

RICK & MORTY API

ALI HOURAG KHALI-AISSA

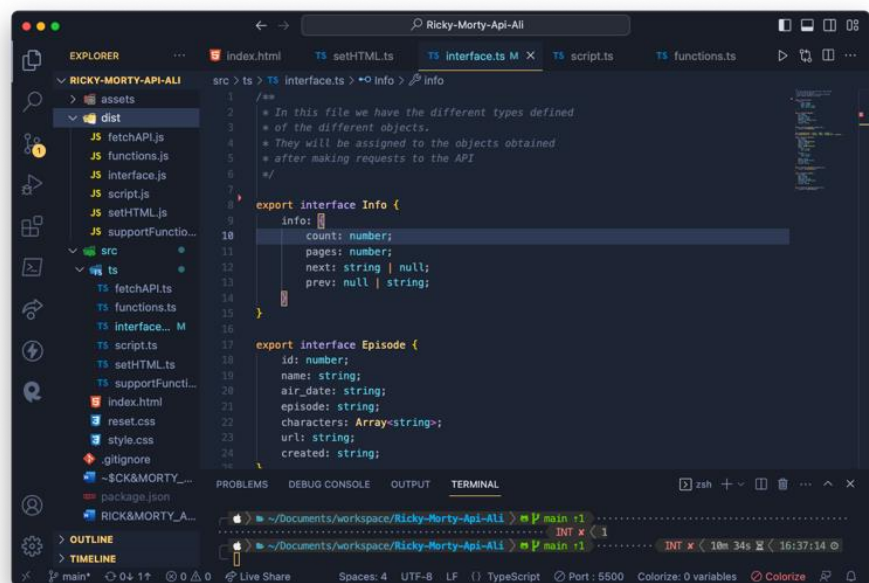
ORGANIZATION

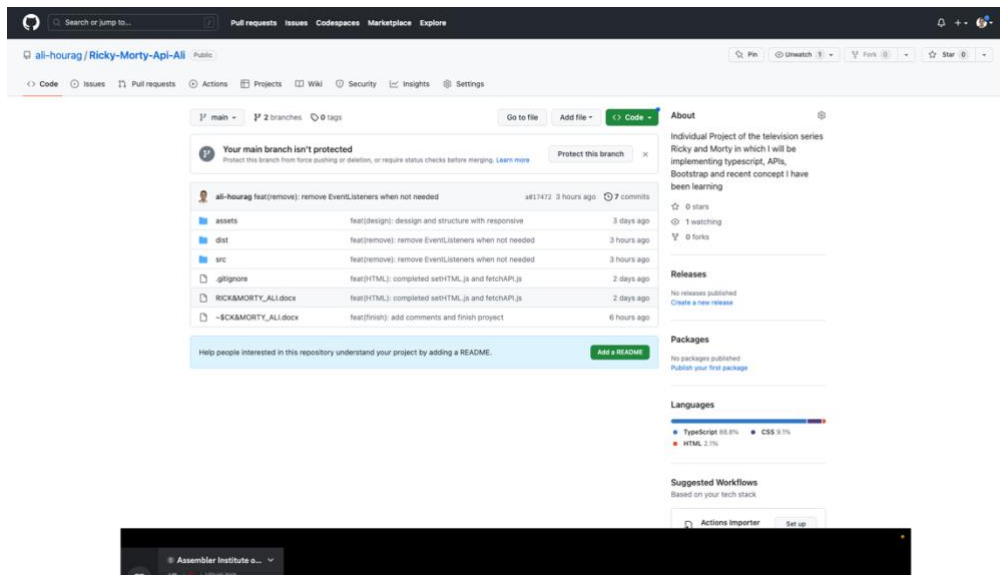
To organize this project different tools have been used:

- TRELLO to have a backlog of ideas, what I am currently working on, what I will do if I have time, completed functionalities and so on.
- This word document (pdf) to keep a worklog of what has been done during each day, issues that have occurred as well as GitHub to have my project in a remote Repository.
- How I planned the project and steps taken as well as how I have used typescript, bootstrap, APIs request, files organization and so on.

