Informe del Proyecto

Introducción

Este proyecto consiste en desarrollar una aplicación web basada en el universo de *Rick and Morty*, donde los usuarios pueden explorar personajes del programa y realizar búsquedas. La página principal muestra una galería de personajes obtenidos desde una API externa, permitiendo búsquedas y filtros en tiempo real.

El objetivo principal fue integrar varias capas de una arquitectura en Django, incluyendo transporte de datos, transformación de formatos, y una interfaz amigable para el usuario, enfocándonos en ofrecer una experiencia visual atractiva y funcional.

1.)
Ubicación: services.py
Función: getAllImages()

```python
# capa de servicio/lógica de negocio

from ..persistence import repositories
from ..utilities import translator
from ..transport import transport
from django.contrib.auth import get_user

def getAllImages(input=None):
    # obtiene un listado de datos "crudos" desde la API, usando a transport.py.
    json_collection = transport.getAllImages(input)

    # recorre cada dato crudo de la colección anterior, lo convierte en una Card y lo agrega a images.
    images = [translator.fromRequestIntoCard(image_data) for image_data in json_collection]

    return images      ggodoy, hace 2 meses • initial commit

```

Descripción: Esta función interactúa con las capas de transporte “transport.py” y traducción “translator.py” para obtener datos desde una API externa y transformarlos en objetos "Card". Si se recibe un parámetro “input”, se filtran los resultados antes de devolverlos.
Desafíos: Asegurar que la API responda y manejar errores como datos incompletos.
Decisiones: Se implementó un filtro en “transport.py” para descartar objetos sin la clave "image".

2.)
Ubicación: views.py
Función:home(request)

```python
# 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()
    favourite_list = []

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

```

Descripción: Esta función es la base de la página principal. Obtiene las imágenes de services.getAllImages() y las pasa al template home.html .
Desafíos: Integrar dinámicamente las imágenes con el template.
Decisiones: Para que imprima las images, le asignamos la función service.getAllImages del template, para q la función home recorra las images

3.)
Ubicación: views.py
Función: search(request)

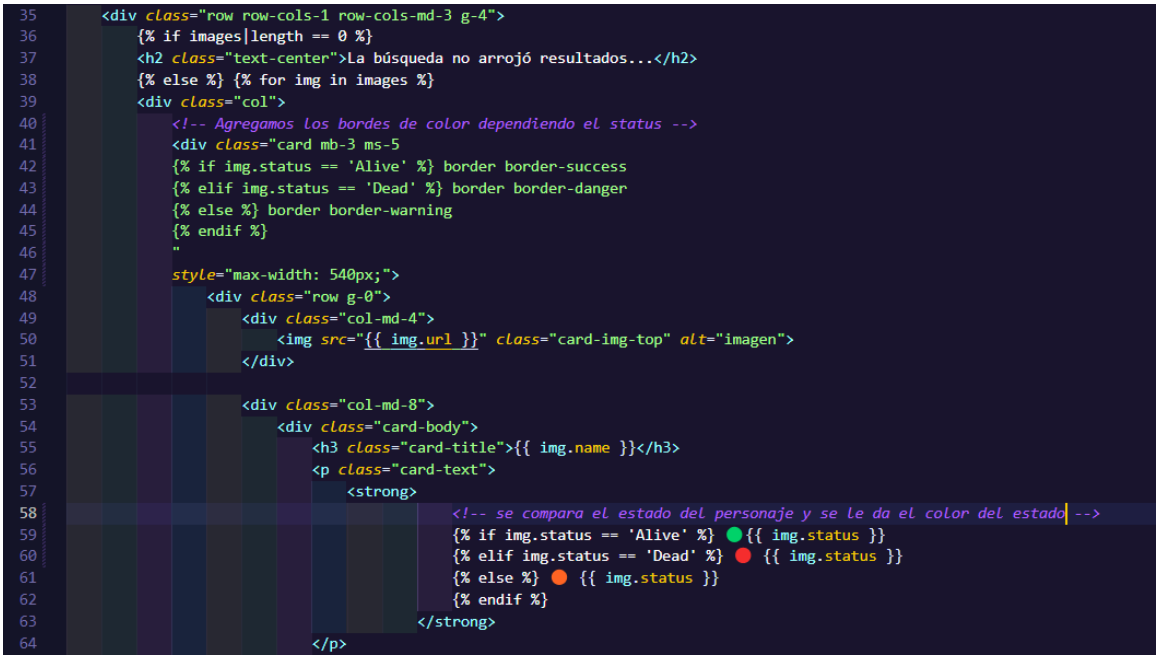
```python
def search(request):
    search_msg = request.POST.get('query', '')

    # si el texto ingresado no es vacío, trae las imágenes y favoritos desde services.py,
    # y luego renderiza el template (similar a home).
    if (search_msg != ''):
        images = services.getAllImages(search_msg)
        #renderizo el template home con las imagenes filtradas
        return render(request,'home.html',{'images':images})
    else:
        return redirect('home')

```

Descripción: Esta función maneja el buscador de la aplicación. Toma el término ingresado por el usuario y lo utiliza para filtrar los resultados obtenidos desde la API.
Desafíos: Filtrar los resultados de manera eficiente.
Decisiones: Utilizar un filtro en la API cuando es posible “transport.py”.

4.)
Ubicación: home.html
Función: cambiar el border y el estado según el 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">
        <!-- Agregamos los bordes de color dependiendo el status -->
        <div class="card mb-3 ms-5
        {% if img.status == 'Alive' %} border border-success
        {% elif img.status == 'Dead' %} border border-danger
        {% else %} border border-warning
        {% endif %}
        "
        style="max-width: 540px;">
            <div class="row g-0">
                <div class="col-md-4">
                    <img src="{{ img.url }}" class="card-img-top" alt="imagen">
                </div>

                <div class="col-md-8">
                    <div class="card-body">
                        <h3 class="card-title">{{ img.name }}</h3>
                        <p class="card-text">
                            <strong>
                                <!-- se compara el estado del personaje y se le da el color del estado -->
                                {% if img.status == 'Alive' %} 🟢 {{ img.status }}
                                {% elif img.status == 'Dead' %} 🔴 {{ img.status }}
                                {% else %} 🟠 {{ img.status }}
                                {% endif %}
                            </strong>
                        </p>
```

Descripción: Se modificó la estructura de las tarjetas para incluir clases dinámicas de Bootstrap que ajustan el color del borde en función del estado del personaje. Más abajo en la línea 59 hasta la 62 comparamos el estado del jugador y se le aplica el color predeterminado.

Desafíos: Ubicar los condicionales para que bordeen la carta con su respectivo color.

Decisiones: Finalmente se tomó la decisión de utilizar las clases dinámicas de Bootstrap para que se pueda visualizar en pocas líneas de código y en el mismo template.