Materia: Introducción a la Programación.

Comisión 4

Profesor: Argarañas Omar y Nores Nancy.

Alumnos: Nicastro Matheo, Vega Ramiro y Toledo Lisandro.

Fecha de Entrega: 27/11/2024.

Índice:

Introducción:

El presente trabajo tiene como objetivo principal desarrollar una aplicación web utilizando el framework Django, que permita buscar y desplegar imágenes de los personajes de la reconocida serie animada Rick & Morty. Para ello, se hará uso de la API oficial de la serie, la cual facilita el acceso a información detallada sobre los personajes, sus ubicaciones, y los episodios en los que aparecen.

Django es un framework de desarrollo web de código abierto escrito en el lenguaje de programación Python. Se entiende por "framework" un entorno de trabajo preconfigurado que proporciona herramientas y componentes para facilitar el desarrollo de aplicaciones. Por otro lado, el término API, que proviene del inglés Application Programming Interface (interfaz de programación de aplicaciones), hace referencia a un conjunto de definiciones y protocolos que permiten la interacción entre distintos programas o aplicaciones. En este contexto, la API de Rick & Morty servirá como puente entre nuestra aplicación y la base de datos que contiene la información relevante sobre la serie.

La aplicación desarrollada estará diseñada para presentar la información obtenida de la API en un formato visual atractivo y funcional, utilizando “cards”. Estas tarjetas mostrarán la imagen de cada personaje junto con información clave, como su estado (si está vivo, muerto o su estado es desconocido), su última ubicación conocida y el episodio inicial en el que fue presentado. Además, se prevé la posibilidad de ampliar la funcionalidad de la aplicación incorporando características adicionales. Entre ellas se destaca la implementación de un buscador central que permita filtrar personajes por nombre o estado, así como la creación de un módulo de autenticación básica mediante usuario y contraseña. Este módulo permitiría a los usuarios registrados almacenar sus personajes favoritos en una sección personalizada dentro de la página.

Desarrollo:

Universidad Nacional de General Sarmiento

El código que implementamos utiliza funciones y variables para procesar la información proporcionada por una API sobre Rick and Morty. Haciendo uso del framework Django, junto con HTML y sus complementos, transforma esa información en una presentación visual atractiva a través de tarjetas (cards). Estas tarjetas, que se despliegan en una página web, muestran datos clave de los personajes: el nombre, su estado (vivo, muerto o desconocido), su última ubicación conocida y el primer episodio en el que aparecen.

Además, se incorpora un detalle visual dinámico: el borde de cada tarjeta cambia de color según el estado del personaje. Si está vivo, el borde es de un color; si está muerto, de otro, y si su paradero es desconocido, se asigna un color distinto. Este enfoque no solo organiza la información de forma clara y accesible, sino que también añade un componente visual intuitivo que facilita la comprensión del contenido.

Implementaciones:

En este trabajo se implementaron diversas funciones para lograr nuestro fin. La primera función implementada es la que se encuentra en el archivo *views.py:*

```
def home(request):
    images = services.getAllImages()
    favourite_list = services.getAllFavourites(request)
    return render(request, 'home.html', { 'images': images, 'favourite_list': favourite_list })
```

La función `home` utiliza la variable `request` y, mediante la variable `image`, llama a una función externa llamada `getAllImages`, ubicada en `services.py`, que recorre una lista con información proveniente de una API. Además, incluye la variable `favourite_list`, que genera una lista personalizada de elementos favoritos elegidos por el usuario. Finalmente, la función retorna la información almacenada en `images`, junto con la lista de favoritos guardada en `favourite_list`.

La siguiente función que implementamos, ubicada en el mismo archivo, es:


```
def search(request):
    search_msg = request.POST.get('query', ' ')
      if (search_msg != ' '):
        images = services.getAllImages(search_msg)
        favourite_list = services.getAllFavourites(request)
        return render(request, 'home.html', { 'images': images, 'favourite_list': favourite_list
})
    else:
        return redirect('home')
```

La función `search` toma como argumento la variable `request` y define la variable `search_msg`, que obtiene el valor del parámetro `'query'` del método `POST` en `request`. Si este parámetro está vacío, se asigna un valor predeterminado vacío (`' '`). Luego, establece un condicional donde, si `search_msg` no está vacío, la variable `image` llama a la función externa `getAllImages`, ubicada en `services.py`, pasando `search_msg` como argumento. Del mismo modo, la variable `favourite_list` utiliza `search_msg` como argumento. Si se cumple la condición, la función retorna un renderizado de `home.html` con las imágenes y la lista de favoritos. Si no se cumple, redirige al home de la página web.

Luego de terminar con el archivo views.py pasamos al otro archivo pedido que era el services.py el cual contiene:

```
from ..transport import transport
```

Lo primero que hicimos fue traer del archivo transport ubicado en otra carpeta, todas las funciones que este alberga.