SET UP

First of all, I had to create a Git repository, set the first folders and files with typescript and html, configure tsconfig.json and set everything up to be able to compile. I set the file of interface.ts with the types of objects of Episode, Character and Location which will be exported.



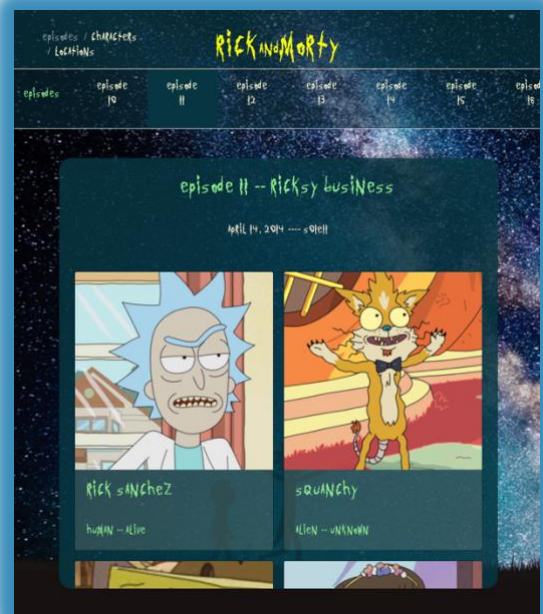
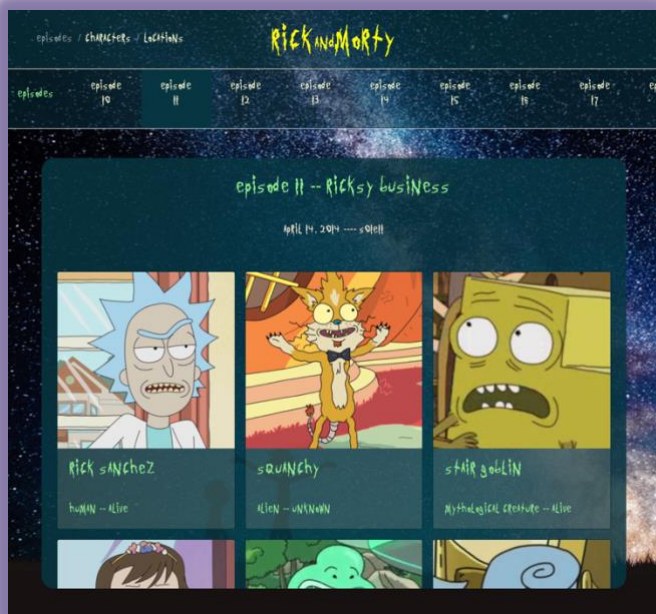


After creating my first commits and branches, I start developing my project.

DESIGN

- With regards to the design. First I will be developing a desktop first and then translate it to mobile since for this particular case I see it as the option to go.
- There are different section with characters, episodes and locations as you can see.





LOCATIONS SECTION



EPISODES SECTION



LOGIC

-I have organized the project's folder in a way in which if a new programmer opens it for the first time, it would not be able to find a way out.

This is due to the fact of having, apart from regular index.html and script.css, a general script.ts which is connected to the other typescript files which have a value inner of itself.

I have the next files: setHTML.ts, functions.ts, supportFunctions.ts, fetchAPI.ts and interface.ts.

This results in an enhanced legibility, efficiency and maintainability of the code.

TYPESCRIPT PROBLEMS

While writing the code, I encountered different problems due to its extremely typed nature.

I understood it right away. However, I did have some problems with the interaction of the typescript with the DOM, which resulted on type problems since some could be none.

