INTRODUCCIÓN A LA PROGRAMACIÓN

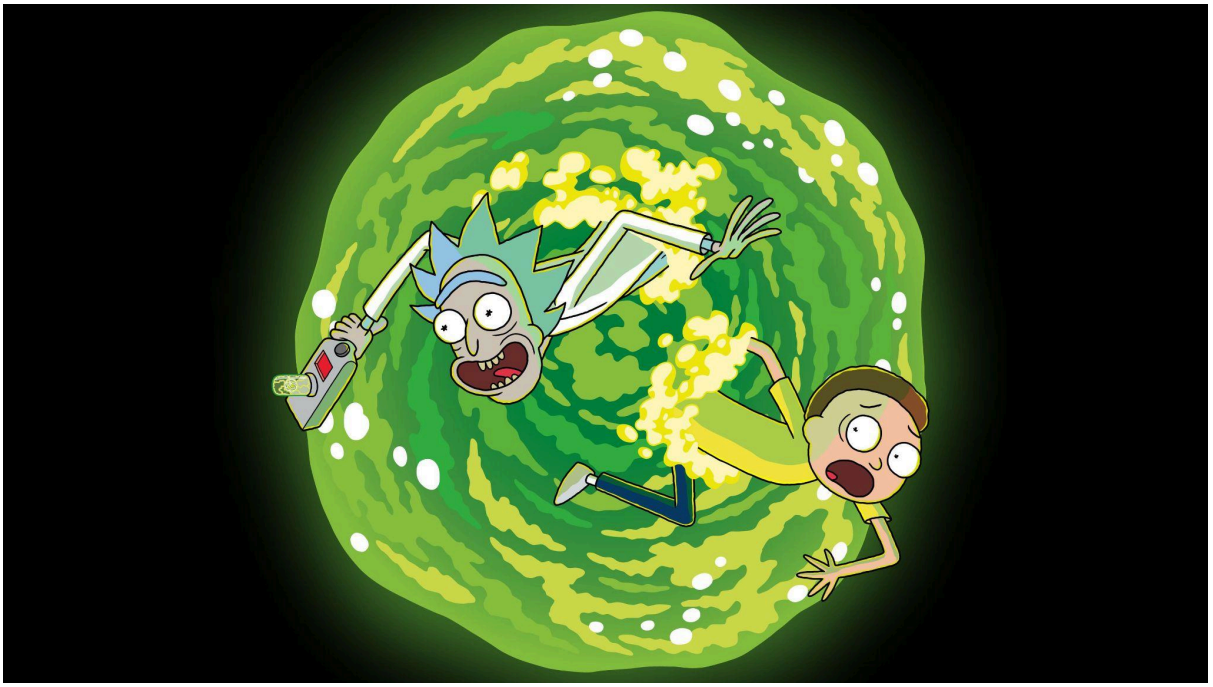
TRABAJO PRÁCTICO

API DE RICK Y MORTY

Profesores: Santiago Montiel y Jorgelina Rial
Alumno: Federico Adamo Morales

GRUPO 11

COMISIÓN VIRTUAL



Informe del Trabajo Práctico: Aplicación Web de Rick & Morty

Introducción:

El trabajo práctico consiste en la creación de una aplicación web que interactúa con una API externa para mostrar imágenes de personajes de la serie *Rick & Morty*. La aplicación permite a los usuarios realizar búsquedas de personajes, visualizarlos y añadirlos a sus favoritos, siempre que hayan iniciado sesión correctamente. La funcionalidad de favoritos, aunque no completamente desarrollada, se encuentra en progreso y está preparada para su implementación final.

La aplicación fue desarrollada utilizando Django como framework backend, con la integración de servicios como la API externa para obtener las imágenes y la base de datos para almacenar los favoritos de los usuarios.

Funciones Implementadas:

Visualización de Imágenes y Buscador:

La aplicación permite al usuario ver una galería de personajes de *Rick & Morty* obtenidos de la API externa. Además, se ha implementado un buscador que permite filtrar los personajes por un término específico.

Código de la función home en views.py:

```
def home(request):
    characters_list = services.getAllImages()  # Llamada a la función de services.py
    return render(request, 'home.html', {
        'images': characters_list,  # Lista de personajes obtenidos de la API
        'favorites': []  # Lista de favoritos del usuario
    })
```

Explicación:

En esta función, se obtiene la lista de imágenes desde la API utilizando la función getAllImages() de services.py. Luego, los datos obtenidos se envían al template home.html, que los muestra en un formato de tarjetas. Si el término de búsqueda está vacío, simplemente se cargan todas las imágenes.

