

# **Introducción a la Programación**



## **Informe Trabajo Práctico WEB Api Rick y Morty**

**Profesoras: Rial, Jorgelina. Varela, Viviana.**

**Grupo 1; Comisión 8**

**Integrantes: Castro, Lucas. Timiraos, Mateo.  
Polcaro, Lucio. Contreras, Nicolas. Olmedo,  
Lautaro**

**2024**

En este informe se mostrará y explicará cada parte del código desarrollado para que el buscador web que utiliza una API de Rick y Morty funcione correctamente.

Cada parte del código realizado será mostrado bajo una captura del mismo y una explicación no tan formal. Se busca que el lector pueda entender sin la necesidad de **saber** tanta programación. Además, se expondrá la lógica detrás de cada código para su fácil interpretación.

Se espera que al final de la lectura, el lector pueda entender o aprender parte del desarrollo de una web que utiliza una API

## 1. Views.py

```
# esta función obtiene 2 listados que corresponden a las imágenes de la API y los favoritos del usuario, y los usa para dibujar el correspondiente template.
# si el opcional de favoritos no está desarrollado, devuelve un listado vacío.
def home(request):
    images = services.getAllImages() #Trae un lista con todos los personajes al cargar la galeria
    favourite_list = services.getAllFavourites(request)

    return render(request, 'home.html', { 'images': images, 'favourite_list': favourite_list })
```

Lo que hace la función home es recibir un mensaje, en este caso lo llamamos “**request**”.

Luego creando una variable llamada “**images**” que lo que hace es que reciba los valores tomados de la función localizada en un módulo de Django (**services.py**) “**services.getAllImages()**” que trae toda la información de la base de datos de Rick y Morty.

Lo mismo ocurre con la variable “**favourite\_list**” pero en este caso, es para una lista de favoritos.

Finalmente, con el comando “**return render**” lo que hace el programa es pedirle a Django que renderice el código en un **HTML** (lenguaje que sirve para estructurar y mostrar contenido en una web, en este caso “home.html” ya que es la página principal, la información que se manda será la de “**images**” y la de “**favourite\_list**”).

## 2. Services.py

```
def getAllImages(input=None):
    # obtiene un listado de datos "crudos" desde la API, usando a transport.py.
    # recorre cada dato crudo de la colección anterior, lo convierte en una Card y lo agrega a images.
    json_collection = transport.getAllImages(input)
    images = []
    for item in json_collection:
        if 'image' in item:
            images.append(translator.fromRequestIntoCard(item))
    return images
```

La función “**getAllimages**” será para tomar datos de la API desde el modulo `transport.py` que llegan en formato **JSON** (datos crudos; formato de intercambio de datos ligeros y de fácil lectura), luego lo convertirá en una “**Card**” (un elemento de interfaz que almacena fotos, títulos, descripción; más ordenado y leíble).

Se crea una variable para que tome estos datos crudos “**json\_collection = transport.getAllimages(input)**” y una variable que tome una lista “**images = []**”; donde se guardaran las **Cards**, luego mediante un ciclo “**for in**” que recorra cada ítem en “**json\_collection**”; si llega a un ítem que contenga una imagen lo guardara en la variable “**images**” mediante el comando “**images.append(translator.formRequestIntoCard(item))**”, el comando “**translator**” es lo que hará que se transformen en **Cards**.

Finalmente retornara la variable “**images**” ya con la lista de **Cards** (**return images**).

