

6 JS: Async

Up to now, games are hardcoded in the js-file. When you finish this labo, games will be stored in a back-end server. Instead of saving them in a hardcoded array in the js-file, we will ask the server for the data.

6.1 Starting up the webserver

The back-end "The Games Library" is a NodeJS runtime. It allows us to run Javascript applications outside the browser. This application behaves like an external webserver as webontwerp.ucll.be. We will run the application locally. Than your computer behaves as a (local) webserver. Applications on your computer have access via <http://localhost:3000>. Other people do not have access to it.

Download The Games Library back-end (Toledo). Install NodeJS as explained in the file `readme.md`.

Once NodeJS is installed on your computer, you have to prepare the downloaded application. Open a terminal and navigate to the project root folder of the back-end. In following example, The Games Library is downloaded to "Desktop/front-end-demo/TheGamesLibrary-Backend". With "cd ..." we navigate to the project root folder.

```
~ % cd Desktop/front-end-demo/TheGamesLibrary-Backend
```

Run the following commands to install all required node packages and get the server up and running:

```
$ npm install
$ npm start
```

This will start the back-end application on localhost. You stop the application with "Ctrl-C".

You can access the API documentation and test it via Swagger running on <http://localhost:3000/>.

6.2 Fetching all games

Keep on working in "table-overview.js" and "table-overview.html".

6.2.1 Syntax

First we ask the back-end for all animals and store them in the local animals array.



```
const animals = [];  
  
const fetchAnimals = async () => {  
  const response =  
    await fetch("http://localhost:3000/animals");
```

```
const result = await response.json();
animals.push(...result); };
```

We declare an empty array of animals.

The function "fetch()" sends a get request to the back-end. We annotate the fetch with await to ensure javascript waits for the result of the fetch before moving on. When we use the keyword "await", we must annotate the function declaration with "async".

The http response returns a json response. The function ".json()" converts the response to a real javascript array. Finally all items of the response are pushed (one by one with the spread operator ...) to the animals array.

Next, the overview of animals is generated. In the previous section 5.1.4 we defined a function renderAnimals() that creates a table row for each animal and assigns events (hover, click, ...) on each row. We can reuse this function since nothing has fundamentally changed: only the way the array is populated is different. In the next step we combine fetching animals from the back-end and rendering them in a table (with filter functionality and events) to ensure that rendering starts after all animals are loaded.



```
const fetchAndRenderAnimals = async () => {
  await fetchAnimals();
  renderAnimals();
};

fetchAndRenderAnimals();
```

6.2.2 Exercise: Fetch games from the back-end and show them on table-overview



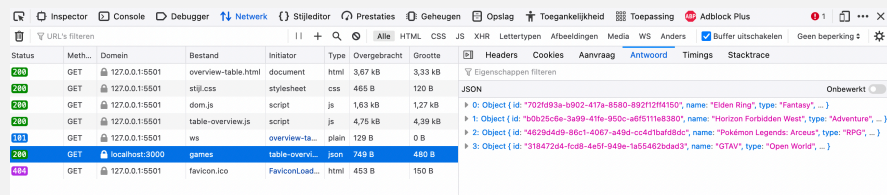
Replace in table-overview.js the hardcoded arraylist of games by a fetch on the back-end.



Evaluation criteria: You should have updated in the file table-overview.js

- ☐ Added a function fetchGames that fetches all games in the back-end and pushes them to the (empty) array games[];

- ❑ Added a function `fetchAndRenderGames()` that fetches the games with the function `fetchGames()` and renders them like before
- ❑ Called the new function `fetchAndRenderGames`.
- ❑ Removed the function call `renderAllGames()` as you did in 5.1.4 .
- ❑ When user navigates to the overview page, the browser sends a GET request to `localhost:3000`



