Assignment - JavaScript

OPO	Front-end development
Place in the OPO	Part 2: Building a front-end with JavaScript
Documentation	See Toledo
Prerequisites	HTML, CSS, basic programming concepts (programming 1)
Learning Goals	Building a dynamic website with HTML, CSS and JavaScript
Product	The Games Library

Table of Contents

1 Project set-up

1.1 The Games Library

During this series of lab exercises, you are going to create a front-end for an existing back-end application. The application allows you to administer your games library: add and remove games from the library, add ratings for games and mark them as favourite.

Up until now, you have learned to create static websites with HTML and CSS. These skills are still necessary to build the foundations of your front-end, but from now on you are going to leverage **JavaScript** to make you website dynamic. Step by step, in each lesson, you will make a front-end app that consumes, processes and displays data from a back-end application.

1.2 Accept GitHub classroom assignment

For this project, we will use GitHub classroom to host your source code. In the first step, you will have to accept the assignment we have created for you. Accept the assignment by clicking this link and following the steps: https://classroom.github.com/a/d3uysPFx.

⚠ Do not skip the step where you must select your name in the list. Otherwise, we cannot correct your assignment! If you can’t find you name on the list, contact your lector.

1.3 Clone repository

After accepting the assignment, an individual repository within the classroom will be created. An URL to this repository will be presented (you might need to refresh the page). Copy this URL.

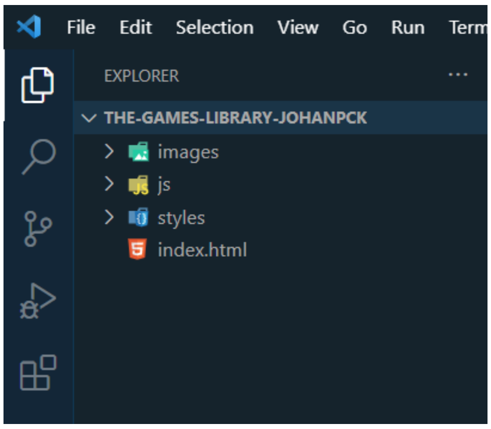
Open a terminal window (e.g. Git Bash) in a local folder (e.g. C:\front-end) and execute the following command:

```
git clone <URL_of_your_repo>
```

The URL of your repo looks similar to this:

```
https://github.com/UCLL-Frontend-afstand/the-games-library-johanpck
```

Open this folder in VSCode, a basic skeleton should already be available:



1.4 Periodically commit your work to GitHub

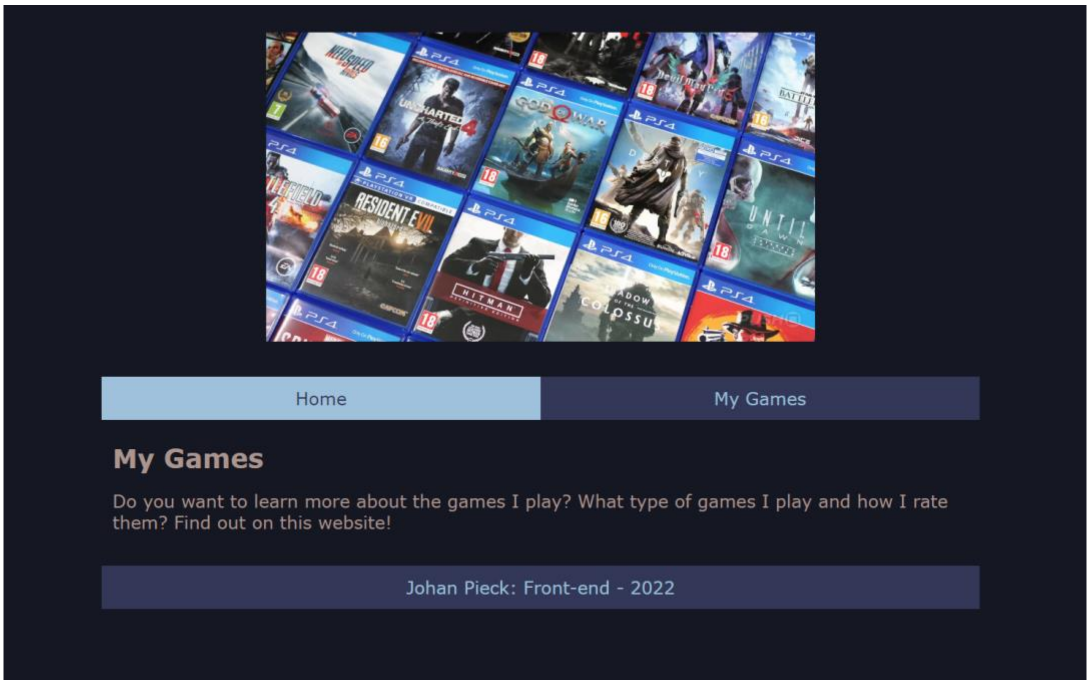
When finishing a new part of the assignment, or preferably multiple times while working, commit your work to GitHub so you have a backup and version history of your work. To do this, execute the following commands from your working directory (e.g. c:\projects\the-games-library-johanpck):

```
git add *
git commit -m "Your commit message"
git push -u origin master
```

1.5 Create a static home page

The first step is finishing the static welcome page of our website for which we have already provided a basic HTML structure. Preview the index.html in the browser: this page still has a very basic layout. Open index.html in VSCode and look at the code. We have included some comments (between the tags <!-- -->) which will give you a hint of what to do.

Use your creativity to create a nice and shiny homepage. Here is some inspiration:



My Games

Do you want to learn more about the games I play? What type of games I play and how I rate them? Find out on this website!

Johan Pieck: Front-end - 2022

Evaluation criteria: Index page

- ❑ index.html is styled using SASS
- ❑ A logo is placed in the header
- ❑ Your name is displayed in the footer

1.6 Overview games

Open the overview.html page: this is a page that we will use to display the resulting data from our JavaScript code. On that page you will need to create an empty element with a specific ID, so we can select it in JavaScript to display data.

First you will need to include the JavaScript files, so they are loaded when opening the webpage. This can be achieved by adding a script tag in the header of your HTML:

```
<script src="js/dom.js" defer></script>
```

The *defer* attribute makes sure that JavaScript files are loaded **after** the main HTML. This is necessary because some HTML elements need to be loaded before we can target them with JavaScript.

You are going to write all your code in “games.js”, which you can find in the “js” folder. The “dom.js” file already contains some helper functions that you will need to use in games.js. Therefore, it is important to load dom.js **before** games.js.

Evaluation criteria: Overview page

- ❑ overview.html is styled using SASS
- ❑ dom.js is included in the head
- ❑ games.js is included in the head, after dom.js
- ❑ A logo is placed in the header
- ❑ Main element contains an empty div with id “status”
- ❑ Your name is displayed in the footer

2 Objects & conditionals

2.1 Creating objects

Objects are variables that can contain many values. For instance, an animal is an object that has a name, is of a particular type and has an age.
In JavaScript you define an object as follows:

```
const animal1 = { name: “Albert”, type: “duck”, age: 2 }
```

As such you define an object called animal1 with name Albert, it is a duck (the type) and has an age of 2 years.

2.1.1 Exercise: Creating games

Create 4 different game objects in a new file games.js in the folder js.
A game has a name, is of a particular type and has a rating assigned.
If you are not familiar with gaming, here are some examples of popular games:

- ❖ Elden Ring, a fantasy game
- ❖ Horizon Forbidden West, an adventure game
- ❖ Pokemon Legends: Arceus, an RPG game
- ❖ GTA V, an open world game

Evaluation criteria: In game.js you should have:

- ❑ 4 different constants
- ❑ for each const: a name and a type field containing a string, and a rating field containing a float should be defined

2.2 Showing all objects

In order to show a new part of text in a new paragraph on the screen, you can use the function addStatus in the JavaScript file dom.js. The parameter status is the text that is added to the HTML element with id “status” and will be shown on the screen.

```
const addStatus = (status) => { … }
```

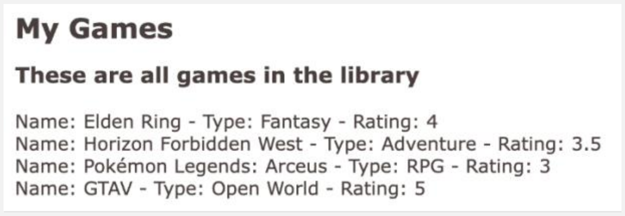
In order to show the sentence Name: Albert – Type: duck – Age: 2 on the screen, you can use the following code:

```
addStatus(`Name: ${animal1.name} - Type: ${animal1.type}
-    Age: ${animal1.age}`)
```

2.2.1 Exercise: Showing games

Show the 4 game objects created in section 3.1 on the overview page.

