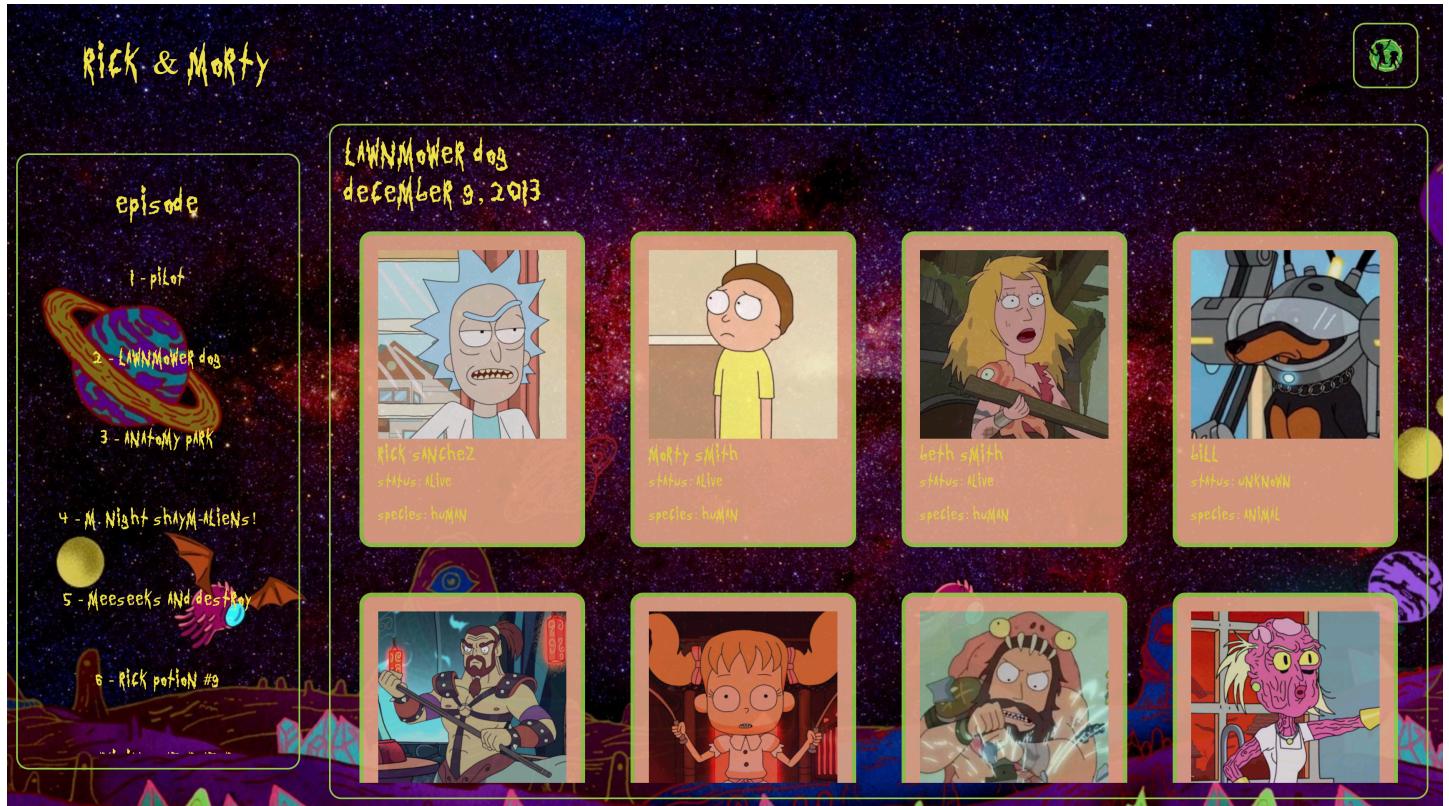
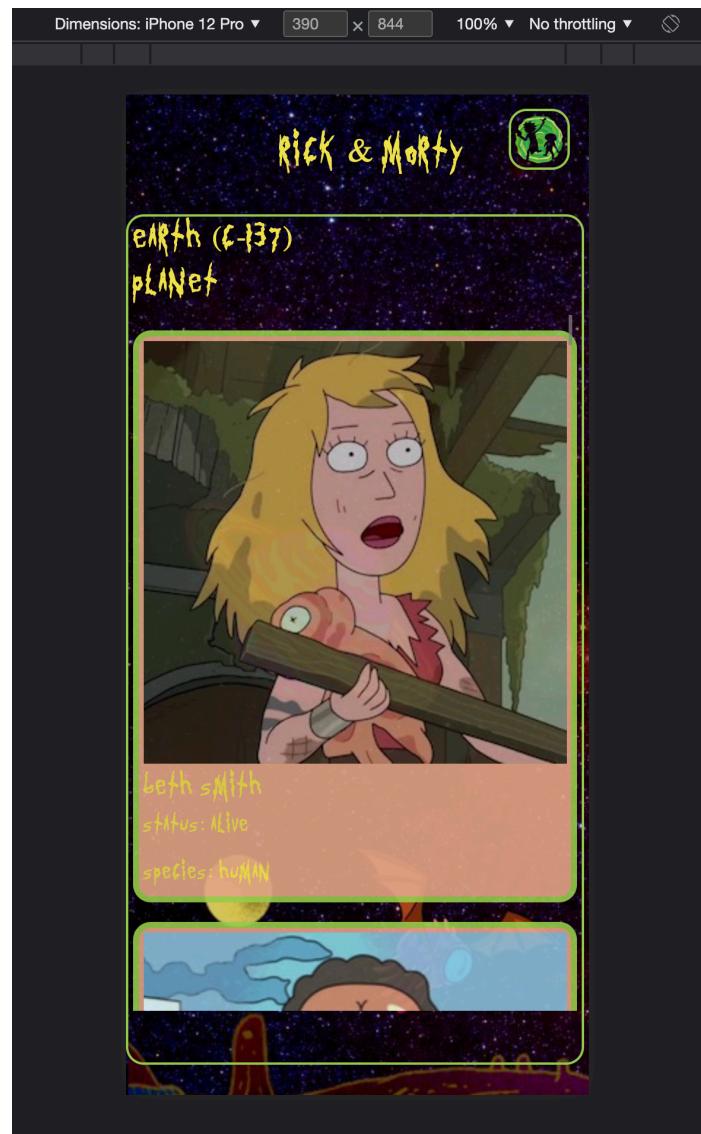