### 3. Home.html

```
<!--Esta seccion da un borde de color segun el estado del personaje-->
<div class="row row-cols-1 row-cols-md-3 g-4">
  {% if images|length == 0 %}
  <h2 class="text-center">La búsqueda no arrojó resultados...</h2>
  {% else %} {% for img in images %}
  <div class="col">
    <div class="card mb-3 ms-5" style="max-width: 540px;
      {% if img.status == 'Alive' %} border: 2.5px solid green;
      {% elif img.status == 'Dead' %} border: 2.5px solid red;
      {% else %} border: 2.5px solid orange;
      {% endif %}>
      <div class="row g-0">
        <div class="col-md-4">
          
        </div>
        <!-- Esta seccion parte indica el color del circulo de estado del personaje-->
        <div class="col-md-8">
          <div class="card-body">
            <h3 class="card-title">{{ img.name }}</h3>
            <p class="card-text">
              <strong>
                {% if img.status == 'Alive' %} ● {{ img.status }}
                {% elif img.status == 'Dead' %} ● {{ img.status }}
                {% else %} ● {{ img.status }}
                {% endif %}
              </strong>
            </p>
            <p class="card-text"><small class="text-body-secondary">Última ubicación: {{ img.last_location }}</small></p>
            <p class="card-text"><small class="text-body-secondary">Episodio inicial: {{ img.first_seen }}</small></p>
          </div>
        </div>
      </div>
    </div>
  {% endfor %}
</div>
```

Este fragmento de **HTML** lo que hará es diseñar las **Cards** (utilizando Django) y mostrar un color según su estado respectivamente.

Si no se recibe ninguna imagen mediante el condicional “**if images length == 0**”, lo que hará es mostrar un texto que diga “La búsqueda no arrojo resultados”.

Si se reciben imágenes, se utilizará un **ciclo for** “**for img in images**” que recorrerá cada imagen y las diseñará.

Lo que hará “**img.status**” es buscar si el personaje está Vivo (Verde), Muerto (rojo), Desconocido (Naranja) y les asignara un color en el borde de cada Card.

Finalmente con “**img.last\_location**” y “**img.first\_seen**” la Card mostrara la última y la primera que aparecieron los personajes mostrados.

#### 4. Buscador

```
def search(request):
    search_msg = request.POST.get('query', '')
    if search_msg: #Terminos de busqueda
        images=services.getAllImages(search_msg) # Llama a la API con el termino de busqueda
        return render(request, 'home.html', {'images': images})

    else:
        return home(request)
```

Lo que hará la función “**search**” será agregar un buscador para filtrar personajes.

La variable “**seach\_msg**” contendrá el texto que el usuario escriba (Ej: Dog; Judge...) mediante el comando “**request.POST.get('query', '')**”. “**request.POST**” tomara el texto ingresado por el usuario mientras que “**.get('query', '')**” busca el valor asociado a ese texto.

El condicional “**if seach\_msg**” busca si hay un término de búsqueda, es decir, si el usuario escribió algo. Si escribió algo, crea una variable (**images**) en donde se arrojarán los datos de la API llamando a la función “**getAllimages**”, es lo mismo que la función “**home**”, solo que ahora tomara los datos escritos en “**search\_msg**”.

Si el usuario escribió y coincide con la base de datos; arrojará las Cards filtradas. De lo contrario o si el usuario no escribió nada, volverá a “**home**”.

## 5. Favoritos

```
# Estas funciones se usan cuando el usuario está logueado en la aplicación.
@login_required
def getAllFavouritesByUser(request):
    favourite_list = services.getAllFavourites(request)

    return render(request, "favourites.html", { "favourite_list": favourite_list })

@login_required
def saveFavourite(request):
    save = services.saveFavourite(request)
    favourite_list = services.getAllFavourites(request)
    return render(request, "favourites.html", { "favourite_list": favourite_list })

@login_required
def deleteFavourite(request):
    delete = services.deleteFavourite(request)
    favourite_list = services.getAllFavourites(request)
    return render(request, "favourites.html", { "favourite_list": favourite_list })
```

Estas funciones localizadas en **views.py** harán que el usuario pueda agregar sus personajes favoritos a una lista, estas toman datos desde el modulo **services.py** que será explicado más tarde.

### **getAllfavouritesByUser:**

Esta función será para mostrar todos los favoritos del usuario.

Mediante una variable llamada “**favourite\_list**” se llama a la función “**getAllFavourites**” que obtiene la lista de favoritos del usuario autenticado. Luego lo muestra en la página con el comando “**return render**” que llena la información en el HTML “**favourites.html**”.