The result should look as follows



My Games

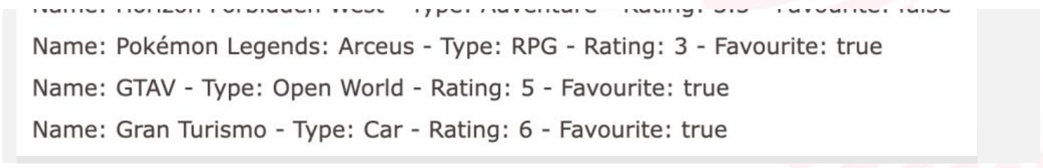
These are all games in the library

```
Name: Elden Ring - Type: Fantasy - Rating: 4
Name: Horizon Forbidden West - Type: Adventure - Rating: 3.5
Name: Pokémon Legends: Arceus - Type: RPG - Rating: 3
Name: GTAV - Type: Open World - Rating: 5
```

Evaluation criteria: You should have defined in game.js:

- ❑ a function printAllGames
- ❑ a function toString
- ❑ the function printAllGames calls 4 times the addStatus method and 4 times the toString method
- ❑ at the end of this file you should have called only the function printAllGames

2.2.2 Extension: Showing more games with more information ...



```
Name: Pokémon Legends: Arceus - Type: RPG - Rating: 3 - Favourite: true
Name: GTAV - Type: Open World - Rating: 5 - Favourite: true
Name: Gran Turismo - Type: Car - Rating: 6 - Favourite: true
```

Evaluation criteria: You should only have changed the following in your games.js

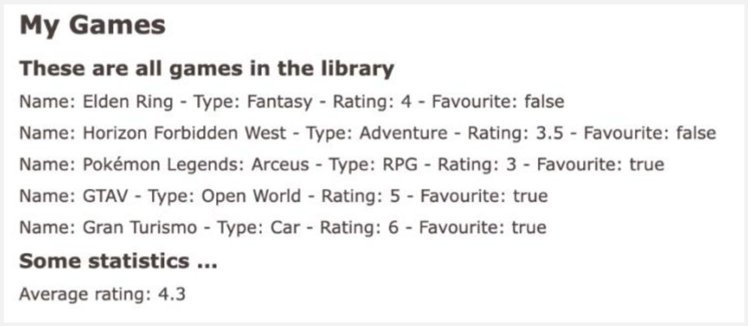
- ❑ Added a favourite field containing a boolean to each object
- ❑ added a new const for the Gran Turismo game with all its specific values
- ❑ within the function printAllGames you should only add 1 line of code to add show the new game on the screen
- ❑ within the function toString you should only add 1 little piece of code to show the value of favourite and the extra text indicating whether it is a favourite game or not

2.3 Adding more functionality

2.3.1 Exercise: Showing the average rating of all your games

Calculate and show the average rating of all the games.

The result should look as follows



My Games

These are all games in the library

```
Name: Elden Ring - Type: Fantasy - Rating: 4 - Favourite: false
Name: Horizon Forbidden West - Type: Adventure - Rating: 3.5 - Favourite: false
Name: Pokémon Legends: Arceus - Type: RPG - Rating: 3 - Favourite: true
Name: GTAV - Type: Open World - Rating: 5 - Favourite: true
Name: Gran Turismo - Type: Car - Rating: 6 - Favourite: true
```

Some statistics ...

```
Average rating: 4.3
```

Evaluation criteria: In the file game.js you should have defined

- ❑ a function getAverageRating calculating the average rating
- ❑ 1 addStatus call generating the title
- ❑ 1 addStatus call showing the average rating

The average result on the page should be

- ❑ 4.3

2.4 Conditionals

To do something in a specific situation and to do something else in another situation, you can use an if-then-else.

```
if (condition is true) {
    do something
} else {
    do something else
}
```

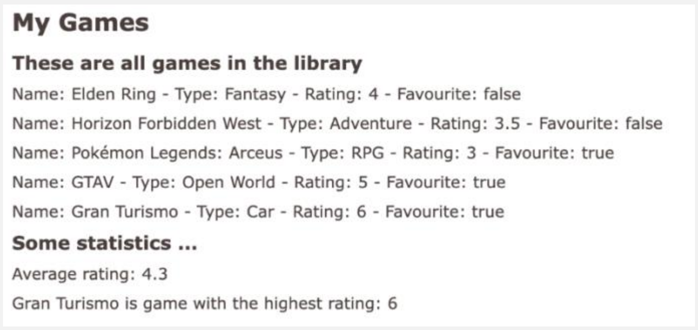
An example of such an if-then-else in the animals demo is shown below. If the age of the first object is greater than the age of the second object, assign the first object to result, else if the age of the second object is greater than the age of the first object, assign the second object to the result.

```
const getOldestAnimal = (first, second) => {
  let result = null
  if (first.age > second.age) {
    result = first
  } else if (first.age < second.age) {
    result = second
  }
  return result;
}
```

2.4.1 *Exercise: Showing game with the highest rating*

Add to the statistics the game with the highest ranking

The result should look as follows



My Games

These are all games in the library

Name: Elden Ring - Type: Fantasy - Rating: 4 - Favourite: false

Name: Horizon Forbidden West - Type: Adventure - Rating: 3.5 - Favourite: false

Name: Pokémon Legends: Arceus - Type: RPG - Rating: 3 - Favourite: true

Name: GTAV - Type: Open World - Rating: 5 - Favourite: true

Name: Gran Turismo - Type: Car - Rating: 6 - Favourite: true

Some statistics ...

Average rating: 4.3

Gran Turismo is game with the highest rating: 6

Evaluation criteria: In the file game.js you should have defined

- ❑ a function getHighestRating
- ❑ within this function you should have used multiple times the if
- ❑ this function returns the whole object (not only the highest rating)
- ❑ this function is only called once to show the result on the page, so not multiple times!

2.5 Conditional ternary oparator

The if-then-else can also be defined shorter with the conditional ternary operator.

```
condition is true ? do something : do something else
```

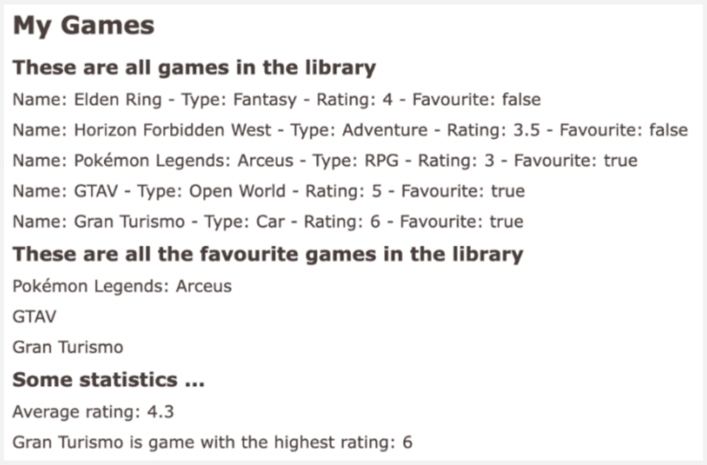
An example of such a conditional ternary operator in the animals demo is shown below. If an oldest animal is found, the name of the oldest animal is shown, otherwise the sentence “Animals are equally old.” is shown on the page.

```
const oldest = getOldestAnimal(animal1, animal2);
addStatus(`Oldest animal: ${oldest !== null ?
oldest.name : "Animals are equally old."}`);
```

2.5.1 *Exercise: Showing your favourite games*

Show the list of your favourite games, only the names should be shown on the page.

The result should look as follows



```
My Games
These are all games in the library
Name: Elden Ring - Type: Fantasy - Rating: 4 - Favourite: false
Name: Horizon Forbidden West - Type: Adventure - Rating: 3.5 - Favourite: false
Name: Pokémon Legends: Arceus - Type: RPG - Rating: 3 - Favourite: true
Name: GTAV - Type: Open World - Rating: 5 - Favourite: true
Name: Gran Turismo - Type: Car - Rating: 6 - Favourite: true
These are all the favourite games in the library
Pokémon Legends: Arceus
GTAV
Gran Turismo
Some statistics ...
Average rating: 4.3
Gran Turismo is game with the highest rating: 6
```

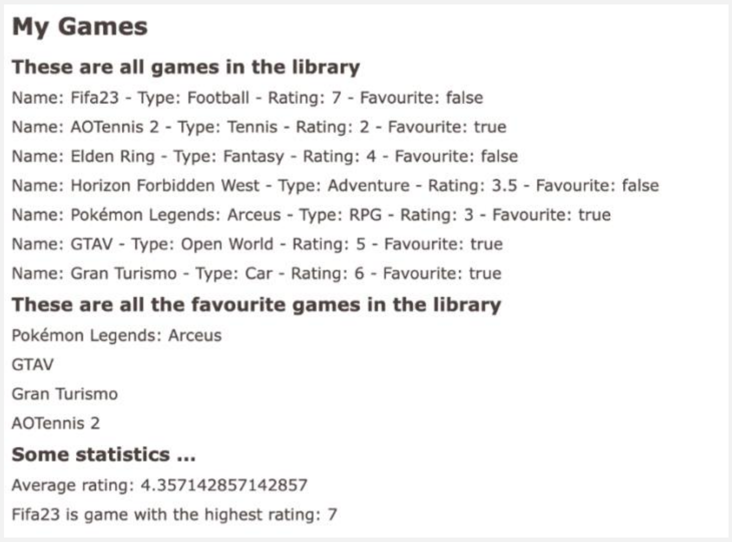
Evaluation criteria: In the file game.js you should have defined

- ❑ a function printFavouriteGames
- ❑ a function isFavourite = (game)
- ❑ the function printFavouriteGames uses the isFavourite function
- ❑ the function printFavouriteGames uses the conditional ternary operator

2.5.2 Exercise: Adding more game objects

Add 2 more games at the beginning of the games.js file.