Status	Meth...	Domain	Bestand	Initiator	Type	Overgebracht	Grootte	Headers	Cookies	Aanvraag	Antwoord	Timings	Stacktrace
200	GET	127.0.0.1:5501	overview-table.html	document	HTML	3,67 kB	3,33 kB						
200	GET	127.0.0.1:5501	stijl.css	stylesheet	css	465 B	120 B				JSON		
200	GET	127.0.0.1:5501	dom.js	script	js	1,63 kB	1,27 kB						
200	GET	127.0.0.1:5501	table-overview.js	script	js	4,75 kB	4,39 kB						
200	GET	127.0.0.1:5501	ws	overview-la...	plain	129 B	0 B						
200	GET	localhost:3000	games	table-overvi...	json	749 B	480 B				<pre>[{ "id": "702f693a-b602-417a-8580-892712f61501", "name": "Elden Ring", "type": "Fantasy", ... }, { "id": "b0b25c6e-3a99-41fe-950c-e6f511b83801", "name": "Horizon Forbidden West", "type": "Adventure", ... }, { "id": "462844d9-86c1-4067-a48d-c04d1baf88dc", "name": "Pokémon Legends: Arceus", "type": "RPG", ... }, { "id": "318472d4-fc08-4e5f-949e-1a55482bdad3", "name": "GTAV", "type": "Open World", ... }]</pre>		
200	GET	127.0.0.1:5501	favicon.ico	FaviconLoad...	html	453 B	150 B						

6.3 Refactor existing functionalities

6.3.1 Exercise: Check all other functionalities

All functionalities should keep working. After all, the only change you made is the way the array is built. If not, find the errors and refactor your code.

Not sure about the outcome? Take a look at the http-response in your browser's developer tools and convince yourself that there are no favourite games up to now.

6.4 Filter games on name

In Exercise **Fout! Verwijzingsbron niet gevonden**, you created a filter functionality ("rating higher than"). This filtering is implemented as an event on the input field. When the user changes the value of the input field, Javascript searches *in the present DOM (HTML)* for matching items.

However, when the database contains a lot of data, it is not recommended to fetch first the large amount of data and then filter it in the browser. In such cases, only the appropriate data should be fetched from back-end.

In this exercise you create a functionality that filters the data on the back-end and fetches only the filtered data from the server. You don't have to worry about the back-end: the necessary functions are already implemented. Try in the browser the following URL: <http://localhost:3000/games?query=e>.

The response shows all games in the back-end with a name containing the letter “e”. Try other values. The keyword “query” in the URL names the search value after “=” such that the back-end recognizes it as input and can use it as a parameter.

6.4.1 Exercise: Fetch games from back-end, filtered by name



Add a button “Fetch games” on the table-overview page. When user writes some value in the corresponding input fields and clicks on the button, the browser shows only games with a name containing the given characters.

- Use URL <http://localhost:3000/games?query=chars> where “chars” is the sequence of characters that the user has put in the input field.
- Show feedback to the user with a table caption. Default value (when user loads the overview page) is “All games”. When user submits a request, it shows the text “Games with name containing “chars””.
- Other functionalities (Show favourites, rating higher than) should still work. Note that they operate only on the fetched (filtered) data.

My Games

Show my favourites

Show all

Show games with rating higher than:

Fetch games

from backend with name containing:

e

Games with name containing "e"

Name	Type	Rating
Elden Ring	Fantasy	4
Horizon Forbidden West	Adventure	3.5
Pokémon Legends: Arceus RPG		3

Note that in the HTML DOM only table data have changed: the other DOM-elements (e.g. content of div#status) remain the same.



Evaluation criteria: You should have updated in the file table-overview.js

- ☐ Add a button and input field for the characters the user is looking for.
- ☐ Add a caption to the table

- ❑ Write a function “searchByFetch(chars)” that fetches only games containing “chars” in their name from back-end by sending request with given URL.
- ❑ Write a function “searchByFetchAndRender” that renders these games in the table.
- ❑ When user submits the button “Fetch Games”, browser sends a GET request to localhost:3000 as shown below

Status	Meth...	Domain	Bestand	Initiator	Type	Overgebracht	Grootte	Headers	Cookies	Aanvraag	Antwoord	Timings	Stacktrace
200	GET	127.0.0.1:5501	overview-table.html	document	html	3,67 kB	3,33 kB						
200	GET	127.0.0.1:5501	still.css	stylesheet	css	465 B	120 B						
200	GET	127.0.0.1:5501	dom.js	script	js	1,63 kB	1,27 kB						
200	GET	127.0.0.1:5501	table-overview.js	script	js	4,75 kB	4,39 kB						
200	GET	127.0.0.1:5501	ws	overview-ta...	plain	129 B	0 B						
200	GET	localhost:3000	games	table-overv...	json	749 B	480 B						
200	GET	127.0.0.1:5501	favicon.ico	FaviconLoad...	html	453 B	150 B						
200	GET	localhost:3000	games/queryen	table-overv...	json	638 B	369 B						

6.4.2 Extra Exercises

- Note that the functions “fetchAndRender()” and “searchByFetchAndRender()” do almost the same thing. Refactor both functions into one function that can handle both requests.
- Create a “reset” button that undoes the filtering.