Universidad Nacional de General Sarmiento

```
getAllImages(input=None):
    json_collection = transport.getAllImages(input)
    images = []
    for object in json_collection:
        card = translator.fromRequestIntoCard(object)
        images.append(card)

    return images
```

Definimos la función `getAllImages`, que recibe un parámetro `input` (por defecto `None`). Dentro de esta, se declara la variable `json_collection`, que llama a la función `getAllImages` del archivo `transport.py`, pasando el `input` como argumento y obteniendo una lista de datos. Luego, se crea una lista vacía llamada `images`, donde se almacenarán los resultados. Con un bucle `for`, iteramos sobre los elementos de `json_collection`, llamando a cada uno `object`, y en cada iteración se convierte el `object` en una "card" usando la función `fromRequestIntoCard`. Esta "card" se agrega a la lista `images`. Finalmente, la función retorna la lista completa de tarjetas procesadas.

En el siguiente archivo, llamado home.html el cual es el cuerpo de la página web, se implementó lo siguiente

```
<strong>
  {% if img.status == 'Alive' %}
```

```
        🟢 {{ img.status }}
      {% elif img.status == 'Dead' %}
        🔴 {{ img.status }}
      {% else %}
        🟠 {{ img.status }}
      {% endif %}
    </strong>
```

En primer lugar, se implementó un condicional en Django que, según el **status** del personaje proporcionado por la API, muestra un círculo de color específico: verde si el personaje está vivo, rojo si está muerto y naranja si su estado es desconocido.

```
<div class="col">
  <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;">
```

Además, se implementó un condicional en Django que, según el valor de `img.status` (vivo, muerto o desconocido), asigna un color específico al borde de las tarjetas. Estos bordes utilizan clases predefinidas: **success** para un borde verde (vivo), **danger** para un borde rojo (muerto) y **warning** para un borde naranja (desconocido).

El apartado de inicio de sesión se dejó con la configuración por defecto para evitar trabajar con bases de datos debido al tiempo limitado para completar el proyecto. Por

lo tanto, no se implementó un sistema de registro de cuentas, y se utilizó la cuenta predeterminada “admin/admin” (usuario/contraseña) para desarrollar y probar las funciones relacionadas con la gestión de favoritos. Aun así, para cerrar sesión se implementó la siguiente función:

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

La misma dice que hay que acceder a una cuenta para ser utilizada. Lo que hace es redirigir a la parte de la página donde se cierra la sesión y vuelve a la página inicial.

Para la sección de **Favoritos** se modificaron los archivos de **views.py**, **repositories.py** y **services.py** correspondientes a las funciones que lo hagan funcionar (valga la redundancia). Los códigos en cuestión son:

Para views.py:

```
@login_required
def getAllFavouritesByUser(request):
    favourite_list = repositories.getAllFavourites(request.user)
    return render(request, 'favourites.html', { 'favourite_list': favourite_list })
```

“@login_required” condiciona a la función para ser utilizada sólo cuando hay un usuario que haya iniciado sesión. La función toma como parámetro a `request`. En la misma se crea una lista a partir de la función `getAllFavourites` desde el archivo de `repositories.py`, tomando como parámetro a `request.user`, es decir, los favoritos del usuario. Finalmente, retorna un render en el que se pueden ver los favoritos de la lista en pantalla.

```
@login_required
def saveFavourite(request):
    if request.method == 'POST':
        try:
            fav = translator.fromTemplateIntoCard(request)  # Se transforma el request en una "Card".
            fav.user = request.user  # Asignamos el usuario correspondiente.

            # Guardamos el favorito
            repositories.saveFavourite(fav)

            # Redirigimos a la página de favoritos después de guardar
            return redirect('favoritos')

        except Exception as e:
            # Si hay un error, puedes manejarlo y devolver una respuesta adecuada
```

```
            return HttpResponse(f"Error al guardar el favorito: {str(e)}", status=500)

        else:
            # Si no es un POST, puedes devolver una respuesta adecuada
            return HttpResponse("Método no permitido", status=405)
```

Nuevamente, la función está condicionada por el inicio de sesión. Luego, la función encargada de guardar a los favoritos del usuario comienza con un condicional donde lo que se envía tiene que ser con el método `POST`, el cual es usado para enviar datos al servidor. Allí, se intenta transformar lo enviado, es decir, los datos del favorito, en una card; y se asigna al usuario correspondiente con `request.user`. Posteriormente, se guarda al favorito con la función `saveFavourite` desde el archivo `repositories.py` usando como parámetro a `fav`, es decir, los favoritos transformados en cards. A continuación, se redirige a la pestaña de favoritos de la página a partir de la url ligada al nombre `'favoritos'`. Las últimas líneas del código son para responder en caso de algún tipo de error al intentar guardar el/los favorito/s.

```
@login_required
def deleteFavourite(request):
    if request.method == 'POST':
        # Obtener el ID del favorito a eliminar desde el formulario
        fav_id = request.POST.get('id')

        # Verificar que el ID sea válido
```


```
        if fav_id:
            # Obtener el favorito correspondiente al ID (si no existe, devolver 404)
            favourite = get_object_or_404(Favourite, id=fav_id, user=request.user)

            # Eliminar el favorito
            favourite.delete()

            # Redirigir a la página de favoritos
            return redirect('favoritos')

        return HttpResponse("No se pudo eliminar el favorito.", status=400)

    return HttpResponse("Método no permitido.", status=405)
```

Esta función, también condicionada por el inicio de sesión, lo que hace es eliminar al favorito que se encuentra en la sección homónima. Al igual que la anterior, tiene un condicional dentro ligada al método `POST`. Se crea la variable `fav_id` en la que se pide la id del favorito guardado; luego se la analiza para saber si tiene algo dentro y, con una nueva variable llamada `favourite`, se obtiene el favorito correspondiente a la id, pero en caso de que no exista, devuelve el error 404. Posteriormente, borra el favorito con `favourite.delete()` y redirige a la página en la parte de favoritos usando la url con `return redirect('favoritos')`. Finalmente, ante cualquier error, se cubre con mensajes como "No se pudo eliminar el favorito." o "Método no permitido.".