The result should look as follows



```
My Games
These are all games in the library
Name: Fifa23 - Type: Football - Rating: 7 - Favourite: false
Name: AOTennis 2 - Type: Tennis - Rating: 2 - Favourite: true
Name: Elden Ring - Type: Fantasy - Rating: 4 - Favourite: false
Name: Horizon Forbidden West - Type: Adventure - Rating: 3.5 - Favourite: false
Name: Pokémon Legends: Arceus - Type: RPG - Rating: 3 - Favourite: true
Name: GTAV - Type: Open World - Rating: 5 - Favourite: true
Name: Gran Turismo - Type: Car - Rating: 6 - Favourite: true
These are all the favourite games in the library
Pokémon Legends: Arceus
GTAV
Gran Turismo
AOTennis 2
Some statistics ...
Average rating: 4.357142857142857
Fifa23 is game with the highest rating: 7
```

Evaluation criteria: In the file game.js you should have

- ❑ added 2 new objects at the first two lines of this file
- ❑ added 2 lines of code to the printAllGames function
- ❑ added 2 lines of code to the printFavouriteGames function
- ❑ added some code to the getHighestRating function
- ❑ added some code to the getAverageRating function

3 Arrays

3.1 Keeping track of our games in an array

Up until now we've been keeping track of our games using specific variables for each game. This means that in our function *printAllGames* we must call *addStatus* explicitly for every game. We can automate this by storing a list of our games in an array and then looping over each element.

```
const animals = [
  animal1,
  animal2
];

for (const animal of animals) {
  console.log(animal.name);
}
```

Once we have our games stored in an array, we can loop over the array and execute some functionality on every element.

3.1.1 Exercise: Automatically show all games

Create an array, "games", that contains all your existing games.

Update the printAllGames function to use a for..of loop. Using this for..of loop, call the addStatus function for each element in the "games" array.

Evaluation criteria: You should have in the games.js file:

- ❑ an array, "games", that contains all your games
- ❑ updated the printAllGames function to use a for..of loop
- ❑ only use 1 addStatus call (which happens inside the loop)

3.2 Updating existing functionality

3.2.1 Exercise: Updating the average rating of all your games using arrays

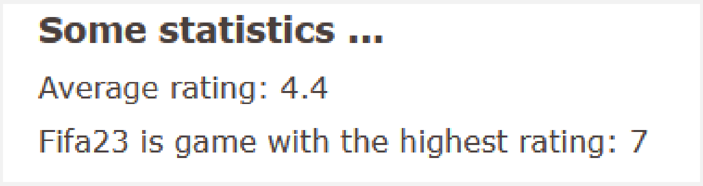
Now that we have an array containing all our games, we can automatically calculate the average rating using a for loop. This way, we don't have to update the getAverageRating function every time we add or remove a game. When calculating the average age of our farm animals, we use a simple for-loop:

```
let result = 0;
for (const animal of animals) {
  result += animal.age;
}
return result / animals.length;
```

Calculate and show the average rating of all the games by looping over the "games" array.

The result should look the same as before, but with an average rating calculated using a loop.

If you get a big number with a lot of decimals, you can set the number of decimals using the built-in JavaScript toFixed function.



```
Some statistics ...
Average rating: 4.4
Fifa23 is game with the highest rating: 7
```

Evaluation criteria: You should have updated in the file game.js

- ❑ The function getAverageRating to calculating the average rating using a loop

The average result on the page should be

- ❑ 4.4

You can now add new games to your “games” array to see that your page will automatically update all the elements without you having to write additional code.

3.2.2 Exercise: Showing game with the highest rating using arrays

We’ve already updated some parts of our code to work with the new “games” array, but we still need to update our getHighestRating function to do this as well.
We can get the highest rating by using a loop as well. Make sure that no error is thrown when the “games” array is empty. Add an if statement that displays a message if that is the case, instead of getting the highest rating.

Update the getHighestRating function to get the highest rating game using a loop.

Evaluation criteria: You should have updated in the file game.js

- ❑ The function getHighestRating to get the highest rating game using a loop
- ❑ Only 1 if statement is needed, inside the loop

3.2.3 Exercise: Show your favourite games using arrays

We still have one function that is using our hard-coded game objects instead of the “games” array: printFavouriteGames. We should update this function to use a loop as well.

Update the printFavouriteGames function to get loop over all the games and print the ones that have the *favourite* property set to *true*.

Evaluation criteria: You should have updated in the file game.js

- ❑ The function printFavouriteGames to print all games where the *favourite* property is set to *true*
- ❑ Only use 1 conditional, inside the.

3.3 Array destructuring

3.3.1 Syntax

We have now updated all our functions to use our new “game” array. We now also want to show the first 2 games in our array. We can extract the first 2 items from an array by using the destructuring syntax. In the case of our farm animals, this goes as follows:

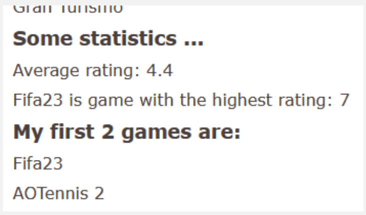
```
const animals = [ /* All our animals */ ];
const [first, second] = animals;
```

3.3.2 Exercise: Destructuring the first 2 games from our “games” array

Extract the first 2 games from our array and show their name at the bottom of the page by calling addStatus for each of them. Add a meaningful title for them of course.

Evaluation criteria: You should have updated in the file game.js

- ❑ Added at the bottom some code to extract the first 2 games using the destructuring syntax.
- ❑ Added at the bottom some code to show these 2 games on our page.



```
Gran Turismo
Some statistics ...
Average rating: 4.4
Fifa23 is game with the highest rating: 7
My first 2 games are:
Fifa23
AOTennis 2
```

You can now re-arrange the items in your “games” array to check that your page displays different results.

3.4 Add games of your best friend

3.4.1 Syntax

Your best friend sees that you are keeping track of your games and would like to use the same overview. You decide to merge your list of games with their games, but keep them in 2 separate arrays. When you want to print all your games, you can *spread* the 2 arrays together. When we want to spread 2 arrays together, we can do this as follows:

```
const animals = [ /* All our animals */ ];
const otherAnimals = [ /* Some other animals */ ];
const allAnimal = […animals, …otherAnimals]
```

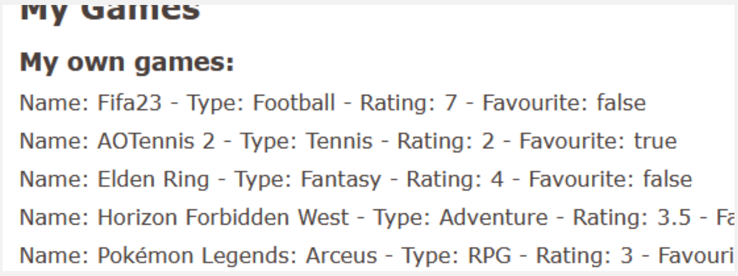
3.4.2 Exercise: Add an array for games from your best friend

Update the printAllGames function with an array as parameter instead of using the “games” array by default. Also update the loop inside this function to use this received array. Update all invocations of the printAllGames function to include this new parameter. Also move the instruction to print a title from the printAllGames function outside of the function, and print it *before* calling the printAllGames function.

Add a new array, “friendGames”, and put some games inside it. Print these games at the bottom of your page by calling the printAllGames function using this new array.

Evaluation criteria: You should have updated in the file game.js

- ❑ Updated the printAllGames function to receive an array as parameter
- ❑ Updated the printAllGames to loop over the received array instead of the “games” array.
- ❑ Updated all invocations of printAllGames to pass the “games” array
- ❑ Removed the addStatus call that prints the title inside printAllGames to outside the function

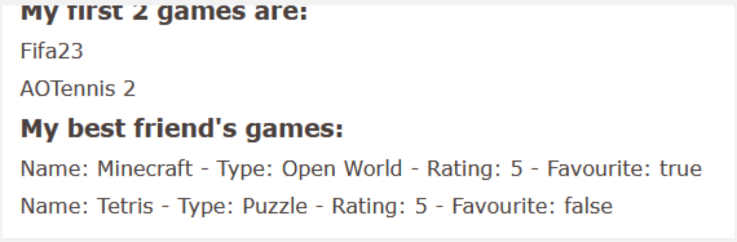


My Games

My own games:

Name: Fifa23 - Type: Football - Rating: 7 - Favourite: false
Name: AOTennis 2 - Type: Tennis - Rating: 2 - Favourite: true
Name: Elden Ring - Type: Fantasy - Rating: 4 - Favourite: false
Name: Horizon Forbidden West - Type: Adventure - Rating: 3.5 - Fa
Name: Pokémon Legends: Arceus - Type: RPG - Rating: 3 - Favouri

- ❑ Added at the bottom a new array with games from your friend.
- ❑ Added at the bottom some code to show these games.