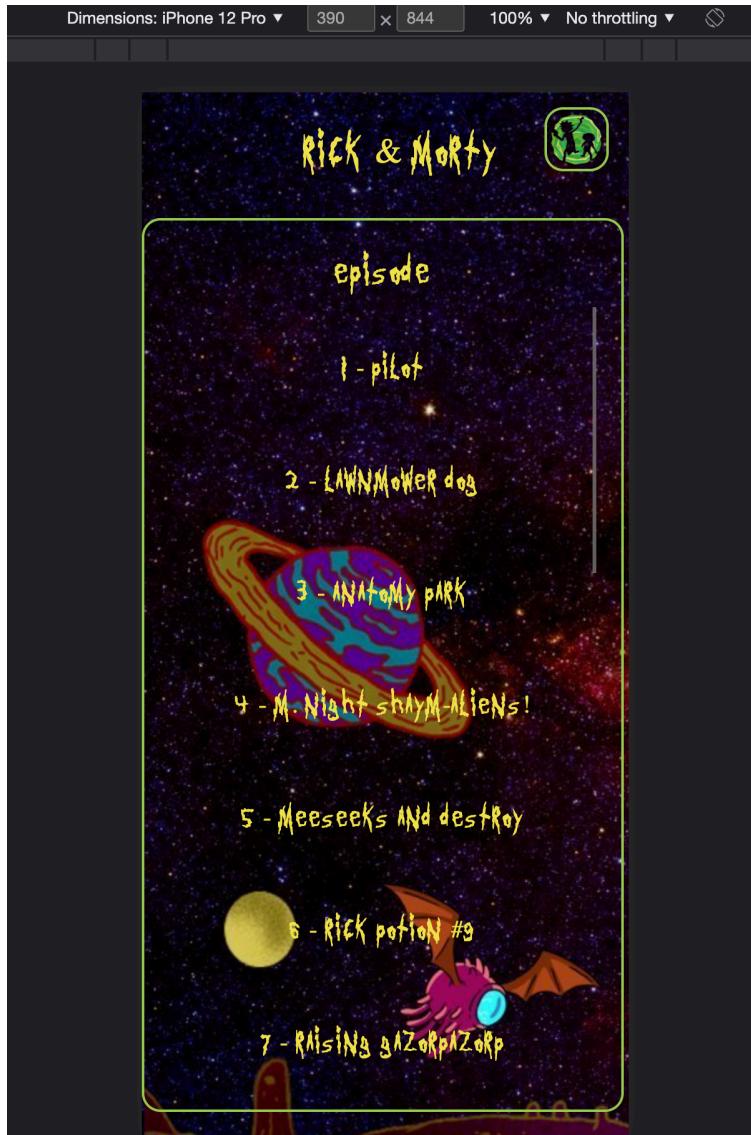