Para repositories.py:

from app.models import Favourite

Desde la parte de modelos de la carpeta `app` importa la clase `Favourite`.

```
def saveFavourite(image):
    try:
        fav = Favourite.objects.create(url=image.url, name=image.name, status=image.status, last_location=image.last_location, first_seen=image.first_seen, user=image.user)
        return fav
    except Exception as e:
        print(f"Error al guardar el favorito: {e}")
        return None
```

En la variable `fav` de esta función, se guardan los datos que corresponden al favorito que se intenta guardar y retorna a la misma variable. En caso de que no se pueda por cualquier motivo, se imprime `"Error al guardar el favorito: {e}"` y se retorna `None`, es decir, nada.

```
def getAllFavourites(user):
    favouriteList = Favourite.objects.filter(user=user).values('id', 'url', 'name', 'status', 'last_location', 'first_seen')
    return list(favouriteList)
```

Esta función lo que hace es obtener todos los favoritos que guardó el usuario para ponerlos en una variable, la cual luego retornará como lista.

Para services.py:

```
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.
```

Lo primero que hace esta función es, en la variable `fav`, es transformar lo que se da para guardar como favorito (el `request`) en una card, usando la función `fromTemplateIntoCard` desde `translator.py`. Luego se asigna al usuario que quiere guardar al favorito con `get_user(request)`. Finalmente, lo que retorna es que se guardan esos datos en la base con la función `saveFavourite` desde `repositories.py` a partir de la variable `fav`.

```
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.fromRequestIntoCard(favourite) # transformamos cada favorito en una Card, y lo almacenamos en card.

            mapped_favourites.append(card)


        return mapped_favourites
```

Lo que hace esta función es primero ver si el usuario se encuentra logueado/autenticado; en caso de que no, devolverá una lista vacía, pero en caso de que sí lo esté, pedirá al usuario y lo guardará en la variable user. Luego de eso, creará una lista en la que pedirá a todos los favoritos usando como parámetro a la variable `user`, utilizando la función `getAllFavourites` desde `repositories.py`; a continuación, crea una lista vacía en una variable llamada `mapped_favourites`. Posteriormente, analiza la lista de favoritos con la variable `favourite` y transforma cada dato en una card con la función de `translator.py` llamada `fromRequestIntoCard`. Finalmente, en `mapped_favourites`, pone esas cards y retorna la misma variable.

Una vez finalizada la explicación de los códigos, identificamos varios desafíos que, aunque menores, generaron ciertas dificultades durante el desarrollo. Sin embargo, el mayor reto fue que en nuestro código, en lugar de mostrar los primeros episodios de aparición de cada personaje, se desplegaban las URLs que

referenciaban a dichos episodios. Este problema nos llevó bastante tiempo resolver, pero finalmente logramos corregirlo y mostrar correctamente la información deseada, esto lo hicimos mediante llamar a una función externa con todos los datos de la card, en vez de volver generar la card con todos los datos en una función nueva.

Otra cosa que no pudimos resolver es que a una card que ya se encontraba en favoritos, le aparezca un recuadro que indique que ya está en esa sección.

Conclusión:

Durante el desarrollo del trabajo, adquirimos diversos conocimientos y habilidades que resultaron fundamentales para llevarlo a cabo. En primer lugar, aprendimos los conceptos básicos de Git, como realizar commits, trabajar con forks y compartir el código de manera eficiente en equipo. Estos conocimientos nos ayudaron a mantener un flujo de trabajo colaborativo y organizado. También nos adentramos en el uso de Django, comprendiendo su aplicación en la creación de páginas web. Además, exploramos cómo funcionan las APIs, aprendiendo a interactuar con ellas para obtener y procesar datos, así como a integrar esa información en nuestro proyecto.

Lastimosamente por el tiempo, no pudimos implementar otras cosas como la paginación de resultados, añadir comentarios y otros opcionales.

Sin embargo, este aprendizaje nos permitió desarrollar un código más extenso y funcional, entendiendo mejor los desafíos que implica construir aplicaciones más complejas. En cuanto a las dificultades, aunque enfrentamos algunos inconvenientes puntuales, como los mencionados anteriormente, consideramos que el trabajo en general no resultó tan complicado. Las herramientas y tecnologías que utilizamos, junto con el trabajo en equipo, facilitaron bastante el proceso y nos dejaron con una experiencia enriquecedora y satisfactoria.