The result to this is just study when the return is null, if it is null, then make an empty return that will be studied in the future or the present.

```
//2 next read the episode section since the structure is very similar
const episodeSection: (HTMLElement | null) = document.querySelector(".episode-info");
const characterSection: (HTMLElement | null) = document.querySelector(".character-info");
const h3Location: (HTMLHeadingElement | null) = document.querySelector(".h3-episode");
const pLocation: (HTMLParagraphElement | null) = document.querySelector(".p-info-episode");
let urlLocation: string | null;
if (episodeSection === null) return;
if (characterSection === null) return;
if (h3Location === null) return;
if (pLocation === null) return;
```

I have tried to set a type to everything since that is the essence of typescript.

RECORD OF ISSUES

- 1.- Separating files in a logical and efficient manner.
- 2.- Handling the asynchronous code efficiently, since being the first project using an API and its data to make a responsive and dynamic web page.

```
async function pageLoaded(): Promise<void> {  
    //Clear session storage from possible data entered before.  
    //It will be needed if the page is refreshed and data has been stored in session storage  
    sessionStorage.clear();  
    //Set HTML  
    setHeader();  
    //Add episodes to the episodes bar after getting the fetch of episodes  
    const episodes: Episodes = await getEpisodes();  
    addEpisodes(episodes.results);  
  
    //Event listener to the sidebar to make it infinite  
    const episodesContainer: (HTMLDivElement | null) = document.querySelector(".episodes-bar-container");  
    if (episodesContainer === null) return;  
    episodesContainer.addEventListener("scroll", infiniteScrollSB);  
}
```

- 3.- Modularizing function so that it is easier to read and understand. If a functionality is done twice, then subprogram it. Legibility and maintainability is heavily improved, since functions tend to be big and complicated.

```
9      * part of the web we are situated  
10     * @param activeItem receives a number indicating where we are  
11     * if activeItem === 0, then class active will be added to episodes  
12     * else if activeItem === 1, then class active will be added to characters  
13     * else if activeItem === 2, then class active will be added to locations  
14     */  
15     export function setBreadcrumb(activeItem: (0 | 1 | 2)): void {  
16         const breadcrumbItems: NodeListOfHTMLElement = document.querySelectorAll(".breadcrumb-item");  
17         breadcrumbItems.forEach((breadcrumbItem: HTMLElement): void => {  
18             if (breadcrumbItem.classList.contains("active")) breadcrumbItem.classList.remove("active");  
19         });  
20         breadcrumbItems[activeItem].classList.add("active");  
21     }  
22     //-----  
23     /**  
24     * If an episode is clicked, a class named active is added which gives a stronger background  
25     * so that it is highlighted.  
26     * This function removes that if any episode in the sidebar has it  
27     */  
28     export function removeEpisodeSelected(): void {  
29         const episodesSideBar: NodeListOfHTMLAnchorElement = document.querySelectorAll(".episode-link");  
30         episodesSideBar.forEach((episode: HTMLAnchorElement): void => {  
31             if (episode.classList.contains("active")) episode.classList.remove("active");  
32         })  
33     }  
34     //-----
```


4.- Remove eventListener if they are no longer going to be used to free up space in memory and make the web more efficient. This gave problems of efficiency and applying the right event.

```
//  
/**  
 * Removes eventListener from scroll when it is no longer needed.  
 */  
function removeEventListenerScroll() {  
  const episodesContainer: (HTMLDivElement | null) = document.querySelector(".episodes-bar-container");  
  if (episodesContainer === null) return;  
  episodesContainer.removeEventListener("scroll", infiniteScrollSB);  
}  
//  
/**  
 * Removes eventListener from cards of episodes when it is no longer needed.  
 */  
export function removeEventListenersCardsEpisodes() {  
  const cardsEpisodesContainers: NodeListOf<HTMLDivElement> = document.querySelectorAll(".episode-card-character-selected");  
  cardsEpisodesContainers.forEach(cardEpisode => cardEpisode.removeEventListener("click", episodeClicked));  
}  
//  
/**  
 * Removes eventListener from cards of characters when it is no longer needed.  
 */  
export function removeCardCharacterContainerEventListener() {  
  const cardCharacterContainer: NodeListOf<HTMLDivElement> = document.querySelectorAll(".card-character-container");  
  cardCharacterContainer.forEach(cardCharacter => cardCharacter.removeEventListener("click", characterClicked));  
}
```