My first 2 games are:

Fifa23
AOTennis 2

My best friend's games:

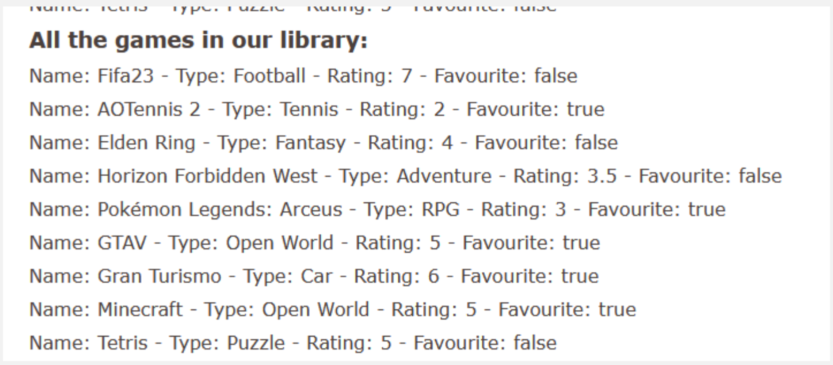
Name: Minecraft - Type: Open World - Rating: 5 - Favourite: true
Name: Tetris - Type: Puzzle - Rating: 5 - Favourite: false

3.4.3 *Exercise: Show all the games, both my own and the ones from my best friend*

Add some code that will print all the games that we have, both my own games and the ones from my friend. Do this by spreading the 2 arrays together into 1 array and calling passing this new array on to the printAllGames function as a parameter

Evaluation criteria: You should have updated in the file game.js

- ❑ Created a new array at the bottom that contains both my games and the ones from my friend. Use the spread operator for this.
- ❑ Displayed these games using the printAllGames function.



```
All the games in our library:
Name: Fifa23 - Type: Football - Rating: 7 - Favourite: false
Name: AOTennis 2 - Type: Tennis - Rating: 2 - Favourite: true
Name: Elden Ring - Type: Fantasy - Rating: 4 - Favourite: false
Name: Horizon Forbidden West - Type: Adventure - Rating: 3.5 - Favourite: false
Name: Pokémon Legends: Arceus - Type: RPG - Rating: 3 - Favourite: true
Name: GTAV - Type: Open World - Rating: 5 - Favourite: true
Name: Gran Turismo - Type: Car - Rating: 6 - Favourite: true
Name: Minecraft - Type: Open World - Rating: 5 - Favourite: true
Name: Tetris - Type: Puzzle - Rating: 5 - Favourite: false
```

4 Functions

4.1 Refactoring your code using higher-order functions

4.1.1 Syntax

To loop over an array you can use the forEach higher-order function instead of the for...of loop. In the example below, the array animals is looped through using the forEach function., and each element is stored in the variable “animal”. When the age of the animal is greater than the age of the animal currently stored in the variable result (containing the animal with the highest age untill now), the animal will be stored in the variable result. By doing this, we are sure that “result” will contain the oldest animal.

```
const getOldestAnimal = () => {
  let result = animals[0];

  animals.forEach((animal) => {
    if (animal.age > result.age) {
      result = animal;
    }
  });

  return result;
};
```

4.1.2 Exercise: Use forEach() instead of for...of loops

Use the higher-order forEach function to loop over your arrays.

Evaluation criteria: You should have updated in the file game.js

- ❑ The for..of loop is changed in a forEach in the functions printAllGames, getAverageRating, getHighestRating and printFavouriteGames

4.1.3 Syntax

The filter function is another example of a higher-order function. It filters the elements of an array with the defined filter. In the example below, the animals array is filtered on only those animals of which the name contains the given chars (capitals are not important, therefor we apply the toLowerCase function on the given chars and the name of the i-th animal to which we apply the filtering). When the whole array is done filtering, a new array with only the animals matching the given chars is the result.

```
const printAllAnimals = (chars) => {
  animals
    .filter((animal) => chars ?animal.name.toLowerCase()
       .includes(chars.toLowerCase()) : true)
    .forEach((animal) => addStatus(toString(animal)));
}
```

4.1.4 Exercise: Use the filter function for showing the favourite games.

Use the filter function to filter only the favourite games and showing only these on the screen.

Evaluation criteria: You should have updated in the file game.js

- ❑ The function printFavouriteGames should use the filter function. The forEach doesn’t need to check anything anymore: it only prints the name of the games in the filtered array on the screen.
- ❑ You still should use the isFavourite function.

4.2 Adding new functionality

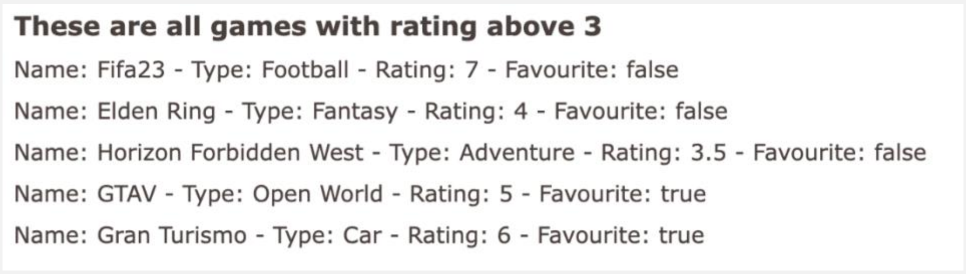
4.2.1 Exercise: Filter games on rating above 3

Implement a function that only shows the games with a rating higher than the given rating. Call this function with a rating of 3.

Evaluation criteria: You should have updated in the file game.js

- ❑ A new function printGamesRatingAbove
- ❑ With parameters games and rating
- ❑ With a combination of the filter function and forEach function
- ❑ With code to print it to the screen (as shown in the screenshot below)
- ❑ This function should be called at the bottom of the file, with the games array as the first parameter, and a rating of 3 as the second parameter.

  The resulting text when the function is called with these parameters:



```
These are all games with rating above 3
Name: Fifa23 - Type: Football - Rating: 7 - Favourite: false
Name: Elden Ring - Type: Fantasy - Rating: 4 - Favourite: false
Name: Horizon Forbidden West - Type: Adventure - Rating: 3.5 - Favourite: false
Name: GTAV - Type: Open World - Rating: 5 - Favourite: true
Name: Gran Turismo - Type: Car - Rating: 6 - Favourite: true
```

4.2.2 Syntax

A higher-order function accepts another function as parameter. We can use this to apply a custom filter to an array. For instance, in this example we will supply a custom filter to the animals array that we also supply as a parameter:

```
const filterAndPrintAnimals = (animals, customFilter) =>
{
  animals
    .filter(customFilter)
    .forEach((animal) => addStatus(toString(animal)));
}

filterAndAnimals(animals, (animal) =>
animal.name.toLowerCase().includes(“al”));
```

This will print all the animals that have the characters “al” in their name.

4.2.3 Exercise: Create a function that applies a custom filter

Implement a function that accepts a custom filter as a parameter and applies this custom filter to the supplied games. Call this function 2 times, once to print all favourite games, and once to print all games of type “Open World”. Do not use the existing isFavourite function, but write the custom filter inline.

Evaluation criteria: You should have updated in the file game.js

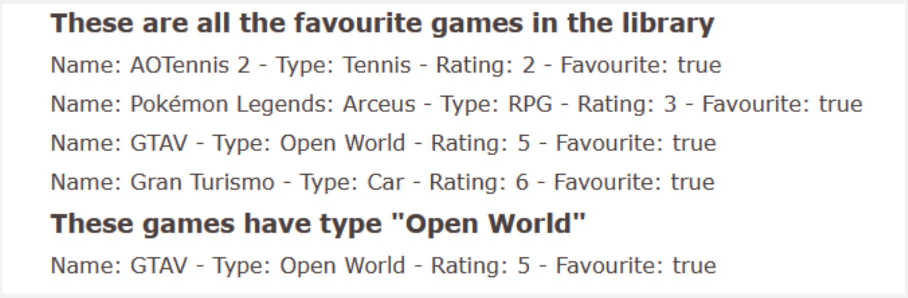
- ❑ A new function filterAndPrintGames
- ❑ With parameters games and customFilter
- ❑ Which filters the supplied games array using the supplied custom filter, and a forEach function that prints the filtered games.
- ❑ With code to print it to the screen (as shown in the screenshot below)

❑ This function should be called twice at the bottom of the file

❑ Once to filter all favourite

❑ Once to filter all games that have the type “Open World”.

❑ The resulting text when the function is called with these parameters:



```
These are all the favourite games in the library
Name: AOTennis 2 - Type: Tennis - Rating: 2 - Favourite: true
Name: Pokémon Legends: Arceus - Type: RPG - Rating: 3 - Favourite: true
Name: GTAV - Type: Open World - Rating: 5 - Favourite: true
Name: Gran Turismo - Type: Car - Rating: 6 - Favourite: true
These games have type "Open World"
Name: GTAV - Type: Open World - Rating: 5 - Favourite: true
```

4.2.4 Syntax

Another higher-order function that is built into arrays is the “map” function. This function will apply a transformation to each element, and return a new array with the result.
We can use this to first call the “toString” function on all our elements, and call addStatus on every element of the resulting array:

```
const printAllAnimals = (animals) => {
  animals
    .map(toString)
    .forEach(addStatus);
}
```

4.2.5 Exercise: Update printAllGames to use the map function

Update the printAllGames function to first transform every element with the toString function, and call addStatus on each element of the resulting array.

Evaluation criteria: You should have updated in the file game.js

❑ Updated the printAllGames function

❑ Use a combination of map function and the forEach function

❑ The map function will first call the toString function

- ❑ The forEach function will then call addStatus on every element of the "mapped" array.

5 DOM and Events

In order to keep an overview of our work, we start with new files.
Create a HTML file “table-overview.html”. Add page structure (e.g. navigation) as before.
Create a js file “table-overview.js” and include in the new HTML file. You can include all other js-files in table-overview.html if necessary.

5.1 Creating elements via Javascript

5.1.1 Syntax

We can add, update and remove HTML elements using javascript.