6.5 Add game

6.5.1 Syntax

There are several types of http-requests. In the previous exercise we got animals from the back-end with a GET-request. This is the default method of a fetch. When we expect that the status on the back-end side will change, (for instance because we add an animal or modify it) we have to send a POST-request. An extra field “method” is added to the function fetch().



```
const adoptAnimal = async () =>
{
  const response = await
    fetch("http://localhost:3000/animals/adopt",
      { method: "POST", });
};
```

The browser can send a parameter with the request to the back-end. This can happen via the URL (as querystring, see 6.4.1). More complex data as objects are sent via the body of the http request, together with all other information the browser needs to forward to the back-end to have a valid http request. So extra parameters are necessary in the function “fetch()”.



```
const addAnimal = async () => {
  const animal = {name: "Woef", type: "Dog", age: 12 };
  const response = await fetch(
    "http://localhost:3000/animals",
    { method: "POST",
      headers:
        { Accept: "application/json",
          "Content-Type": "application/json", },
      body: JSON.stringify(animal),
    }
  );
};
```

The field “headers” tells the back-end that the request contains a parameter in JSON format (and not a string value e.g.). The line “JSON.stringify()” converts the real JavaScript object to JSON -format because the HTTP request can not handle Javascript objects.

Note that the URL <http://localhost:3000/animals> is identical to the URL in 6.2.2 . The back-end recognizes the request-method (GET vs POST) and knows that they must be handled in a different way.

6.5.2 Exercise: Make it possible to add a new game



Create new files “add-game.html” and “add-games.js”.

```

<main>
  <h2>Add a game</h2>
  <form id="add-game-form">
    <p>
      <label for="name">Name:</label>
      <input id="name" type="text" value="bb">
    </p>
    <p>
      <label for="type">Type:</label>
      <input id="type" type="text" value="ann">
    </p>
    <p>
      <label for="rating">Rating:</label>
      <input id="rating" type="number" value="2" max="15" min="1">
    </p>
    <p>
      <input id="submit" type="submit" value="Add Game" aria-label="submit">
    </p>
  </form>
  <div id="status">
</div>
</main>

```

Make sure that a new game is created in the back-end when the user submits the form.

- Use the URL <http://localhost:3000/games> . When the http POST-request is sent to this URL with a new game as json in body of request, the game is added to the back-end. The name of the added game is sent as a response.

Add a game

Name:

Type:

Rating:

Name: Genshin Impact -- Type: Chinese game -- Rating: 2

Status	Meth...	Domain	Bestand	Initiator	Type	Overgebracht	Grootte	Headers	Cookies	Aanvraagparameters	Antwoord	Timings	Stacktrace
200	GET	127.0.0.1:5501	add-game.html	document	html	3,65 kB	3,31 kB						
200	GET	127.0.0.1:5501	stijl.css	stylesheet	css	465 B	120 B						
200	GET	127.0.0.1:5501	dom.js	script	js	1,63 kB	1,27 kB						
200	GET	127.0.0.1:5501	add-game.js	script	js	1,05 kB	687 B						
200	GET	127.0.0.1:5501	ws	add-game.h...	plain	129 B	0 B						
200	GET	127.0.0.1:5501	favicon.ico	FaviconLoad...	html	453 B	150 B						
200	OPTL...	localhost:3000	games	fetch	plain	329 B	0 B						
200	POST	localhost:3000	games	add-game.js...	json	292 B	25 B						

JSON

```

{
  "name": "Genshin Impact",
  "rating": 2,
  "type": "Chinese game"
}

```

- User gets feedback in div#status.
- When the overview page is loaded, the new game is shown in the table.



Evaluation criteria: You should have updated in the file table-overview.js

- ❑ Add submit-event to form. Use “preventDefault()” to avoid built-in behaviour of the HTML form element.