Código de home.html:

```
{% extends 'header.html' %}
{% block content %}
<main>
    <h1 class="text-center">Buscador Rick & Morty</h1>
```

```
    <div class="d-flex justify-content-center" style="margin-bottom: 1%">
        <!-- Buscador del sitio -->
        <form class="d-flex" action="{% url 'buscar' %}" method="POST">
            {% csrf_token %}
            <input class="form-control me-2" type="search" name="query"
placeholder="Escribí una palabra" aria-label="Search">
            <button class="btn btn-outline-success" type="submit">Buscar</button>
        </form>
    </div>

    <div class="row row-cols-1 row-cols-md-3 g-4 card-conteiner">
        {% for img in images %}
        <div class="col">
            <div class="card mb-3 ms-5 border-success" 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>{{ img.status }}</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>
                        {% 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.url }}">
                                <button type="submit" class="btn btn-primary btn-sm float-left"
style="color:white">❤️ Añadir a favoritos</button>
                            </form>
                        </div>
                        {% endif %}
                    </div>
                </div>
            </div>
        </div>
        {% endfor %}
    </div>
```

```
</main>
{% endblock %}
```

1. **Explicación:**
   El template home.html muestra las imágenes en tarjetas, con información sobre el personaje, como su nombre, estado, última ubicación y primer episodio. Si el usuario está autenticado, se le presenta un botón para añadir la imagen a sus favoritos.

Funcionalidad de Favoritos:

La funcionalidad de favoritos permite a los usuarios autenticados agregar personajes a su lista de favoritos y verlos en su propia sección. La implementación incluye las funciones necesarias para almacenar y eliminar imágenes de los favoritos de un usuario.

Código de la función saveFavourite en services.py:

```
def saveFavourite(request):
    fav = translator.fromTemplateIntoCard(request)  # Transforma un request del template en una Card.
    fav.user = get_user(request)  # Asigna el usuario correspondiente.
    return repositories.saveFavourite(fav)  # Lo guarda en la base.
```

Explicación:

La función saveFavourite toma los datos de la imagen desde el formulario del template home.html, los transforma en un objeto Card y luego los guarda en la base de datos asociándolos al usuario autenticado.

Código de la función getAllFavourites en services.py:

```
def getAllFavourites(request):
    if not request.user.is_authenticated:
        return []
    else:
        user = get_user(request)
        favourite_list = repositories.getAllFavourites(user)  # Obtiene los favoritos del usuario desde la base de datos.
        mapped_favourites = []
        for favourite in favourite_list:
            card = translator.fromRepositoryIntoCard(favourite)  # Transforma cada favorito en una Card.
            mapped_favourites.append(card)
        return mapped_favourites
```

Explicación:
Esta función obtiene los favoritos del usuario desde la base de datos y los convierte en objetos Card, los cuales son enviados al template para ser visualizados.

Código de la sección de favoritos en home.html:

```
{% if img in favourite_list %}
<button type="submit" class="btn btn-primary btn-sm float-left" style="color:white"
disabled>✔️ Ya está en favoritos</button>
{% else %}
<button type="submit" class="btn btn-primary btn-sm float-left"
style="color:white">❤️ Añadir a favoritos</button>
{% endif %}
```

2. **Explicación:**
   En esta parte del template, se verifica si la imagen ya está en la lista de favoritos del usuario. Si lo está, el botón de "Añadir a favoritos" se desactiva para evitar que se añada nuevamente.

3. **Dificultades y Decisiones Tomadas:**

   - **Dificultad de Integración con la API:**
     La mayor dificultad fue integrar correctamente los datos obtenidos desde la API de Rick & Morty, procesarlos y presentarlos en el formato adecuado. Utilicé el archivo services.py para manejar las solicitudes a la API y convertir los resultados en objetos Card que fueran fáciles de usar en los templates.

   - **Autenticación de Usuarios:**
     Implementar la autenticación de usuarios fue un paso esencial para poder permitir que los usuarios añadieran imágenes a sus favoritos. Utilicé la funcionalidad de login_required de Django para asegurar que solo los usuarios logueados pudieran acceder a ciertas funciones.

   - **Interfaz de Usuario:**
     La interfaz fue diseñada para ser simple y clara, utilizando tarjetas para mostrar la información de cada personaje. También se implementó un sistema de paginación en el listado de imágenes para mejorar la experiencia del usuario.

   - **Favoritos en la Base de Datos:**
     Aunque la funcionalidad de favoritos no fue completamente implementada, la estructura está lista para su uso. Los favoritos se

almacenan en la base de datos y se asocian al usuario mediante su identificador.