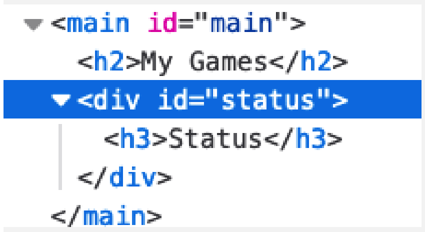
```
const footer = document.createElement("footer");
footer.innerHTML = "some text"
document.querySelector("body").appendChild(footer);
footer.className = "message";
```

First, element footer is created. The element is filled with the text “Some text”. Then footer is added in the DOM as last child of HTML element <body>. Finally, the class-attribute “message” is added to the footer.
Remark we use “querySelector()” and not “getElementById()”: querySelector returns the DOM element with given (CSS)-selector (here “body”), see also https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector .

5.1.2 Exercise: Create elements div#status and h3 in that div

Write javascript code that creates a div with id “status” in the (new) HTML-file. Add an element “h3” with content “Status” to the new div.

The result should look like



```
▼<main id="main">
   <h2>My Games</h2>
  ▼<div id="status">
     <h3>Status</h3>
   </div>
 </main>
```

Evaluation criteria: In the file overview-table.js you should have
- ❑ Created elements “div” and h3

- ❑ Added the id “status” to the new div
- ❑ Added some text to h3
- ❑ Appended h3 to the new div and the div to the element main.

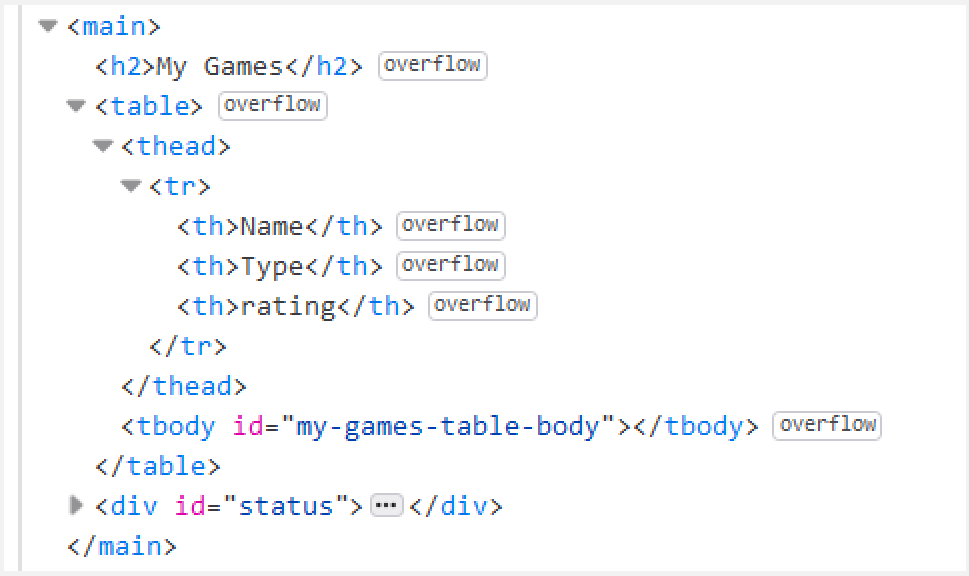
5.1.3 *Exercise: Create new table*

Likewise, we can also create a new table where we can keep track of our animals. This is cleaner than just writing our content in paragraphs.

Add a new table in the table-overview.html file.

The table should have 4 headings: Name, Type, and Rating.
For now, the tbody element should be empty, and should have id “my-games-table-body”.

The result should look like this:



```
▼<main>
   <h2>My Games</h2> overflow
  ▼<table> overflow
    ▼<thead>
      ▼<tr>
         <th>Name</th> overflow
         <th>Type</th> overflow
         <th>rating</th> overflow
       </tr>
     </thead>
     <tbody id="my-games-table-body"></tbody> overflow
   </table>
  ▶<div id="status">…</div>
 </main>
```

Evaluation criteria: In the file table-overview.html you should have

- ❑ Created a new table element
- ❑ Added the required HTML to the table element
- ❑ Made sure that the tbody element has id “my-games-table-body”

5.1.4 *Exercise: Populating our table with game data*

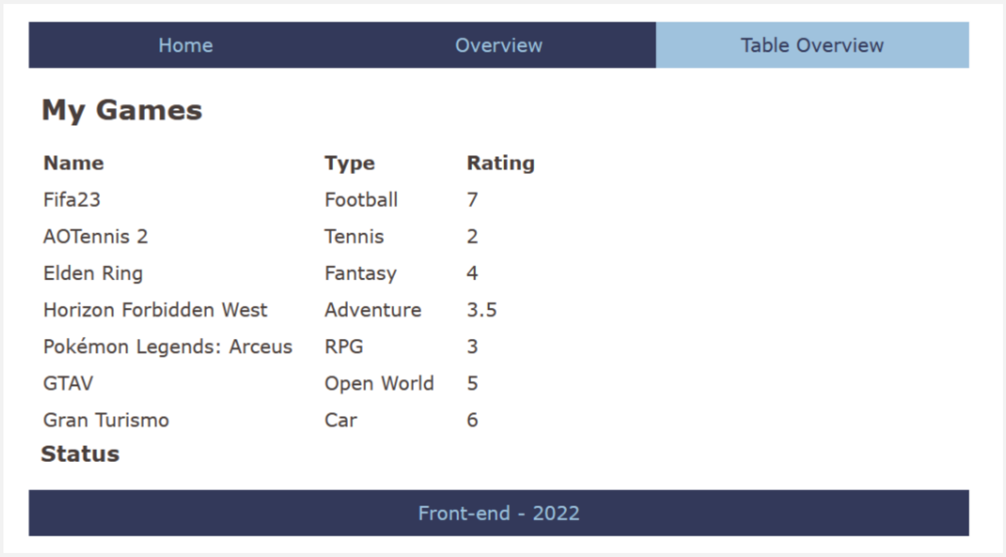
Now that we have our table, we can create a function that will automatically populate it with game data. We can use forEach to create a new row in tbody for each game that we have.

Write a function, renderGames, that accepts a games array as parameter and creates a new table row for each game inside the array.

Copy the games array from "games.js" and add it to the top of the file "table-overview.js", so that the array can be reused.

Call the function with the games array.

The result should look like this:



My Games

Name	Type	Rating
Fifa23	Football	7
AOTennis 2	Tennis	2
Elden Ring	Fantasy	4
Horizon Forbidden West	Adventure	3.5
Pokémon Legends: Arceus	RPG	3
GTAV	Open World	5
Gran Turismo	Car	6

Status

Front-end - 2022

Evaluation criteria: In the file overview-table.js you should have

- ❑ Created a new function with parameter "games", that will create a new table row for each game in the games array.
- ❑ Copied the games array from "games.js" to "table-overview.js"
- ❑ Called the newly created function with the games array as parameter.

5.2 Adding events

With javascript the webpage can be made more interactive. You can define effects that happen when user performs some action like hovering with mouse, clicking etc.

5.2.1 *Syntax*

```
tableRow.addEventListener("click", () => selectAnimal(animal));
    tableRow.addEventListener("mouseover", () => tableRow.className =
"select");
    tableRow.addEventListener("mouseout", () => tableRow.className = "");
```

The method “addEventListener” makes that something fires when user e.g. clicks on the element. The first parameter (“click”) is the type of the event. The second parameter is the function we want to call when the event occurs.

5.2.2 *Exercise: add hover-effect on div#status*

When user hovers over the (newly created) div#status, background-color is changed. Take care that background-color disappears when mouse is removed from div.

Evaluation criteria: In the file overview-table.js you should have

- ❑ Added mouseover- and mouseout-event to div#status
- ❑ Added and removed attribute style with setAttribute() and removeAttribute()
- ❑ Or added and removed some class-attribute

5.2.3 *Exercise: add click-effect on h2*

When the user clicks on the h2 title on our page, javascript should choose a random textcolor for that element. It is not allowed to add an id- or class-attribute to the h2 element.

Use

- Math.floor() and Math.random() to create random number between 0 and 360
- Hsl(number, 100%, 50%) to define a color

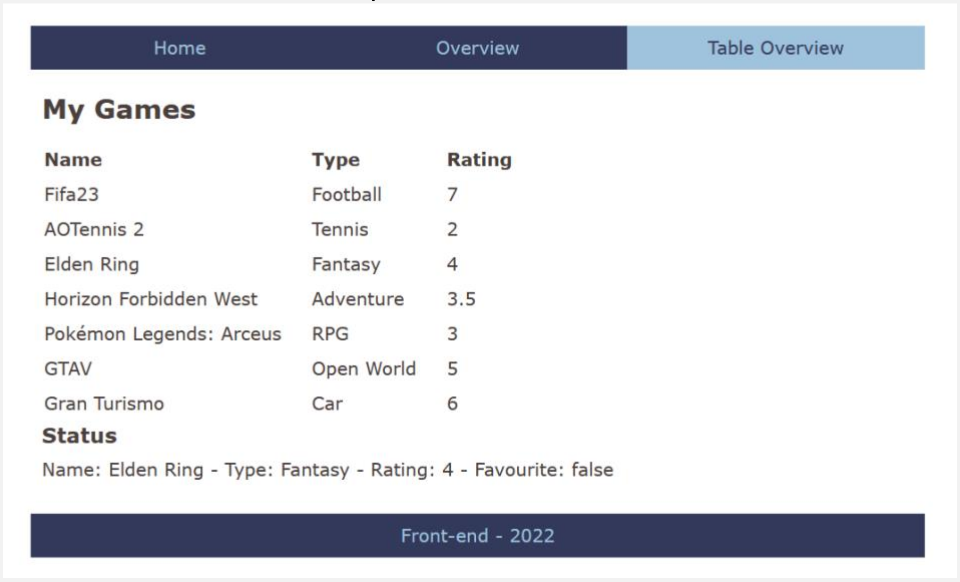
Evaluation criteria: In the file overview-table.js you should have

- ❑ Used querySelector() to select element h2;
- ❑ Defined a function createColor = () => that returns a random color in hsl-format
- ❑ Added a click-event to h2

5.2.1 Exercise: Add click-action on each table row

Our table is visible now, but we want to show all game information in div#status when a user clicks on a game.

