

# Informe del Trabajo Práctico: Buscador Rick & Morty

## Grupo 4

### Integrantes: Pérez Sebastián y Andrada Ariel

#### Introducción

Este trabajo práctico fue sobre crear una aplicación web con Django que usa la API de Rick & Morty para mostrar personajes en una galería. La aplicación tiene un buscador, permite iniciar sesión y también guardar personajes como favoritos. Todo esto permite que el usuario explore la galería y gestione sus favoritos de manera fácil.

#### Primer Commit: Mostrar imágenes de la API y galería dinámica

Este commit permite traer los personajes desde la API y mostrarlos en la página principal usando tarjetas que cambian el color del borde según el estado del personaje (vivo, muerto o desconocido).

#### Código: `views.py`

Esta función obtiene las imágenes de los personajes desde la API. Por ahora, no hay funcionalidad para favoritos, así que esa lista queda vacía. Luego, envía los datos al HTML para que se vean en la galería.

```
def home(request):
    # Llama al servicio para obtener las imágenes desde la API
    images = services.getAllImages()
    # Lista vacía de favoritos (sin desarrollar aún)
    favourite_list = []

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

#### Dificultad:

Asegurar que los datos obtenidos desde la API sean enviados correctamente al HTML y visualizados como tarjetas.

#### Decisión:

Se separó la lógica de obtención de datos en una capa de servicios para mantener la función simple y facilitar modificaciones futuras.

#### Código: `services.py`

Esta función se encarga de traer los datos desde la API. Si el usuario está buscando algo en particular, los filtra; si no, trae todos los personajes. Después, convierte esos datos en un formato que se pueda usar para mostrar en las tarjetas.

```

from ..transport.transport import getAllImages as fetch_images
# Función para obtener datos de la API
from ..utilities.translator import fromRequestIntoCard # Función para
transformar datos crudos en una Card

def getAllImages(input=None):

    # Si hay un filtro, busca datos específicos; si no, trae todo
    if input:
        json_collection = fetch_images(input)
    else:
        json_collection = fetch_images()

    images = []
    # Convierte los datos crudos en objetos más fáciles de manejar
    for item in json_collection:
        card = fromRequestIntoCard(item)
        images.append(card)

    return images

```

### **Dificultad:**

Transformar los datos crudos de la API en objetos tipo `Card` útiles para el HTML.

### **Decisión:**

Usar una función de transformación (`fromRequestIntoCard`) para estandarizar los datos y garantizar consistencia.

### **Código:** `home.html`




Estos dos bloques de código trabajan juntos para mejorar la visualización de los personajes. El primero cambia dinámicamente el color del borde de la tarjeta según el estado del personaje (verde para vivos, rojo para muertos y naranja para desconocidos). El segundo añade un icono junto al texto del estado para facilitar la identificación visual del personaje.

```

<div class="card mb-3 ms-5" style="max-width: 540px; border-color: {%
if img.status == 'Alive' %}green{% elif img.status == 'Dead' %}red{%
else %}orange{% endif %}; border-width: 3px; ">

```

```

{% if img.status == 'Alive' %}  {{ img.status }}
{% elif img.status == 'Dead' %}  {{ img.status }}
{% else %}  {{ img.status }}
{% endif %}

```

### **Dificultad:**

Incorporar lógica condicional directamente en el HTML para manejar dos aspectos dinámicos: el color del borde y el icono del estado. Además, asegurar que los valores (Alive, Dead o Unknown) se correspondan correctamente con sus estilos e iconos.

### **Decisión:**

Implementar la lógica en el template HTML para mantener las vistas y los servicios más limpios y permitir cambios rápidos en el diseño sin alterar la lógica en Python. Esto también facilita que los estilos y visualizaciones dependan solo del estado enviado desde la API.

## Segundo Commit: Buscador

Este commit permite buscar personajes en la galería usando el nombre o una palabra clave.

**Código:** views.py

Esta función permite buscar personajes según el texto ingresado. Si el usuario no escribe nada, lo lleva de nuevo a la galería completa.

```
def search(request):
    search_msg = request.POST.get('query', '').strip() # Toma el
    texto del buscador
    if search_msg:
        images = services.getAllImages(input=search_msg) # Busca con
        el texto ingresado
    else:
        images = services.getAllImages() # Si no se ingresó texto,
        muestra todas las imágenes (igual que en 'home')
        # Lista vacía de favoritos (no desarrollada esta funcionalidad)
        favourite_list = []
        return render (request, 'home.html', {'images': images,
        'favourite_list': []})
```

### Dificultad:

Asegurar que los datos filtrados sean relevantes y se muestren correctamente en la galería.

### Decisión:

Implementar una validación que redirige a la galería completa si el usuario no escribe texto, asegurando una experiencia sencilla.

## Tercer Commit: Inicio de sesión

Este commit añade la posibilidad de iniciar sesión con un usuario fijo para acceder a funcionalidades como favoritos.

**Código:** views.py

La función `login_view` permite al usuario ingresar sus credenciales (admin/admin) para acceder a la aplicación. La función `exit` cierra la sesión y redirige al inicio.

