INFORME TP INTRODUCCIÓN A LA PROGRAMACIÓN

INTEGRANTES: ROMAN FRIAS - SANTIAGO RUIZ - BRIAN ACUÑA.
INTRODUCCIÓN A LA PROGRAMACIÓN COMISION - 13

—----------

El objetivo de este trabajo fue desarrollar una página web en Django que interactúe con la API de Rick & Morty para buscar y mostrar información de personajes, permitiendo también que los usuarios autenticados gestionen sus personajes favoritos, que personajes están vivos, qué personajes no, cuál fue su última ubicación y el primer episodio en el cual aparece el personaje. Este sistema permite practicar el manejo de APIs, renderización dinámica de datos y creación de interfaces interactivas con funcionalidades centradas en la experiencia del usuario.
Lo primero que se hizo, fue agregar y modificar algunos códigos dentro del views.py, siendo asi agregado primeramente el siguiente código:

```
def getAllImages(input=None):
    url = f"https://rickandmortyapi.com/api/character/?name={input}" if input else "https://rickandmortyapi.com/api/character"
    response = requests.get(url)
    data = response.json()
    print(data)  # Verifica qué datos estás obteniendo desde la API
    characters = data.get('results', [])
    cards = []
    for character in characters:
        cards.append({
            'name': character['name'],
            'image': character['image'],
            'status': character['status'],
            'location': character['location']['name'],
            'episode': character['episode'][0].split("/")[-1]
        })
    return cards
```

Este código cumple la función de Construcción del URL La función toma un parámetro opcional input. Si input contiene un valor (por ejemplo, el nombre de un personaje),la URL incluye este parámetro para filtrar resultados. Si no, carga todos los personajes. Se utiliza el request para realizar una solicitud GET a la API, recuperando datos en formato JSON. Luego accede a la clave results del JSON (que contiene los personajes) y crea una lista llamada cards con datos específicos de cada personaje. El nombre,la imagen del personaje, el estado del personaje, la última ubicación y por último el primer episodio donde aparece el personaje. Se ejecuta un print(data) para verificar la estructura del JSON obtenido durante el desarrollo.
Luego se modificó el comando de def home dentro del views.py.

```
def home(request):
    images = getAllImages()  # Llama a la función
    print(images)  # Agrega esto para verificar si se están recibiendo imágenes correctamente
    favourite_list = []
    return render(request, 'home.html', { 'images': images, 'favourite_list': favourite_list })
```

Este código ejecuta enviar los datos necesarios al template home.html para renderizar la página principal de la página. El getALLimages llama a la función así se obtiene una lista de personajes desde la API de Ricky Morty. Puse el print para que esto se ejecute y verifique que

los datos recibidos de la API estén correctos. Dejamos el favourite list vacía para los personajes favoritos del usuario.

Luego también hubo cambios dentro del archivo home.html
img.url → img.image
img.last_location → img.location
img.first_seen → img.episode
tuvimos que cambiar el nombre para que coincidan con el getALLimages que estaba en el views.py, ya que no había coincidencia con los nombres del home.html y el views.py, quedando esta parte del código del home.html de la siguiente manera:

```
<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;">
        <div class="row g-0">
          <div class="col-md-4">
            <img src="{{ img.image }}" class="card-img-top" alt="{{ img.name }}">
          </div>
          <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.location }}</small></p>
              <p class="card-text"><small class="text-body-secondary">Episodio inicial: {{ img.episode }}</small></p>
            </div>
            {% if request.user.is_authenticated %}
            <div class="card-footer text-center">
              <form method="post" action="{% url 'agregar-favorito' %}">
                {% csrf_token %}
                <input type="hidden" name="name" value="{{ img.name }}">
                <input type="hidden" name="url" value="{{ img.image }}">
                <input type="hidden" name="status" value="{{ img.status }}">
                <input type="hidden" name="last_location" value="{{ img.location }}">
                <input type="hidden" name="first_seen" value="{{ img.episode }}">
                {% if img in favourite_list %}
                <button type="submit" class="btn btn-primary btn-sm" disabled>✔️ Ya está en favoritos</button>
                {% else %}
                <button type="submit" class="btn btn-primary btn-sm">❤️ Añadir a favoritos</button>
                {% endif %}
              </form>
            </div>
            {% endif %}
          </div>
        </div>
      </div>
    {% endfor %}
    {% endif %}
  </div>
</main>
{% endblock %}
```