Rick & Morty

Here you have a small example of how the elements have been distributed through the web and the design that I have applied. To highlight I have used the palette of the series and the fonts.



Here we have the responsive view where it should be noted that the button located in the upper right corner would be used to remove the list with a collapsible and be able to access the rest of the data.

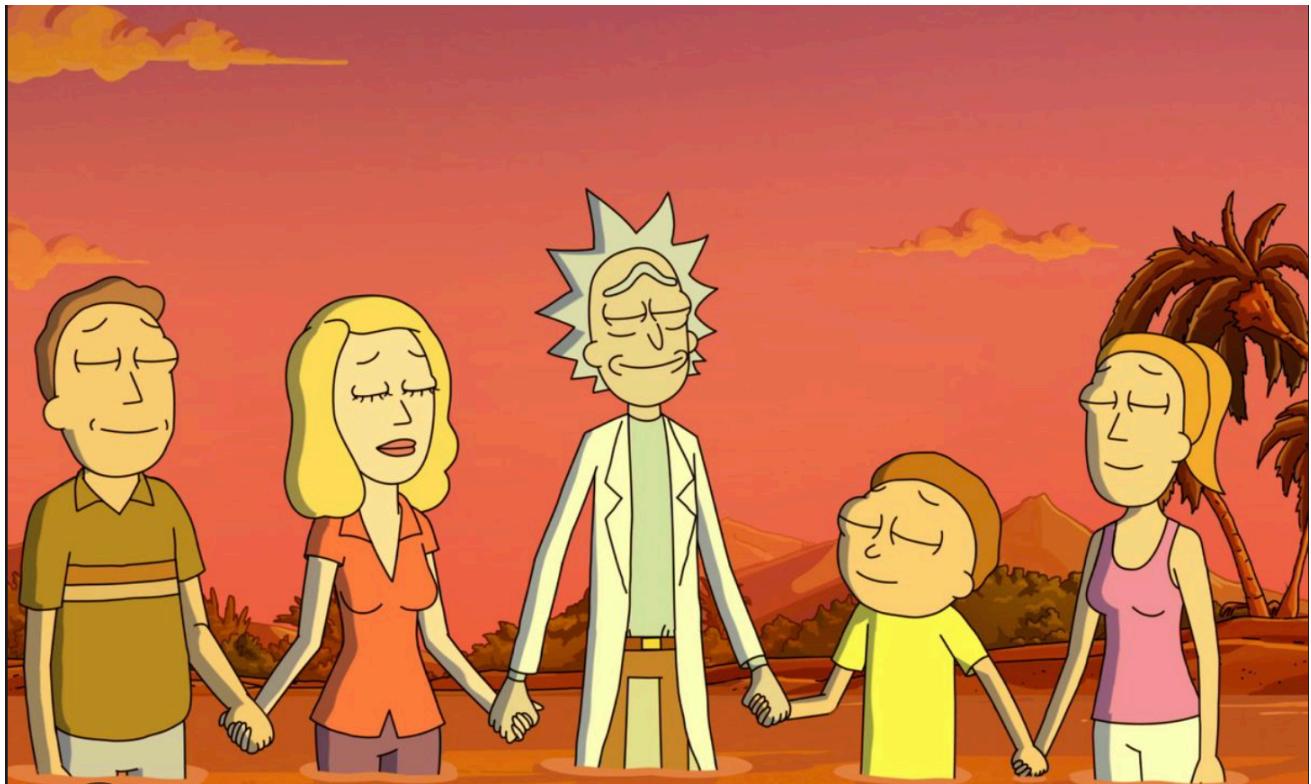


How to use the Web is simple, at first it automatically loads the list of episodes, through there we can start browsing through it.

In the list of episodes we can click on the name of the chapter for which we want to know the information, once the information section is open we see that we have a series of characters in which we can click on any part of the letter that contains them to be able to access the information of that specific user, then we can either choose to use the location of the character to show us the information of the place of origin where all the characters that make it up are shown or we can interact with the list of episodes in which said character appears, if we want we can always use the list that we had used at the beginning to go to a new chapter.