Add an event listener to each table row (inside the renderGames function) that will display the games toString information in the "div#status" element whenever a user clicks on that specific row.



My Games

Name	Type	Rating
Fifa23	Football	7
AOTennis 2	Tennis	2
Elden Ring	Fantasy	4
Horizon Forbidden West	Adventure	3.5
Pokémon Legends: Arceus	RPG	3
GTAV	Open World	5
Gran Turismo	Car	6

Status

Name: Elden Ring - Type: Fantasy - Rating: 4 - Favourite: false

Front-end - 2022

Evaluation criteria: In the file overview-table.js you should have

- ❑ Updated the renderGames function to add a "click" event listener to each created row
- ❑ The event listener should display all the games information in the "div#status" element.
- ❑ When second row is clicked, information of first game is removed.

5.3 Adding filtering

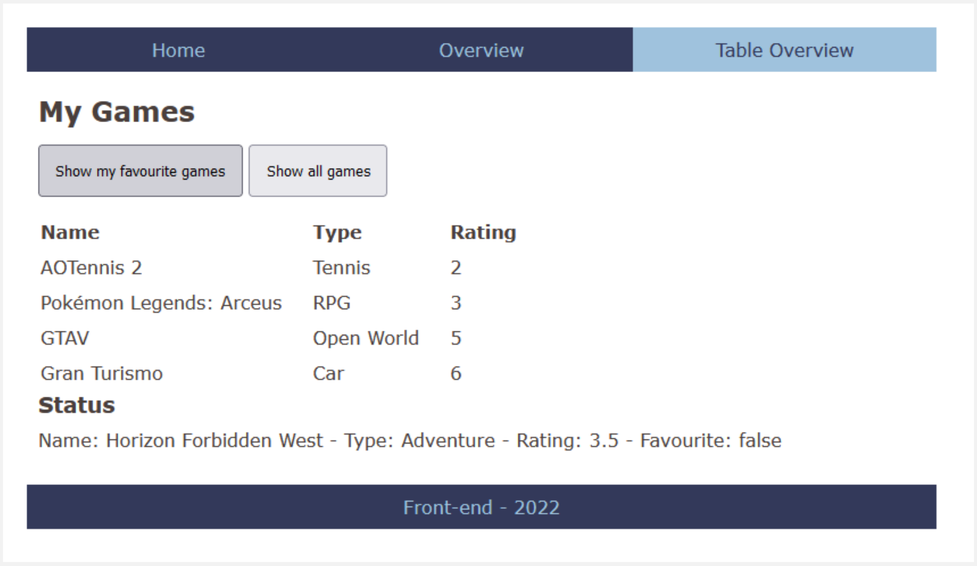
Let’s make the overview page more interactive. We can add some input field to the overview page. When user types some characters in it, only animals with name including these characters are shown.

```
const renderAllAnimals = () => {
  const chars = document.getElementById("field").value;
  animals
    .filter((animal)=>animal.name.includes(chars))
    .forEach((animal) => {…}
}
```

The variable “chars” reads the characters user typed in the input field with id “field”. Then only those animals with name containing chars are filterd by the higher order function “filter”. Finally, something happens with each of those animals (defined in “{…}”).

5.3.1 Exercise: Show only favorite games

Create a button in overview-table.html with title “Show my Favourites”. When user clicks on this button, only favourite games are shown in the table. Also create a button “Show all” that resets the list.



My Games

Name	Type	Rating
AOTennis 2	Tennis	2
Pokémon Legends: Arceus	RPG	3
GTAV	Open World	5
Gran Turismo	Car	6

Status

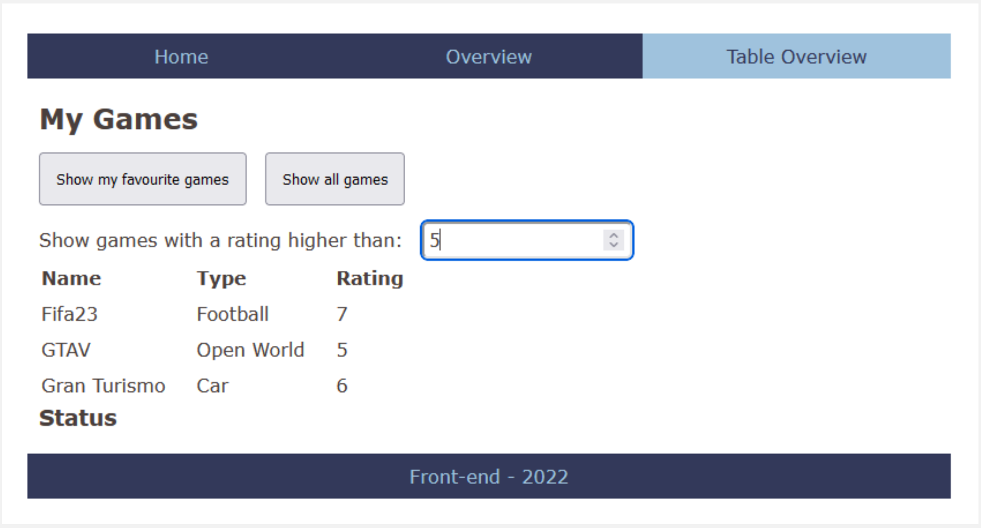
Name: Horizon Forbidden West - Type: Adventure - Rating: 3.5 - Favourite: false

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

- ❑ Added two button-elements in overview-table.html
- ❑ Added click-event on both buttons
- ❑ Added parameter “filterFunction” on the function renderGames
- ❑ Added filter on array: games.filter(()=>)
- ❑ When no filterFunction is passed to renderGames function, all games are shown

5.3.2 Exercise: Show games with rating larger than given value

Create input field in overview-table.html. When the user fills out some number, only games with a rating higher than the input value are shown. The values are shown in real time, i.e. while typing.



Name	Type	Rating
Fifa23	Football	7
GTAV	Open World	5
Gran Turismo	Car	6

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

- ❑ Added an input event listener to the rating input field. When the user types a new rating, call the renderGames function with a filterFunction that uses the entered rating.

6 JS: Async

Up to now, games are hardcoded in the js-file. When you finish this labo, games will be stored on 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 of the browser. This application behaves like an external webserver like for instance webontwerp.ucll.be. We will run the application locally. Then your computer behaves as a (local) webserver. Applications on your computer have access to this web server via http://localhost:3000. Other people do not have access to it.

Download The Games Library back-end (Toledo) and put it somewhere outside of you games working directory. Install NodeJS as explained in the file readme.md. In VS Code there is a nice preview for markdown files: open the .md file and look for an icon in the top right hand corner of your VS Code window ("open preview to the side").

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 "^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.

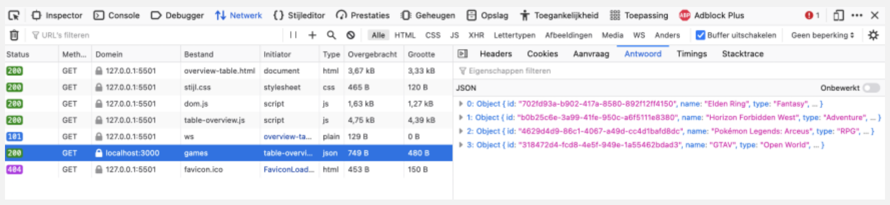
```
const fetchAndRenderAnimals = async () => {

    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...	Domein	Bestand	Initiator	Type	Overgebracht	Grootte
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
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
101	GET	127.0.0.1:5501	ws	overview-ta...	plain	129 B	0 B
200	GET	localhost:3000	games	table-overvi...	json	749 B	480 B
404	GET	127.0.0.1:5501	favicon.ico	FaviconLoad...	html	453 B	150 B