### **saveFavourite:**

Permite guardar al personaje seleccionado en favoritos usando la variable “**save**” que llama a la función “**saveFavourite**” que guarda elementos.

Toma de nuevo la lista de favoritos “**favourite\_list = services.getAllFavourites(request)**” y la actualiza con “**return render(request, 'favourites.html', { 'favourite\_list': favourite\_list })**”.

### **deleteFavourite:**

Esta función permite eliminar un favorito de la lista con la variable “**delete**” que llama al servicio “**deleteFavourite**”.

Otra vez toma la misma lista y la muestra

“**favourite\_list = services.getAllFavourites(request)**”

Como las anteriores funciones, se coloca el return.

“**return render(request, 'favourites.html', { 'favourite\_list': favourite\_list })**”.

## Services.py:

```
# añadir favoritos (usado desde el template 'home.html')
def saveFavourite(request):
    fav = translator.fromTemplateIntoCard(request) # transformamos un request del template en una Card.
    fav.user = get_user(request) # le asignamos el usuario correspondiente.

    return repositories.saveFavourite(fav) # lo guardamos en la base.

# usados desde el template 'favourites.html'
def getAllFavourites(request):
    if not request.user.is_authenticated:
        return []
    else:
        user = get_user(request)

        favourite_list = repositories.getAllFavourites(user) # buscamos desde el repositories.py TODOS los favoritos del usuario (variable 'user').
        mapped_favourites = []

        for favourite in favourite_list:
            card = translator.fromRepositoryIntoCard(favourite) # transformamos cada favorito en una Card, y lo almacenamos en card.
            mapped_favourites.append(card)

        return mapped_favourites

def deleteFavourite(request):
    favId = request.POST.get('id')
    return repositories.deleteFavourite(favId) # borramos un favorito por su ID.
```

En services.py se encuentran las 3 funciones utilizadas en las funciones de views.py. cada una está desarrollada para cada función general (guardar, obtener y eliminar).

### saveFavourite:

añade un favorito al **template** (base prediseñada para el desarrollo web) “**home.html**” vinculando al usuario autenticado.

La variable “**fav = translator.fromTemplateIntoCard(request)**” toma los datos del template y los tranforma en una Card.

Luego con “**fav.user = get\_user(request)**” se obtiene el usuario que hizo la “petición” de favorito.

Finalmente los guarda en la base de datos con el comando “**return repositories.saveFavourite(fav)**” que llama a un repositorio.

### getAllFavourites:

Esta función obtiene la lista de favoritos de usuario.

Primero se verifica si hay un usuario autenticado mediante el condicional “**if not request.user.is\_authenticated**”:

## Return []

Si no hay un usuario, devuelve una lista vacía, si lo hay, busca los favoritos del usuario con las variables **“user = get\_user(request)”** (obtiene el usuario); **“favourite\_list = repositories.getAllFavourites(user)”** (busca los favoritos de usuario en la base de datos).

Luego con un ciclo **“for in”** recorre la variable **“favourite\_list”** y transforma el **JSON** obtenido a una Card y las guarda en una lista llamada **“mapped\_favourites”**.

Finalmente retorna la lista con **“return mapped\_favourites”**.

## deleteFavourite:

Permite eliminar un favorito del usuario.

Primero toma la Id del usuario con la variable **“favId = request.POST.get('id')”** y llama al repositorio que borra el favorito seleccionado utilizando la Id del usuario **“return repositories.deleteFavourite(favId)”**

## 6. Funcion Exit:

```
@login_required
def exit(request):
    logout(request)
    return redirect('home')
```

Esta función permite al usuario deslogearse de la página utilizando la función **“logout”**. de Django.