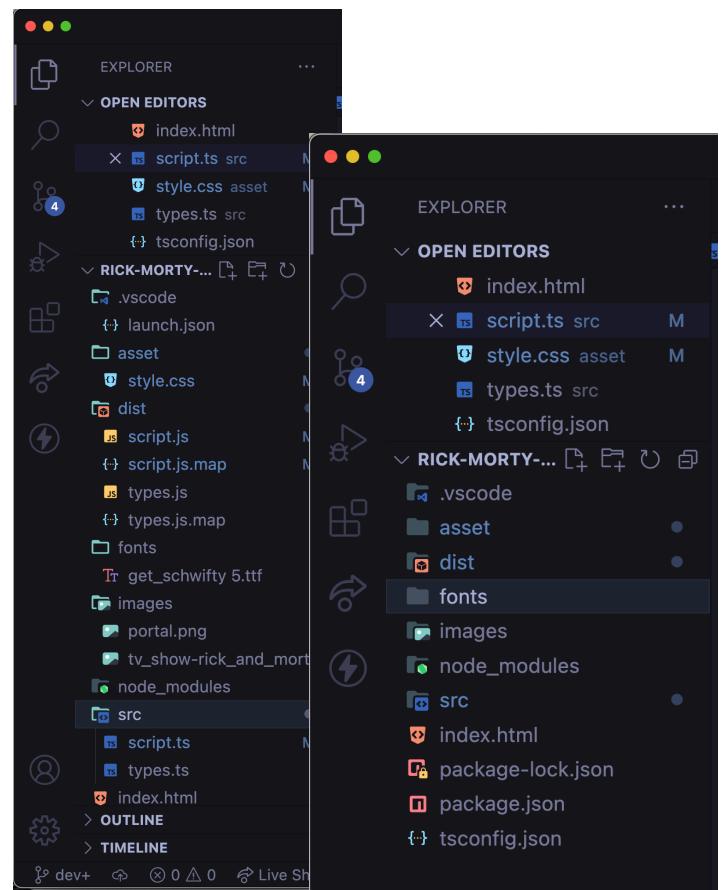
With this project I have learned how to interact with an API, to understand how promises work and the use of Bootstrap, I have practiced a lot with the Dom and I especially liked the Typescript part because it helps me to understand more how JS works, in general I have learned a lot in this last week and I have managed to gain more confidence in addition to the aforementioned knowledge.

Regarding the incidents found throughout the project, it is worth highlighting not having fully decided what I wanted to show or how to do it, in the end I lost time undoing work already done, some incidents in the code such as some problem understanding how bring the data to be able to use them later but I solved it relatively quickly and well, for the rest I have found myself very motivated throughout the project.



The tools that I have used have been, Html, Css, Bootstrap JS, Typescript.

I started by creating the folders that I was going to need during the project. Throughout the project I was creating the necessary files.