```
JSON
▸ 0: Object { id: "702fd93a-b902-417a-8580-892f12ff4150", name: "Elden Ring", type: "Fantasy", … }
▸ 1: Object { id: "b0b25c6e-3a99-41fe-950c-a6f5111e8380", name: "Horizon Forbidden West", type: "Adventure", … }
▸ 2: Object { id: "4629d4d9-86c1-4067-a49d-cc4d1bafd8dc", name: "Pokémon Legends: Arceus", type: "RPG", … }
▸ 3: Object { id: "318472d4-fcd8-4e5f-949e-1a55462bdad3", name: "GTAV", type: "Open World", … }
```

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 Search games by name

In Exercise 5.3.2 **Error! Reference source not found.**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.

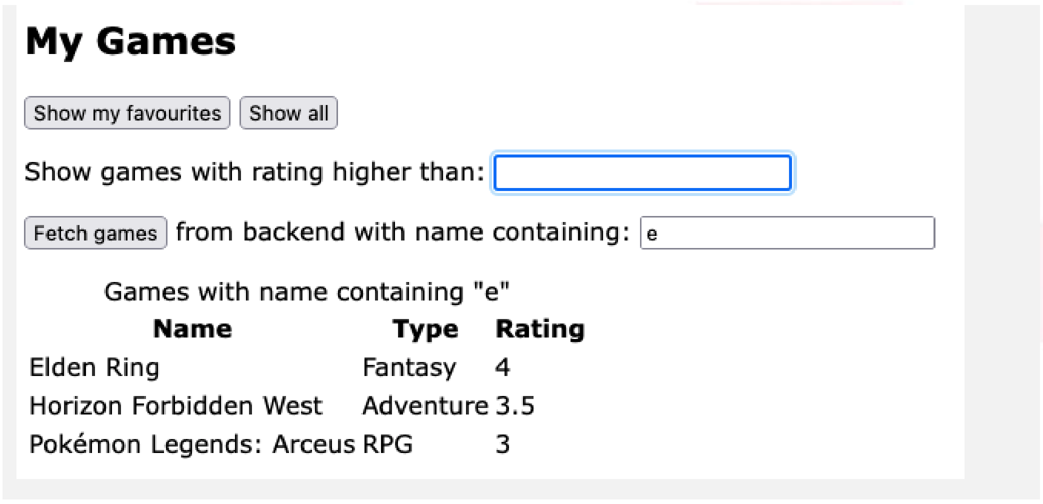
In this excercise you create a functionality that searches the data on the back-end and fetches only the filtered data from the server. You don't have to worry about the back-end: the necessary functions are already implemented. Try in the browser the following URL: http://localhost:3000/games?query=e.

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, search by name

Add a button "Fetch games" on the table-overview page. When the 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 the 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 games with rating higher than:

from backend with name containing: e

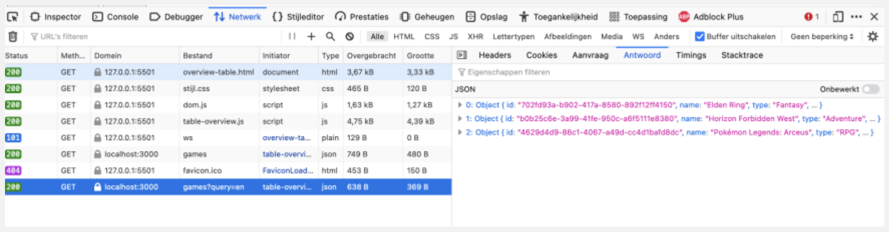
Name	Type	Rating
Elden Ring	Fantasy	4
Horizon Forbidden West	Adventure	3.5
Pokémon Legends: Arceus	RPG	3

Games with name containing "e"

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 the 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...	Domein	Bestand	Initiator	Type	Overgebracht	Grootte
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
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
101	GET	127.0.0.1:5501	ws	overview-ta...	plain	129 B	0 B
200	GET	localhost:3000	games	table-overvi...	json	749 B	480 B
404	GET	127.0.0.1:5501	favicon.ico	FaviconLoad...	html	453 B	150 B
200	GET	localhost:3000	games?query=en	table-overvi...	json	638 B	369 B

```
JSON
▸ 0: Object { id: "702fd93a-b902-417a-8580-892f12ff4150", name: "Elden Ring", type: "Fantasy", … }
▸ 1: Object { id: "b0b25c6e-3a99-41fe-950c-a6f5111e8380", name: "Horizon Forbidden West", type: "Adventure", … }
▸ 2: Object { id: "4629d4d9-86c1-4067-a49d-cc4d1bafd8dc", name: "Pokémon Legends: Arceus", type: "RPG", … }
```

6.4.2 Extra Exercises

1. Note that the functions "fetchAndRender()" and "searchByFetchAndRender()" do almost the same thing. Refactor both functions into one function that can handle both requests.

2. 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()”.

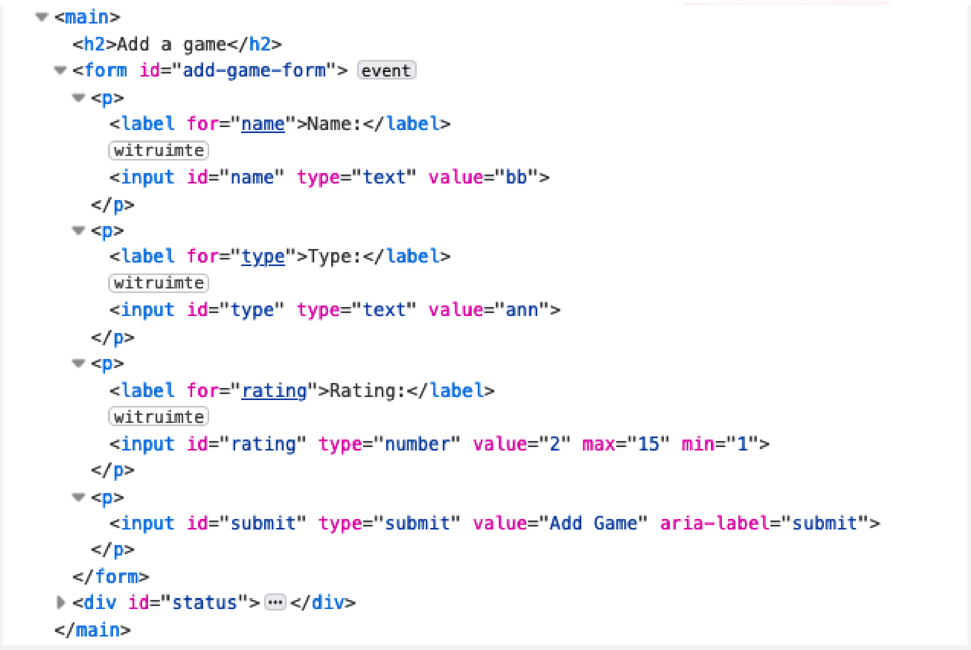
```
const addAnimal = async () => {

          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

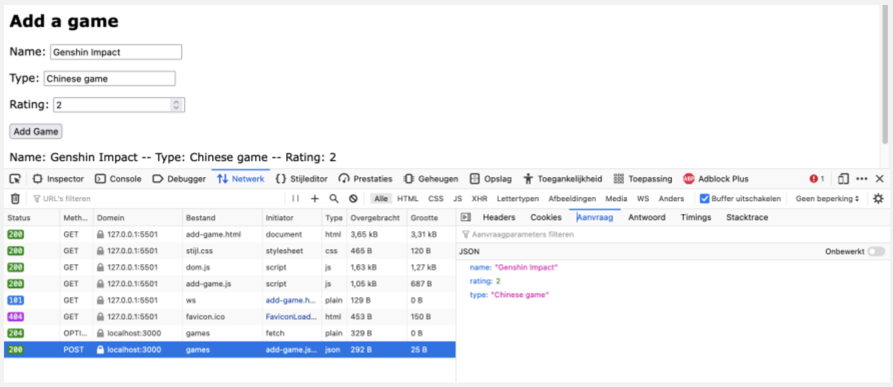
Create new files “add-game.html” and “add-games.js”.



```
▼<main>
   <h2>Add a game</h2>
  ▼<form id="add-game-form"> event
    ▼<p>
       <label for="name">Name:</label>
       witruimte
       <input id="name" type="text" value="bb">
     </p>
    ▼<p>
       <label for="type">Type:</label>
       witruimte
       <input id="type" type="text" value="ann">
     </p>
    ▼<p>
       <label for="rating">Rating:</label>
       witruimte
       <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:	Genshin Impact
Type:	Chinese game
Rating:	2

Add Game

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

Status	Meth...	Domein	Bestand	Initiator	Type	Overgebracht	Grootte
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
101	GET	127.0.0.1:5501	ws	add-game.h...	plain	129 B	0 B
404	GET	127.0.0.1:5501	favicon.ico	FaviconLoad...	html	453 B	150 B
204	OPTI...	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 add-games.js

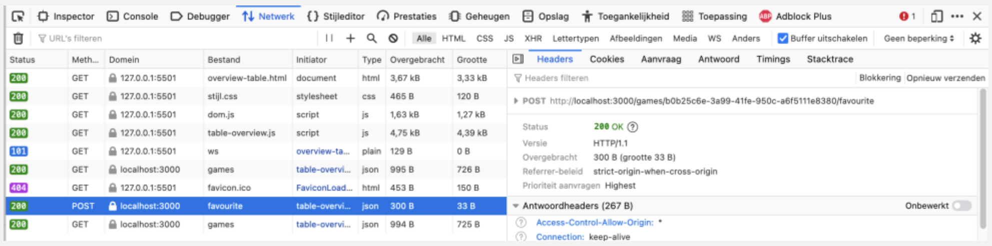
- ☐ Add submit-event to form. Use "preventDefault()" to avoid built-in behaviour of the HTML form element.

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



Status	Meth...	Domein	Bestand	Initiator	Type	Overgebracht	Grootte
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
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
101	GET	127.0.0.1:5501	ws	overview-ta...	plain	129 B	0 B
200	GET	localhost:3000	games	table-overvi...	json	995 B	726 B
404	GET	127.0.0.1:5501	favicon.ico	FaviconLoad...	html	453 B	150 B
200	POST	localhost:3000	favourite	table-overvi...	json	300 B	33 B
200	GET	localhost:3000	games	table-overvi...	json	994 B	725 B

