"id": 3, "age": "34", "gender": "Male", "lookingFor": "Female", "image": "20211104_101924_StudSys-1.png", "name": "Andrea"}]
```

THE STRUCTURE OF THE SPA



The screenshot shows a dark-themed instance of Visual Studio Code. On the left is the Explorer sidebar, which lists the project structure of 'PARCEL-DATING-SPA'. It includes a '.parcel-cache' folder, a 'dist' folder, a 'node_modules' folder, and a 'src' folder. Inside 'src', there are 'components', 'img', and 'pages' folders. The 'pages' folder contains four files: 'create.js', 'profile.js', 'update.js', and 'users.js'. Below these are '# app.css', 'app.js', 'index.html', 'router.js' (which is currently open in the editor), and 'service.js'. At the bottom of the sidebar are 'package-lock.json' and 'package.json'. The main editor area displays the 'router.js' file with the following content:

```
router.js — parcel-dating-spa
JS router.js X
1 import CreatePage from "./pages/create.js";
2 import UpdatePage from "./pages/update.js";
3 import UserProfilePage from "./pages/profile.js";
4 import UsersPage from "./pages/users.js";
5
6 class Router {
7     constructor() {
8         this.routes = [
9             {
10                 path: "/", // path to the view/page component
11                 view: new UsersPage("users") // an instance of the view/page
12             },
13             {
14                 path: "/create",
15                 view: new CreatePage("create")
16             },
17             {
18                 path: "/update/:id",
19                 view: new UpdatePage("update")
20             },
21             {
22                 path: "/user/:id",
23                 view: new UserProfilePage("profile")
24             }
25         ];
26         //declaring properties: pages and navLinks. Initialised in init().
27         this.pages;
28         this.navLinks;
29     }
30
31     /**
32      * Initialising the router. calling attachNavLinkEvents(), popstate event and navigateTo()
33     */
34 }
```

```
✓ src
  ✓ components
    JS loader.js
    JS nav.js
  > img
  ✓ pages
    JS create.js
    JS profile.js
    JS update.js
    JS users.js
  # app.css
  JS app.js
  <> index.html
  JS router.js
  JS services.js
```

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.

```
✓ src
  ✓ components
    JS loader.js
    JS nav.js
  > img
  ✓ pages
    JS create.js
    JS profile.js
    JS update.js
    JS users.js
```



```
# app.css
JS app.js
<> index.html
JS router.js
JS services.js
```

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

```
✓ src
  ✓ components
    JS loader.js
    JS nav.js
  > img
  ✓ pages
    JS create.js
    JS profile.js
    JS update.js
    JS users.js
  # app.css
  JS app.js
  <> index.html
  JS router.js
  JS services.js
```

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.

```
✓ src
  ✓ components
    JS loader.js
    JS nav.js
  > img
  ✓ pages
    JS create.js
    JS profile.js
    JS update.js
    JS users.js
  # app.css
  JS app.js
  <> index.html
  JS router.js
  JS services.js
```

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.

```
✓ src
  ✓ components
    JS loader.js
    JS nav.js
  > img
  ✓ pages
    JS create.js
    JS profile.js
    JS update.js
    JS users.js
  # app.css
  JS app.js
  <> index.html
  JS router.js
  JS services.js
```

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 

```
✓ src
  ✓ components
    JS loader.js
    JS nav.js
  > img
  ✓ pages
    JS create.js
    JS profile.js
    JS update.js
    JS users.js
  # app.css
  JS app.js
  <> index.html
  JS router.js
  JS services.js
```

S E R V I C E S

- 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!