## 7. Diseño de la página:

```
/* ===== General ===== */
body {
  color: #566787;
  font-family: 'Roboto', sans-serif;
  background: url('https://i.redd.it/lwvt86ci5anz.jpg') center center fixed;
  background-size: cover;
  margin: 0;
  padding: 0;
}

main {
  padding: 20px;
  min-height: 100vh;
  background-color: rgba(0, 0, 0, 0.6); /* Overlay semi-transparente */
}
```

Se modificó todo el diseño de la página por uno propio, en la imagen se muestra una parte de la modificación que se encuentra en **#static/styles.css**

Cada sección cuenta con notas que explican que sección se modifica.

**Finalmente, la página queda así con todos los cambios mencionados.**

Página principal (home.html)

Proyecto TP Inicio **Galería** Iniciar sesión

## Buscador Rick & Morty

← 1 2 3 →

Escribí una palabra

Buscar



**Rick Sanchez**

● **Alive**


Última ubicación: Citadel of Ricks  
Episodio inicial: Earth (C-137)



**Morty Smith**

● **Alive**

Última ubicación: Citadel of Ricks  
Episodio inicial: unknown



**Summer Smith**

● **Alive**


Última ubicación: Earth (Replacement Dimension)  
Episodio inicial: Earth (Replacement Dimension)



**Beth Smith**

● **Alive**


Última ubicación: Earth (Replacement Dimension)  
Episodio inicial: Earth (Replacement Dimension)



**Jerry Smith**

● **Alive**

Última ubicación: Earth (Replacement Dimension)  
Episodio inicial: Earth (Replacement Dimension)



**Abadango Cluster Princess**

● **Alive**

Última ubicación: Abadango  
Episodio inicial: Abadango



**Abradolf Lincler**

● **unknown**

Última ubicación: Testicle Monster Dimension  
Episodio inicial: Earth (Replacement Dimension)



**Adjudicator Rick**

● **Dead**

Última ubicación: Citadel of Ricks  
Episodio inicial: unknown



**Agency Director**

● **Dead**

Última ubicación: Earth (Replacement Dimension)  
Episodio inicial: Earth (Replacement Dimension)



Prueba del buscador


Proyecto TPInicioGaleríaIniciar sesión

Buscador Rick & Morty

←123→

Judge

Buscar




● Alive

Giant Judge

Última ubicación: Giant's Town

Episodio inicial: Giant's Town




● Dead

Judge

Última ubicación: Earth (Replacement Dimension)

Episodio inicial: Earth (Replacement Dimension)



● Alive









Public Opinion Judge

Última ubicación: Earth (Replacement Dimension)

Episodio inicial: Earth (Replacement Dimension)