5.- Apply infinity to the scrollbar since using a button is not very pleasing nor efficient. It was hard to apply because while moving the scroll the EventListener is called constantly having a great probability of calling it more times than needed. The solution was to use sessionStorage and a variable called endScroll to control where the scrollbar in the container.

```
export async function infiniteScrollSB(): Promise<void> {  
  const episodesContainer = document.querySelector(".episodes-bar-container") as HTMLElement;  
  
  //scrollWidth: actual width of the content  
  //clientWidth: actual width of the part of the container that can be seen  
  //scrollLeft: width of the part of the content that has been scrolled  
  //if the scroll reaches the end, then scrollWidth = clientWidth + scrollLeft  
  //offsetWidth is not being used since there is no border or padding.  
  if (episodesContainer.scrollWidth - episodesContainer.clientWidth < episodesContainer.scrollLeft + 2) {  
  
    //The endScroll item controls that this change is not done constantly if the user is scrolling too fast.  
    //End scroll is put on false when all the data is fetched and showed. When the data is being fetched or  
    //showed, then endScroll is put on true and it won't enter this function, and hence, not make changes.  
    //It is on true as well when all the episodes have been loaded and there is nothing more to make a request to.  
    if ([sessionStorage.getItem("endScroll")] === "false") {  
      sessionStorage.setItem("endScroll", "true"); //Set it on true to prevent scrolling while adding elements  
      let actualPage: (string | null) = sessionStorage.getItem("page");  
      if (actualPage === null) return;  
      //Fetch of the respective page of the API  
      let pageLoaded: number = parseInt(actualPage) + 1;  
      if (pageLoaded === 3) removeEventListenerScroll();  
      const episodes: Episodes = await getEpisodes(`?page=${pageLoaded.toString()}`);  
      addEpisodes(episodes.results); //Add it to the sidebar  
    }  
  }  
}
```

USE OF API

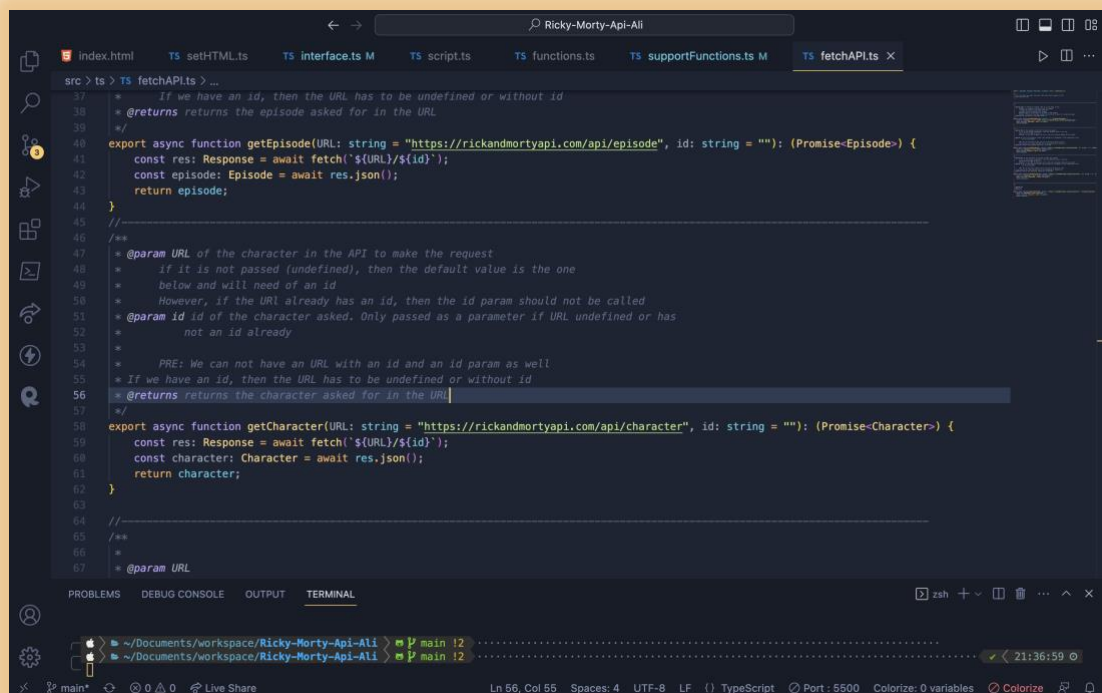
-In the API documentation we have different methods to get episodes, characters or locations.