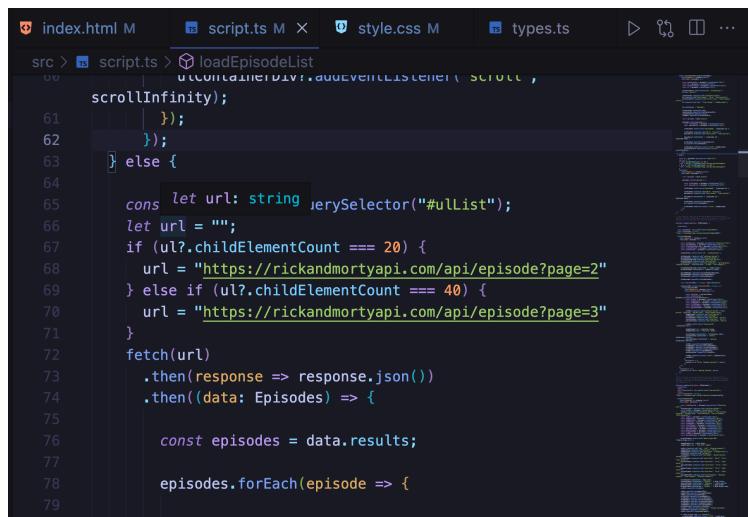
```
18 </head>
19
20 <body class="body-wallpaper font-style">
21   <header>
22     <nav class="navbar navbar-expand-lg navbar-dark bg-transparent">
23       <div class="container-fluid">
24         <h1 class="text-yellow fs-1 text center-title center-title-respons mt-4">Rick & Morty</h1>
25         <button class="button-style container-border" type="button" data-bs-toggle="collapse" data-bs-target="#collapse-side-bar" aria-expanded="false" aria-controls="collapse-side-bar">
26           
27         </button>
28       </div>
29     </nav>
30   </header>
31
32   <!--The content is divided into two sections, in the first the sidebar will be created and in the second the different information will be created.-->
33
34   <main class="flex-display display-respons align-items-center"
35     </main>
```

The code in the editor is the HTML for the 'index.html' file. It starts with a closing head tag, followed by a body tag with a 'body-wallpaper' and 'font-style' class. Inside the body, there's a header section with a navigation bar ('navbar') that has a dark background ('dark') and is transparent ('bg-transparent'). The navigation bar is expanded ('expand-lg') and contains a container fluid ('container-fluid'). Inside this container, there's a h1 tag with a yellow text color ('text-yellow') and a font size of 1 ('fs-1'). The text is centered ('center-title') and responsive ('center-title-respons'). Below the h1 is a button ('button-style') with a 'container-border' class. The button uses the 'collapse' data attribute to target a '#collapse-side-bar' element. It also has 'aria-expanded' and 'aria-controls' attributes. An image ('img') with a 'size-button' class is placed inside the button, and its source is 'images/portal.png'. The main content area ('main') is styled with 'flex-display', 'display-respons', and 'align-items-center'.

I decided that the static parts of my website were going to be made up of a Header, which would be made up of a Nav and a Main made up of two Sections, one of them wrapped in a Div to be able to apply a Collapse to make it easier for me to be responsive.

I have used Fetch to make the requests to the API.

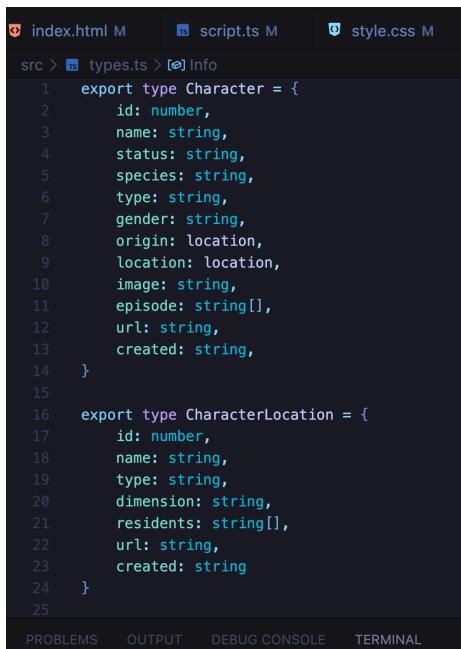
```
fetch(`${urlEpisodes}?page=${nextPage}`)
  .then(response => response.json())
  .then((data: Episodes) => {
```