Prueba de la lista de favoritos

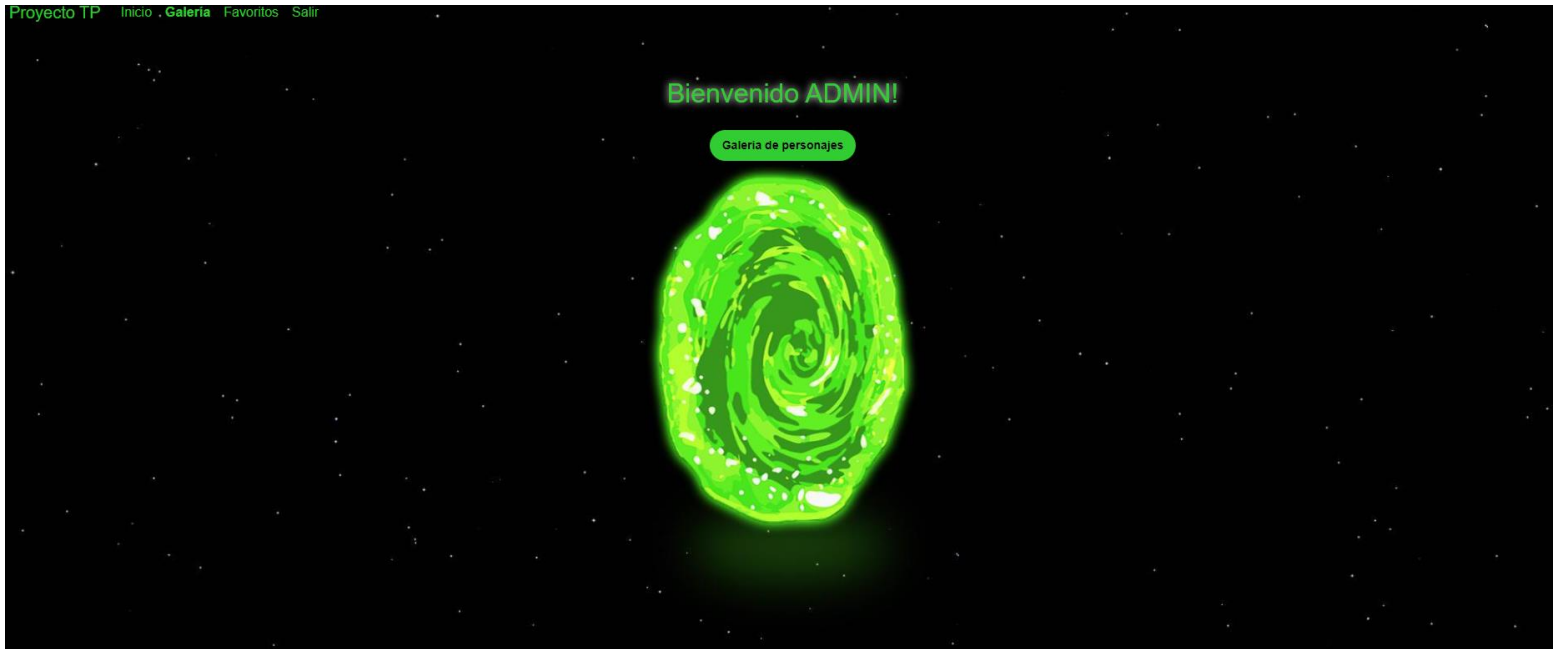
Listado de FAVORITOS

#	Imagen	Nombre	Status	Última ubicación	Episodio inicial	Acciones
-		Rick Sanchez	Alive	Citadel of Ricks	Earth (C-137)	
-		Morty Smith	Alive	Citadel of Ricks	unknown	
-		Beth Smith	Alive	Earth (Replacement Dimension)	Earth (Replacement Dimension)	
-		Jerry Smith	Alive	Earth (Replacement Dimension)	Earth (Replacement Dimension)	

Introducción a la Programación LUNGS

Trabajo práctico - 2do cuatrimestre del 2024

## Página de inicio (autenticado como ADMIN)



### Conclusión:

El trabajo consistió en terminar de codear una aplicación web usando Django que permitiera al usuario buscar imágenes de la serie Rick y Morty, usando una API. Esta información proveniente de la API se verá reflejada en **cards** que muestran la imagen del personaje, estado de vida, última ubicación y el episodio inicial de su aparición. Además, utilizando el buscador central podemos limitar esa información solo al personaje que deseemos, también se logró implementar un sistema de favoritos solo funcional para el admin, el programa se puede finalizar creando un módulo de autenticación y registro de nuevos usuarios para almacenar los favoritos correspondientes a cada usuario en su cuenta propia.

Nos sirvió para desarrollar un ambiente de trabajo respecto a un repositorio remoto, donde los integrantes vivimos y tenemos horarios distintos en los cuales trabajar, por lo cual aprender el uso de GitHub para tener un ambiente compartido en el cual ir subiendo los avances logrados fue necesario para llevar el trabajo al día. Fue una buena experiencia para aprender programación un poco más avanzada, lamentablemente debido a los problemas universitarios no se pudo tener un pleno aprendizaje ya que el desarrollo de la web requería nuevos conocimientos que no se llegaron a ver (HTML, JSON, CSS, Django) y se requirió la ayuda de videos externos, archivos de texto o páginas de IA como ChatGPT.

Este grupo intento hacer lo mejor posible pese a las circunstancias y esperamos que se pueda llegar a entender todas las modificaciones que se hicieron al repositorio original.