Este código implementa una página para buscar personajes de **Rick & Morty**, mostrando los resultados en un formato de tarjetas y permitiendo a los usuarios autenticados agregar

personajes a su lista de favoritos. Con este código ya se podía entrar a la página y se podían ver las imágenes de los personajes dentro de la sección galería, apareciendo de si estaban vivos, muertos, etc. Lo que nos faltaba en ese entonces era el login, la busqueda especifica de los personajes y la paginacion para poder avanzar, ya que contaba con mas de 60 personajes.
Pero primero lo que hicimos fue agregar el recuadro de colores de los personajes en cada una de las imagenes, para ver si su estado era vivo(color verde), muerto(color rojo) o desconocido (naranja). Habiendo un pequeño cambio en el home.html , borrando una linea y agregando una nueva linea.

```
<div class="card mb-3 ms-5" style="max-width: 540px; (se borro)
<!-- Agregamos clases dinámicas según el estado -->
<div class="card mb-3 ms-5 {% if img.status == 'Alive' %}border-success{% elif
img.status == 'Dead' %}border-danger{% else %}border-warning{% endif %}"
style="max-width: 540px;"> (se agrego)
```

Habiendo agregado ese comando, lo que hacía era justamente que se agreguen los recuadros de personajes correspondiente a su condición. Esto mejora la experiencia visual al destacar el estado del personaje de manera clara e intuitiva.
Luego se modifico el views.py nuevamente para poder hacer la búsqueda de personajes correctamente, quedando efectuado el código de la siguiente manera:

```
# capa de vista/presentación
from django.shortcuts import redirect, render
from .layers.services import services
from django.contrib.auth.decorators import login_required
from django.contrib.auth import logout

def index_page(request):
    return render(request, 'index.html')
# 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 = getAllImages()  # Obtén todas las imágenes
    favourite_list = []  # Puedes implementar favoritos si lo necesitas
    return render(request, 'home.html', {'images': images, 'favourite_list': favourite_list})

def search(request):
    search_msg = request.POST.get('query', '').strip()  # Limpia el texto ingresado
    if not search_msg:
        # Si no se ingresa texto, muestra todas las imágenes
        return redirect('home')
    # Filtra las imágenes basándote en el texto ingresado
    images = getAllImages(input=search_msg)
    favourite_list = []  # Implementa favoritos si es necesario
    return render(request, 'home.html', {'images': images, 'favourite_list': favourite_list})

import requests

def getAllImages(input=None):
    url = f"https://rickandmortyapi.com/api/character/?name={input}" if input else "https://rickandmortyapi.com/api/character"
    response = requests.get(url)
    data = response.json()
    print(data)  # Verifica qué datos estás obteniendo desde la API
    characters = data.get('results', [])
```

```
    cards = []
    for character in characters:
        cards.append({
            'name': character['name'],
            'image': character['image'],
            'status': character['status'],
            'location': character['location']['name'],
            'episode': character['episode'][0].split("/")[-1]
        })
    return cards

# Estas funciones se usan cuando el usuario está logueado en la aplicación.
@login_required
def getAllFavouritesByUser(request):
    favourite_list = []
    return render(request, 'favourites.html', { 'favourite_list': favourite_list })

@login_required
def saveFavourite(request):
    pass

@login_required
def deleteFavourite(request):
    pass

@login_required
def exit(request):
    pass
```

En el código modificado, el valor del campo de búsqueda (`query`) es limpiado con `.strip()` para evitar espacios adicionales.Si el texto de búsqueda está vacío, redirige al usuario directamente a `home`. Agrega la funcionalidad de filtrar imágenes basándose en la búsqueda y muestra los resultados en `home.html`.