A screenshot of a code editor showing a scroll event listener in a file named `script.ts`. The code uses the `scroll` event to trigger a fetch request for episode data. The URL for the fetch changes based on the scroll position, specifically checking if the `ul` element has 20 or 40 children to determine the page number.

```
src > script.ts > loadEpisodeList
  scrollInfinity);
  });
}
} else {
  const url: string = querySelector("#ulList");
  let url = "";
  if (ul7.childElementCount === 20) {
    url = "https://rickandmortyapi.com/api/episode?page=2"
  } else if (ul7.childElementCount === 40) {
    url = "https://rickandmortyapi.com/api/episode?page=3"
  }
  fetch(url)
    .then(response => response.json())
    .then((data: Episodes) => {
      const episodes = data.results;
      episodes.forEach(episode => {
```

Here I show the first one that I faced, I had to bring the URL of each page in order to work with that data.



A screenshot of a code editor showing type definitions in a file named `types.ts`. It defines two export types: `Character` and `CharacterLocation`, both containing properties like id, name, status, species, type, gender, origin, location, and image.

```
src > types.ts > [e] Info
1  export type Character = {
2    id: number,
3    name: string,
4    status: string,
5    species: string,
6    type: string,
7    gender: string,
8    origin: location,
9    location: location,
10   image: string,
11   episode: string[],
12   url: string,
13   created: string,
14 }
15
16 export type CharacterLocation = {
17   id: number,
18   name: string,
19   type: string,
20   dimension: string,
21   residents: string[],
22   url: string,
23   created: string
24 }
25
```

I've used types to check the data type of any taken value before it can be used as an input in code.

I created the necessary elements through the DOM, such as the sections, the cards where the characters will be displayed, some of them had to iterate through a loop in order to create the necessary content to display.



The screenshot shows a code editor with several tabs at the top: index.html M, script.ts M X, style.css M, types.ts, and tsconfig.json. The script.ts tab is active, showing the following code:

```
index.html M script.ts M X style.css M types.ts tsconfig.json
src > script.ts > loadEpisodeList > then() callback

10 let nextPage = 1;
11 const urlEpisodes = "https://rickandmortyapi.com/api/episode";
12 const sideBar = document.querySelector("#side-bar");
13 const ul = document.querySelector("#ulList");

14 if (ul?.childElementCount === 51) return
15 if (sideBar?.childElementCount === 0) {

16     fetch(` ${urlEpisodes} ?page=${nextPage}` )
17         .then(response => response.json())
18         .then((data: Episodes) => {
19
20             const containerDiv = document.createElement("div");
21             const h2 = document.createElement("h2");
22             const ulContainerDiv = document.createElement("div");
23             const ul = document.createElement("ul");
24
25             ulContainerDiv.setAttribute("id", "ulContainer");
26             ul.id = "ulList";
27
28             containerDiv.classList.add("container-fluid");
29             h2.classList.add("text-center", "mt-4", "text-yellow");
30             ulContainerDiv.classList.add("scroll-size", "size-respons-scroll");
31             ul.classList.add("nav", "flex-column", "sidebar-menu");
32
33             h2.textContent = "Episode";
34
35             containerDiv.appendChild(h2);
36             containerDiv.appendChild(ulContainerDiv);
37             ulContainerDiv.appendChild(ul);
38             sideBar?.appendChild(containerDiv);

39             const episodes = data.results;
40
41             episodes.forEach(episode => {
42                 const listElement = document.createElement("li");
43
44             });
45         });
46     
```

At the bottom right of the editor window, there is a status bar with the text: Screen Reader Optimized Ln 61, Col 12 Spaces: 2

The work has been distributed in several functions, there are two that serve one to make the infinite scroll and another to clean the section where the information is displayed, on the other hand we have four more functions.

The three make a request to the API and create all the necessary elements to shape the user interface, each one creates the elements that it will need, beyond this we have a Css file to finish giving it styles and supporting the work. from Bootstrap.