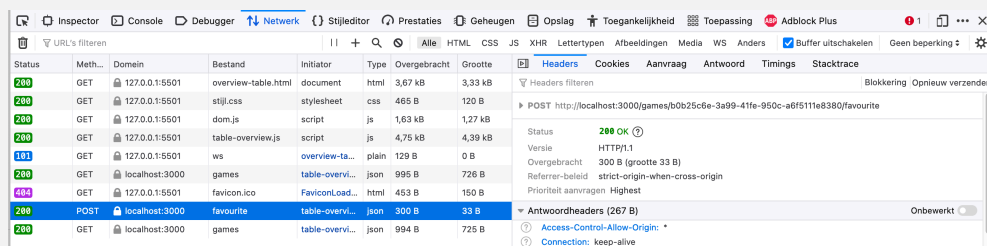
- ❑ Write function “addGame()”. This function
 - reads values for name, type and rating from input fields,
 - creates a game object with this name, type and rating (no field isFavourite)
 - sends the required http request to the back-end (cfr video)
 - and gives feedback in div#status to the user.

6.6 Toggle the isFavourite flag with double click

6.6.1 Exercise: toggle the isFavourite-flag



Create a double-click-event on each tablerow. When the user clicks twice on a game, the flag “isFavourite” is switched from true to false and vice versa. Use the post request <http://localhost:3000/games/:uuid/favourite>. Replace :uuid with the id of the game you want to make (un)favourite. Response of this request is a string with name of the favoured game.



Evaluation criteria: You should have updated in the file table-overview.js

- ❑ Added event “dblclick” on tablerow
- ❑ Created function toggleFavourite(game)
- ❑ This function fetches the given URL with post request. “uuid” is replaced by the id of game (\${game.id}).
- ❑ And this function calls again fetchAndRenderAllGames() to refresh data in browser.
- ❑ The browser sends 2 requests to localhost:3000: a POST request to modify a game, and a GET request to refresh the table data.

- ❑ Extra: put message in div#status “The game with name ... is now (not) my favourite”.

6.7 Delete game with button

Up to now, your application sends POST- and GET-requests to the back-end. It can also send a DELETE-request. Use this to make it possible to delete a game.

6.7.1 Exercise: make it possible to delete a game



Add in table-overview.html for each game a button “Delete”. When the user clicks on this button, the game is deleted on the server. The overview shows the remaining games.

Use the URL <http://localhost:3000/games/:uuid> with method=“DELETE”.

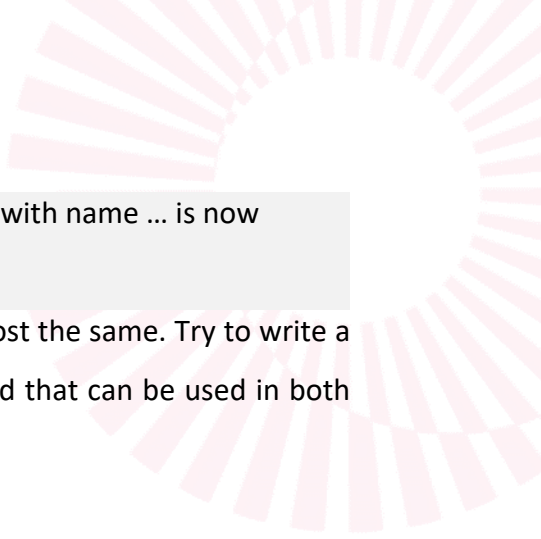
Replace “:uuid” by the id of the corresponding game.

Name	Type	Rating	Delete
Elden Ring	Fantasy	4	Delete
Horizon Forbidden West	Adventure	3.5	Delete
Pokémon Legends: Arceus RPG		3	Delete
GTA V	Open World	5	Delete



Evaluation criteria: You should have updated in the file table-overview.js

- ❑ Added a button “delete” to each table row
- ❑ Added a click-event on this button that calls a function deleteGame(game).
- ❑ Created the function deleteGame(game). This function
 - fetches the given URL with the delete request. “uuid” is replaced by the id of game ({game.id})
 - Calls again fetchAndRenderAllGames() to refresh data in browser.

- 
- ❑ Extra: put message in div#status “The game with name ... is now deleted.”

Extra: the functions “deleteGame” and “toggleFavourite” do almost the same. Try to write a generic function with parameters game, URL and request method that can be used in both cases.