Hubo un cierto percance con respecto a un integrante del grupo ya que no tenía acceso a las computadoras, porque estaba mandada a arreglar y recién el martes a la tarde pudo estar nuevamente y pudo ponerse al día configurando todo lo del git, descargando los recursos necesarios para arrancar con el trabajo.

Un problema bastante grande que tuvimos fue la navegación de página, al momento de iniciar la página nos tiraba distintos errores a medida que íbamos cambiando el views.py y el home.html, probamos tocando las urls también pero nos seguía tirando error. Una vez que nos dejó entrar y navegar entre las páginas 1,2,3… etc funcionaba perfecto pero el código no encajaba con el login ya que al momento de intentar logearnos la página tiraba error y no nos dejaba volver a acceder. Por otra parte por más que funcionara la paginación, al momento de buscar un personaje como por ejemplo “Rick”, solo nos aparecía 1 pagina nomas de 20 “Ricks” pero no nos dejaba ingresar a la 2da página. Finalmente charlamos en el grupo y optamos por dejar eso de lado para arreglar el tema del registro/login asi no perdemos más tiempo y poder terminar lo del login para así, tener mas completo el trabajo.

Pero también tuvimos problemas con el código para iniciar sesión. Lograr que inicie sesión correctamente no fue tanto problema, ya que la página tomaba como único usuario válido el de

usuario y contraseña “admin” . El problema vino cuando se intentó implementar la función de registrarse, en decir, crear un nuevo usuario. Problemas había miles: cuando se apretaba el botón de registrarse saltaba un error, cuando se colocaban los datos para el registro y no se creaba el usuario y la página se quedaba donde estaba, y un largo etc. Pero, después de todo, se pudo implementar exitosamente una pestaña de registro que funciona muy bien. Para esto se creó un nuevo archivo .html, llamado register.html que abriria un formulario para que el registro se complete: {% extends 'header.html' %}

```
{% block content %}
<div class="container" style="max-width: 500px; margin-top: 50px;">
    <h2 class="text-center">Registro de Usuario</h2>

    <!-- Formulario de registro -->
    <form method="POST" action="{% url 'register' %}">
        {% csrf_token %}

        <!-- Mostrar formulario de Django -->
        {{ form.as_p }}  <!-- Muestra el formulario con los campos de
forma automática -->

        {% if form.errors %}
            <ul>
                {% for field in form %}
                    {% for error in field.errors %}
                        <li>{{ error }}</li>
                    {% endfor %}
```

```
                {% endfor %}

            </ul>

        {% endif %}



        <!-- Botón de enviar -->

        <button type="submit" class="btn btn-primary
btn-block">Registrarse</button>

    </form>



    <!-- Enlace para ir al inicio de sesión -->

    <p class="text-center" style="margin-top: 15px;">

        ¿Ya tienes cuenta? <a href="{% url 'login' %}">Inicia sesión
aquí</a>.

    </p>

</div>

{% endblock %}
```

Conclusión

En este trabajo nos pareció muy interesante el poder programar una página y que estuvo muy bueno el trabajo en conjunto con los demás integrantes. Aprendimos el uso del git, de cómo añadir cosas a los repositorios además de que se aprendieron nuevos códigos de programación. Este trabajo nos permitió aprender a integrar diferentes tecnologías en un proyecto web usando Django, como el manejo de autenticación de usuarios, la gestión de datos externos (como las imágenes de la API de Rick & Morty) y la creación de interfaces dinámicas con plantillas HTML. Además de que nos sirvió a resolver diversos problemas técnicos que se

nos presentaron en grupo, este tp final sirve para más adelante preparándonos para futuros proyectos proporcionando una visión mucho más integral sobre el desarrollos de paginas web. También es cierto que tuvimos que investigar sobre muchas cosas que por ahí no entendíamos, se nos hizo muy difícil por el hecho de no tener conocimientos básicos de algunas cosas como por ejemplo el HTML, tuvimos que averiguar masomenos de cómo funcionaba y demás.