```
POST http://localhost:3000/games/b0b25c6e-3a99-41fe-950c-a6f5111e8380/favourite

Status                200 OK
Versie                HTTP/1.1
Overgebracht          300 B (grootte 33 B)
Referrer-beleid       strict-origin-when-cross-origin
Prioriteit aanvragen  Highest

Antwoordheaders (267 B)                                  Onbewerkt
Access-Control-Allow-Origin: *
Connection: keep-alive
```

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

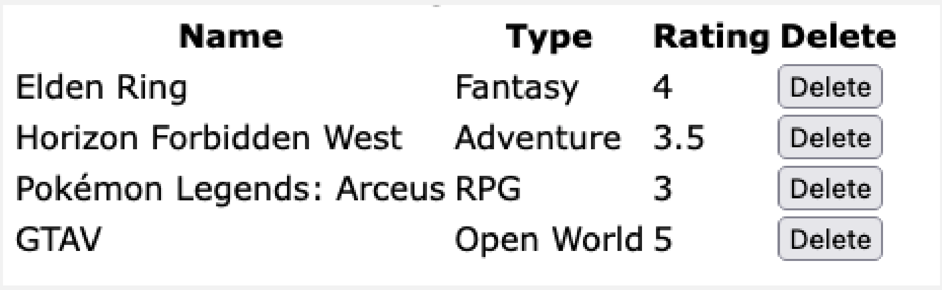
Up to now, your application sends POST- and GET-requests to the back-end. It can also send a DELETE-request. Use this to make it possible to delete a game.

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
GTAV	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
  - o fetches the given URL with the delete request. "uuid" is replaced by the id of game (${game.id})
  - o 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.

7 Error handling

Up till now, the assumption is: a perfect user handles perfect data without errors. However, the world is not perfect. What happens when the back-end has no data? What if the user submits invalid data? In those cases, the user has to be informed of what went wrong.

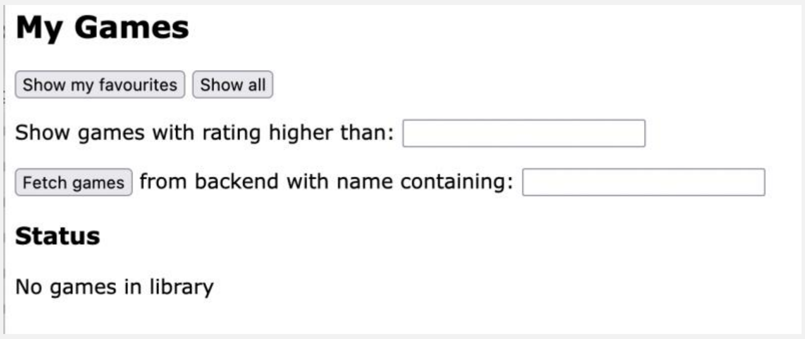
7.1 No data in the back-end

When there is no data in backend, or the search-by-fetch on name has no matching games, you could limit yourself to only show an empty table. However, the user does not now that the empty table is a consequence of the absence of data, or that it is caused by an internal error. Therefore, the table should be hidden (e.g. by the CSS property “display: none”) and an appropriate message should be shown.

7.1.1 Exercise: Hide the table and show a message when there is no data

When data is fetched, your code can monitor the received number of data. When there is no data, the table with the overview is hidden and feedback is shown in div#status.

E.g. when you delete all games, application should show following:



My Games
Show my favourites
Show all
Show games with rating higher than:
Fetch games
from backend with name containing:

Status

No games in library

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

- ❑ Write a function “hideTable({tableId}) that hides the table with given id by adding a classname “hidden”. This class sets the style of the given element to “display: none”;

- ❑ Refactor the function “renderAllGames()” such that it behaves as follows when there are no games in library:
  - Hides the table
  - Adds an error message to div#status
- ❑ Check the application: remove all data. The message should appear.

Add a new game. Is your application still working?

7.2 Input validation

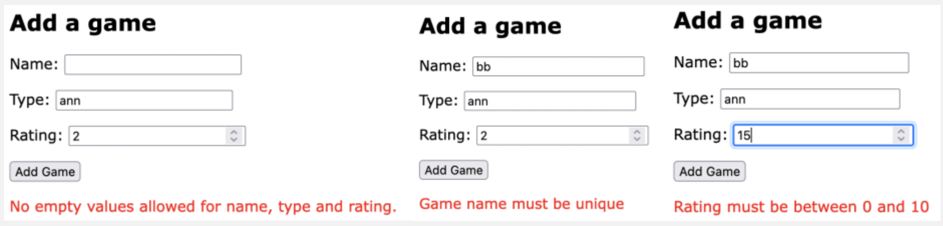
In add-game.js we can add validation to the form, e.g. to inform the user that one of the fields is missing. For some checks (e.g. empty field), HTML-validation can be used (with the attribute “required” for instance). However, as an exercise, we make our own customized input validation.

7.2.1 Exercise: add input validation

Show appropriate messages when

- ❑ One or more fields are empty
- ❑ The length of a name is invalid: 2<=length<=64
- ❑ The rating is not valid: 0 <= rating <= 10
- ❑ The name is not unique in the library.

For this last item (name nog unique): a GET-request to http://localhost:3000/games/name/:name returns the id of the game with given name and null otherwise.



Add a game

Name:

Type: ann

Rating: 2

Add Game

No empty values allowed for name, type and rating.

Add a game

Name: bb

Type: ann

Rating: 2

Add Game

Game name must be unique

Add a game

Name: bb

Type: ann

Rating: 15

Add Game

Rating must be between 0 and 10

Evaluation criteria: You should have updated in the file add-game.js

- ❑ Create function “addStatusError(status)”. This function clears the status message if necessary and adds a new message in red.

- ❑ Read input from fields
- ❑ *Before* you add the game to the backend: check validity of the input. Use if/else statements: when the first check is invalid, the status message is created and the function breaks down.
- ❑ Concerning the uniqueness of name:
    - Try in your browser the url http://localhost:3000/games/name/:name. Replace ":name" by an existing game name. Check the output.
    - Write a function "nameIsUnique({gameName}). This function returns the response of the request above as JavaScript.Do not forget to use async and await.
    - When "nameIsUnique(name) !== null", the backend has found a game with the given name. You can get the id of that game with "nameIsUnique(name).id".
    - Use this function in the if-statement in addGame(). Again, do not forget to use "await"!
- ❑ Finally, add a valid game. Be aware that the status error disappears.

8 Polling

Open your games application in two browser tabs. The first tab shows the overview, the second the add-form. Add a new game. Note that the new game is shown in the second tab (where you added one), but not in the first one unless you refresh the page.
This problem can be solved by polling. This technique triggers the browser to regularly fetch data from server.

8.1.1 Syntax

```
setInterval(fetchAndRenderAnimals, 1000);
```

The function setInterval() expects as first parameter a function that is called when a time interval is passed. That interval is the second parameter (in millisconds).

8.1.2 Exercise: Use polling to synchronize the games overview

Add polling to your application to synchronize the games overview.

- ❑ Open the app in two browser tabs. Add an extra game in one tab and check that the overview page in the other tab is refreshed automatically.
- ❑ At this point, do not care about filter and search functionalities

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

- ❑ Add setInterval(*function,*1000) at the end of the file. The function *function* is your rendering function.
- ❑ If necessary, clear tablebody to avoid repetition of table rows.

8.1.3 *Exercise: repair your search-by-name functionality (if it's broken)*

Refactor your code such that the search functionality (search by name) is working (exercise 6.4.1) (if it is not working).

Evaluation criteria: The solution depends on your program code. A possible solution is

- ❑ Do not call "setInterval()" on "fetchAndRender()", but on "searchByFetchAndRender()".

8.1.4 *Exercise: repair your 'filter on rating' (if it's broken)*

Now check the filter functionality on rating (exercise 5.3.2). If it is not working, refactor your code.

Evaluation criteria: The solution depends on your program code. A possible solution is

- ❑ Maybe you call in "searchByFetchAndRender()" the function "renderAllGames()"? In exercise 5.3.1 we suggested to pass a filter function as parameter to "renderAllGames()". Can you pass the function that filters the rating as a parameter when you call renderAllGames() in searchByFetchAndRender()?

8.1.5 *Exercise: repair 'show favourites' (if it's broken)*

Now check the button "show my favourites". This button should work in all cases: no filtering, filtering on rating, searching on name, combination of both. If it is not working, refactor your code.

Evaluation criteria:

The solution depends on your program code. The trick of the previous exercise is not working anymore. A possible solution is

- Introduce global parameter “showFavourites: Boolean” to your script.
- The click-events “show favourites”/”show all” toggle this Boolean
- Depending on the Boolean, renderAllGames() does a (second) filtering on favourites.

RESERVE

8.1.6 Exercise: Show the table and remove the message when data is present

Add again some games to your library.

Check your application with the “fetch games” search button. Note that the table disappears when searching gives no results, but that the table is not “renewed” when you submit another search query. The table style stays “display: none” and the error message does not disappear.

Solve this problem.

Extra: provide a special message when the empty table is caused by the search-button e.g. “There are no games with name containing abc”.

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

- ❑ Write a function “showTable({tableId}) that sets table style to display: table;
- ❑ Refactor the function “renderAllGames()” such that it behaves as follows when there are no games in library:
  - Makes the table visible
  - Removes the error message from div#status
- ❑ Check the application.