-If we want to access a specific one, we have to apply an id like this:

<https://rickandmortyapi.com/api/character/1>

I have used as well ?page=1 ,2 or 3 to get different paginations of Episodes adding it to the URL. However there are many more additions to map by name, origin, location...

In my project specifically, I have created a file named fetchAPI.ts which has functions that makes requests to the API.



```
src > ts > TS fetchAPI.ts > ...
37 * If we have an id, then the URL has to be undefined or without id
38 * @returns returns the episode asked for in the URL
39 */
40 export async function getEpisode(URL: string = "https://rickandmortyapi.com/api/episode", id: string = ""): (Promise-Episode) {
41   const res: Response = await fetch(`${URL}/${id}`);
42   const episode: Episode = await res.json();
43   return episode;
44 }
45 //-----
46 /**
47  * @param URL of the character in the API to make the request
48  * If it is not passed (undefined), then the default value is the one
49  * below and will need of an id
50  * However, if the URL already has an id, then the id param should not be called
51  * @param id id of the character asked. Only passed as a parameter if URL undefined or has
52  * not an id already
53  *
54  * PRE: We can not have an URL with an id and an id param as well
55  * If we have an id, then the URL has to be undefined or without id
56  * @returns returns the character asked for in the URL
57  */
58 export async function getCharacter(URL: string = "https://rickandmortyapi.com/api/character", id: string = ""): (Promise-Character) {
59   const res: Response = await fetch(`${URL}/${id}`);
60   const character: Character = await res.json();
61   return character;
62 }
63 //-----
64 /**
65  *
66  *
67  * @param URL
```

LESSONS LEARNED

- Usage of npm to create dependencies.
 - Making asynchronous code with async and wait.
 - Making requests to an API using fetch.
 - Developing in a strongly typed programming language.
 - Organization of a project files and how to divide it logically.
 - Usage of css libraries of Bootstrap.
-

WORKLOG

Worklog: 12 May→ This day I have set everything up, started the design and already planned about how to organize everything, its styles, logic and so on. Tomorrow I will dedicate it to finish the entire design with responsive.

Worklog: 15 May->This day I have finished the design completely, started with the logic and I have already made a plan on how to go about it. Now it's time to start with the most challenging part!

Worklog: 16May -> Completely created HTML file from ts and have set up the top scrollbar, which keeps adding episodes with infinite scrollbar until all of them are set. Also perfected the fetchAPI.ts file as well as the interface.ts since there were some things which could cause potential errors.

Started functions.ts which has the function that make the page dynamic and make calls to the fetch.

Worklog: 17May -> Finished everything, with the extras. Commented and organized functions and files. Interaction with the web page working perfectly fine.

Worklog: 18 May -> Removed unnecessary eventListeners. Retyped everything since when there was the possibility of null I put the typecasting with as at the end and that was wrong, since it acts as an any which is javascript basically. With the help of Luis, created the supportFunctions.ts file to increase legibility and created two enum types in the interface.ts for the character interface.