```
from django.contrib.auth import login, logout
from django.contrib.auth.models import User

def login_view(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        if username == 'admin' and password == 'admin': # Validación
            fija
            user = User.objects.get(username='admin') # Obtiene el
            usuario de la base
            login(request, user) # Inicia sesión
            return redirect('home') # Redirige a la galería principal
```

```
@login_required
def exit(request):
    logout(request) # Cierra la sesión
    return redirect('index-page') # Redirige al inicio
```

**Dificultad:**

Configurar el inicio de sesión de forma sencilla pero funcional para el proyecto.

**Decisión:**

Usar credenciales fijas (admin/admin) para facilitar pruebas y desarrollo sin depender del panel de administración.

**Cuarto Commit: Favoritos**

Este commit agrega la funcionalidad para que los usuarios logueados puedan guardar, eliminar y ver sus personajes favoritos.

**Código:** views.py

Las funciones permiten gestionar favoritos de la siguiente manera:

1. home incluye los favoritos del usuario autenticado y los pasa al HTML junto con las imágenes.
2. search asegura que los favoritos estén disponibles al buscar personajes.
3. saveFavourite y deleteFavourite permiten agregar y eliminar personajes de la lista de favoritos.
4. getAllFavouritesByUser muestra todos los favoritos guardados por el usuario.

```
def home(request):

    images = services.getAllImages()

    # Obtiene los favoritos del usuario si está autenticado; de lo
    contrario, asigna una lista vacía.

    if request.user.is_authenticated:

        favourite_list = services.getAllFavourites(request)

    else:

        favourite_list = []

def search(request):

    # Obtiene la lista de favoritos del usuario autenticado (si
    corresponde)

    favourite_list = services.getAllFavourites(request) if
    request.user.is_authenticated else []
```

@login\_required

```
def getAllFavouritesByUser(request):
    favourite_list = services.getAllFavourites(request) # Obtiene
    favoritos
    return render (request, 'favourites.html', {'favourite_list':
    favourite_list})
```

### @login\_required

```
def saveFavourite(request):

    if request.method == 'POST':

        services.saveFavourite(request) # Guarda el favorito

    return redirect('home') # Vuelve a la galería
```

### @login\_required

```
def deleteFavourite(request):
    if request.method == 'POST':
        services.deleteFavourite(request) # Borra el favorito
    return redirect('favoritos') # Vuelve a la lista de favoritos
```

### Dificultad:

Hacer que los favoritos funcionen en todas las páginas, como en la galería y al buscar. También asegurar que solo los usuarios logueados puedan usar esta funcionalidad.

### Decisión:

Usamos el decorador `@login_required` para que solo los usuarios autenticados puedan agregar, borrar o ver sus favoritos. También decidimos agregar los favoritos directamente en las vistas como `home` y `search` para que siempre se muestren donde corresponda.

### Código: services.py

Estas funciones manejan los datos de los favoritos en la base de datos. Se encargan de convertir los datos recibidos en un formato más sencillo de manejar (tipo Card) y aseguran que solo se guarden, eliminen o recuperen los favoritos correspondientes al usuario autenticado.

```
from ..utilities.translator import fromRequestIntoCard,
fromTemplateIntoCard, fromRepositoryIntoCard

from ..persistence import repositories

def saveFavourite(request):

    fav = fromTemplateIntoCard(request) # Convierte el request en un
    objeto manejable

    fav.user = request.user # Asigna el usuario autenticado

    return repositories.saveFavourite(fav) # Guarda el favorito
```

```

def getAllFavourites(request):

    if not request.user.is_authenticated:

        return []

    user = get_user(request)

    favourite_list = repositories.getAllFavourites(user) # Busca
    favoritos del usuario

    # Transformamos cada favorito en una Card

    mapped_favourites = [fromRepositoryIntoCard(fav) for fav in
    favourite_list]

    return mapped_favourites

def deleteFavourite(request):

    favId = request.POST.get('id') # Obtiene el ID del favorito

    return repositories.deleteFavourite(favId) # Lo borra

```

### **Dificultad:**

Garantizar que solo se guarden, eliminen o recuperen los favoritos que pertenecen al usuario autenticado. Además, transformar correctamente los datos desde y hacia un formato manejable para evitar errores.

### **Decisión:**

Usar funciones específicas como `fromTemplateIntoCard` y `fromRepositoryIntoCard` para transformar los datos en cada paso. Esto hace que el código sea más claro y fácil de mantener, además de garantizar que los datos procesados sean correctos.