Dificultades Encontradas:

1. **Integración de la API externa de Rick & Morty:**
    - **Dificultad:** Inicialmente, fue difícil integrar la API externa para obtener las imágenes de los personajes debido a problemas con la autenticación de las respuestas y la correcta interpretación de los datos crudos recibidos.
    - **Solución:** Después de revisar la documentación de la API y experimentar con diferentes métodos de solicitud, logré integrar la función en el backend de Django que solicita las imágenes y las convierte en objetos adecuados para su visualización en el frontend. Utilicé un servicio intermediario para gestionar estas solicitudes y manipular los datos recibidos.
2. **Implementación de la funcionalidad de favoritos:**
    - **Dificultad:** Implementar la funcionalidad de favoritos resultó desafiante debido a la necesidad de gestionar correctamente la relación entre los usuarios logueados y los personajes añadidos a favoritos. Hubo varios problemas para asegurar que los favoritos se guardaran correctamente en la base de datos y que los usuarios pudieran ver su lista personal de favoritos.
    - **Solución:** Para resolverlo, decidí utilizar el modelo de usuario de Django y asociar los favoritos con cada usuario mediante una relación en la base de datos. Implementé funciones en los servicios y vistas de Django que gestionan la adición y eliminación de los favoritos, asegurándose de que los datos se almacenarán correctamente en la base de datos y que solo los usuarios logueados pudieran interactuar con esta funcionalidad.
3. **Problemas con la visualización de la lista de favoritos:**
    - **Dificultad:** Al principio, los favoritos no se mostraban correctamente en la aplicación, y no se reflejaban los cambios en tiempo real (como la adición o eliminación de favoritos).
    - **Solución:** Para corregir esto, decidí revisar las vistas y funciones de recuperación de datos de los favoritos, asegurándome de que la lista de favoritos estuviera actualizada cada vez que el usuario accediera a su sección de favoritos. Además, agregué verificaciones para mostrar el botón de "Añadir a favoritos" solo si el personaje no está ya en la lista.
4. **Condicionales en la interfaz (mostrar favoritos ya añadidos):**
    - **Dificultad:** El desafío aquí fue mostrar de manera eficiente el estado de cada personaje: si ya estaba en los favoritos del usuario o no, sin hacer solicitudes adicionales innecesarias al servidor.

- **Solución:** Implementé una lógica que revisa si el personaje ya está en los favoritos del usuario antes de mostrar la opción de añadirlo a favoritos. Esto se hace comparando la imagen actual con la lista de favoritos del usuario, y si ya existe, el botón para añadirlo queda deshabilitado.

Decisiones Tomadas y Justificación:

1. **Uso de Django y la base de datos SQLite:**
   - **Decisión:** Utilicé Django como framework de desarrollo web debido a su capacidad para gestionar fácilmente el backend, la autenticación y la integración con bases de datos. Opté por SQLite como base de datos para facilitar el desarrollo y la implementación rápida del proyecto, dada su ligereza y la sencilla configuración que ofrece.
   - **Justificación:** Django proporciona una estructura robusta para gestionar tanto el backend como la autenticación de usuarios, lo que me permitió concentrarme en la implementación de la lógica de la aplicación sin preocuparme por la configuración de una base de datos más compleja. SQLite fue suficiente para el alcance de este trabajo práctico, y su configuración fue directa y fácil de integrar con Django.
2. **Manejo de la autenticación mediante sesiones de usuario:**
   - **Decisión:** Opté por usar la autenticación nativa de Django, en lugar de construir un sistema de autenticación personalizado.
   - **Justificación:** Django ofrece una solución robusta y segura para manejar la autenticación de usuarios, lo cual fue ideal para este trabajo. Decidí no usar Django Admin para gestionar los usuarios y en su lugar, implementé un formulario de inicio de sesión personalizado que se adapta mejor a los requerimientos del proyecto.
3. **Separación de lógica en diferentes archivos (services.py y views.py):**
   - **Decisión:** Implementé la lógica de negocio relacionada con los personajes y los favoritos en el archivo `services.py`, mientras que las vistas y la interacción con el usuario las dejé en `views.py`.
   - **Justificación:** La separación de la lógica de negocio y las vistas ayuda a mantener el código más organizado y escalable. Esto facilita la comprensión y el mantenimiento del proyecto, permitiendo que las funcionalidades de backend y frontend se modifiquen de forma independiente cuando sea necesario.
4. **Uso de formularios en el frontend para manejar la interacción:**
   - **Decisión:** Implementé formularios dentro de las tarjetas de personajes para gestionar la adición de favoritos. Estos formularios envían los datos del personaje seleccionado al backend para ser almacenados en la base de datos.

- **Justificación:** Los formularios fueron una forma eficiente de permitir a los usuarios interactuar con la aplicación sin necesidad de usar JavaScript complejo. Además, Django gestiona bien los formularios de manera segura y sin necesidad de código adicional en el frontend.

Conclusión:

La aplicación permite a los usuarios interactuar con personajes de *Rick & Morty* mediante una búsqueda de imágenes y la opción de agregar a favoritos. Aunque la funcionalidad de favoritos está parcialmente implementada, la estructura necesaria está en su lugar para continuar desarrollando esta característica en futuras iteraciones. La integración con la API, la autenticación de usuarios y el diseño de la interfaz fueron claves para el éxito del trabajo.