



# **Universidad Nacional de General Sarmiento**

## **Proyecto Rick & Morty**

Materia: Introducción a la programación

Alumnos: Franco Pachao, Mateo Cuellar, Francisco Diaz y Sebastián Carrasco

Docentes: Santiago Montiel, Gonzalo Godoy, Jorgelina Rial y Yair Ruiz

Comisión: Virtual


## Introducción

Este proyecto se enfoca en el desarrollo de una galería de personajes del universo de "Rick y Morty". La idea es mostrar en una interfaz visual las imágenes de los personajes junto con los detalles importantes de cada uno, tales como su estado de vida, última ubicación conocida y el primer episodio en el que aparecieron. Además, se emplea un sistema visual que permite identificar rápidamente el estado de cada personaje mediante un borde de color en las tarjetas de presentación: verde para aquellos que se encuentran vivos, rojo para los que se encuentran fallecidos y naranja para aquellos cuyo estado es desconocido. Para obtener los datos e imágenes de los personajes, se utiliza una API pública que facilita la extracción de información en tiempo real, lo que permite que la galería se mantenga actualizada sin tener que intervenir nosotros. Se han implementado varias funciones para gestionar y transformar esta información, permitiendo que la aplicación presente los datos de forma organizada y efectiva.

## Códigos que implementamos

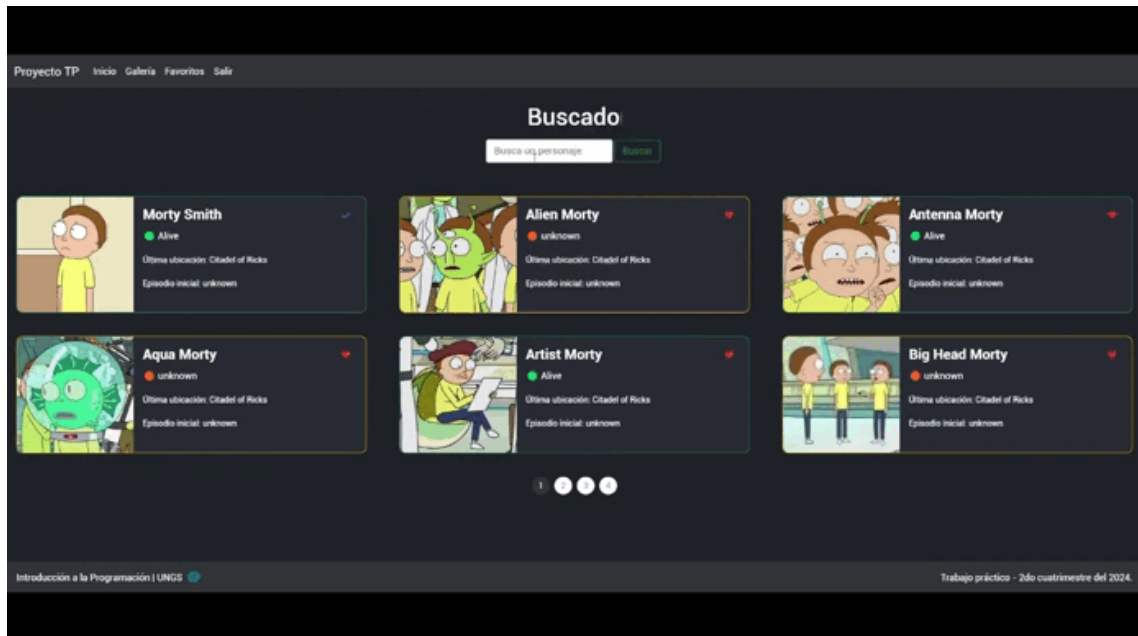
### 1. Para ver lo siguiente:



En la función de `getAllImages(input=None)`, de `services.py`, implementamos (`json_collection = transport.getAllImages()`) y recorremos cada elemento y la agregamos a la lista de images. En la función de `home(request)`, de `views.py`, implementamos (`images = services.getAllImages()`) para obtener las imágenes de services. Para ver si el personaje está vivo, muerto, o no sabemos, usamos condicionales de Django: (`{% if img.status == 'Alive' %}` , etc.) y para ver el color de su borde, dependiendo su estado, usamos Bootstrap (`{% if img.status == 'Alive' %}border-success{% elif img.status ==`

'Dead'%}border-danger, etc). En general estas implementaciones de código no se nos dificultó mucho.

2. Para hacer funcionar el buscador, en la función de `search(request)`, de `views.py`, implementamos (`images = services.getAllImages(search_msg)`) para filtrar lo que se escribe en la barra de búsqueda, para traer las imágenes desde `services.py` y así, después lo renderiza en el template `home.html`. Por lo general, implementar esto no se nos hizo difícil.
3. Para que funcione el inicio y cierre de sesión de una cuenta, en este caso `admin/admin`, en `login.html` agregamos ("`{% url 'login' %}`") para iniciar sesión. Y en el `header.html` utilizamos ("`{% url 'logout' %}`") para que, si el usuario inició sesión, pueda cerrar su sesión. Esta implementación se nos dificultó un poco, ya que había que buscar la manera de que funcione correctamente.
4. Para que funcione el querer guardar y eliminar personajes favoritos tuvimos que modificar algunas funciones en `services.py`, como por ejemplo en la función (`saveFavourite(request)`) y (`getAllFavourites(request)`). Además, modificamos funciones en `views.py`, como, por ejemplo, (`saveFavourite(request)`) y (`deleteFavourite(request)`), donde si el usuario está logueado, puede guardar o eliminar su personaje favorito.
5. Implementamos una paginación de Django, importándolo en `views.py`, donde en cada página se va a mostrar 6 personajes, esto lo decidimos con los compañeros del grupo ya que por default la API nos trae 20 imágenes de personajes, y estas se ven desordenadas, por esa razón la restringimos a 6 para que se vean ordenadas, implementamos (`image_paginator = Paginator(images, 6)`) en `views.py`. Además, creamos un path en `urls.py`, así cuando vaya pasando página por página, en la url se va mostrando en qué número de la página estamos, como por ejemplo ("`...home/2/...`").
6. Renovamos la interfaz gráfica convirtiendo la página a “modo oscuro” para que sea más agradable a la vista de quien lo visita, para renovarla modificamos el archivo `style.css`.



También renovamos algunos aspectos de la página como el inicio de sesión, creando un archivo CSS llamado login.css en la carpeta static.

7. Desarrollamos un *loading spinner* que simula la carga de las imágenes antes de redirigir al usuario a la galería. Lo realizamos con CSS y una función de JavaScript, con un div con un `style display="none"` para no mostrarlo en pantalla y una vez que se clickea el link para ver la galería se cambia el style a `display="block"` para que se muestre por unos segundos. Decidimos hacerlo de esta manera ya que era la más sencilla y eficiente.
8. Implementamos la opción de poner un comentario cuando se agrega a favoritos a un personaje. Lo realizamos con una función de JavaScript con un `onclick="comentario()"` en el botón de agregar a favoritos, se visualiza un `prompt()` y se guarda el comentario ingresado por el usuario en la base de datos, posteriormente se visualiza en la tabla de favoritos en la columna de "comentario".