****

**Introducción a la Programación** **(IP)**

**Trabajo Final**

**Universidad Nacional de General Sarmiento (UNGS)**

**Alumnos**:

* **Joaquín Possi**:

joaquinpossi@gmail.com

* **Santiago Gervasio**:

santiagobenja21@gmail.com

* **Joaquín Romero**:

joanferomero@gmail.com

**Docentes**:

* **Omar Argeñaras**
* **Nancy Mores**

**Comisión**: 04

**Fecha de entrega**: 27-11-2024

Introducción:

Este trabajo consiste en modificar una aplicación web previamente desarrollada en Django, de modo que vuelva a estar en funcionamiento. El proyecto consiste en buscador de imágenes de los personajes de la serie de Ricky & Morty, utilizando una API homónima. En este informe se hará un breve repaso de todo lo que se modificó a la hora de realizar el trabajo. Tanto los cambios obligatorios, de los cuales son fundamentales para el funcionamiento de la, como de las modificaciones opcionales para brincar una mejor experiencia al navegar por la app.

Primeramente, modificamos la función home(), que se encuentra dentro del archivo views.py. Las modificaciones que se realizaron, permiten que la función ahora pueda recoger las imágenes que llegan desde la API, a través de services.getAllImages(), además de también obtener los elementos favoritos del usuario, mediante services.getAllFavourites(), con el propósito de poder insertarlas en el template. Además, implementamos cierta lógica sobre la paginación que retomaremos mas adelante.

def home(request):

pagina = request.GET.get("page","1")

images = services.getAllImages(None,pagina)

favourite\_list = services.getAllFavourites(request)

return render(request, 'home.html', {'images': images, 'favourite\_list': favourite\_list })

Posteriormente pasamos a definir la función services.getAllImages(),lo que posibilita que la función pueda recorrer los datos de la api utilizando transport.getAllImages() (función que trae datos crudos de la api) y transformarlos en imagen para poder mostrarlos en el template correspondiente.

def getAllImages(busqueda,pagina):

json\_collection=transport.getAllImages(busqueda,pagina)

images = []

for image in json\_collection:

images.append(translator.fromRequestIntoCard(image))

return images

Luego modificamos el template de home.html, para que este se modifique en base a los datos provenientes de la API, por ejemplo, si el personajes está vivo, en la card se mostrara un círculo verde, si está muerto se verá un círculo rojo y si no se conoce su estado se verá amarillo.

<strong>

{% if img.status == 'Alive' %}

🔴 {{ img.status }}

{% elif img.status == 'Dead' %}

🔴 {{ img.status }}

{% else %}

🔴 {{ img.status }}

{% endif %}

</strong>

La siguiente modificación hace que la card muestre el primer episodio donde aparece el personaje y la última ubicación vista.

<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>

Ya vistos los cambios obligatorios, seguimos con los opcionales.

Primero arrancamos con el buscador:

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')

Este, mediante la variable search\_msg, lo que hace es agarrar un valor enviado por el usuario, y luego si search\_msg no es un conjunto vacío, se lo da como argumento a la función getAllImages() para que traiga las imágenes que contengan ese argumento en el nombre.

Despues se encuentra el inicio de sesión, el cual es fundamental para después realizar el sistema de favoritos. Para simular un inicio de sesión, utilizamos el usuario admin, que ya está en la base de datos. Para poder salir de ese usuario importamos la función logout() desde django.contrib.auth y la aplicamos a la función exit(). La cuál nos dirige a la página principal después de sacarnos de la cuenta

@login\_required

def exit(request):

logout(request)

return redirect('home')

Bien, ahora seguimos con la implementación de personajes favoritos, para poder implementar esta función primeramente retocamos en services.py la función getAllfavourites() al implementarle repositories.getAllFavourites() con el usuario como argumento y luego transformamos a todos los favoritos del usuario en una card mediante la función fromRepositoryIntoCard()  
  
def getAllFavourites(request):

if not request.user.is\_authenticated:

return []

else:

user = get\_user(request)

favourite\_list = repositories.getAllFavourites(user)

mapped\_favourites = []

for favourite in favourite\_list:

card = translator.fromRepositoryIntoCard(favourite)

mapped\_favourites.append(card)

return mapped\_favourites

Posteriormente para poder guardar los favoritos y que se rendericen en la página, utilizamos la función services.saveFavourite() que obtiene un usuario y transforma en card todos los templates.

def saveFavourite(request):

fav = translator.fromTemplateIntoCard(request)

fav.user = get\_user(request)

return repositories.saveFavourite(fav)

Para la funcionalidad de borrar favoritos utilizamos la función services.deleteFavourite(), que lo que hace es borrar los favoritos mediante su id.

def deleteFavourite(request):

favId = request.POST.get('id')

return repositories.deleteFavourite(favId)

Y por último, el sistema de paginación, para este modificamos en la sección views.py

def home(request):

pagina = request.GET.get("page","1")

images = services.getAllImages(None,pagina)

favourite\_list = services.getAllFavourites(request)

next\_page = str(int(pagina)+1)

return render(request, 'home.html', {'images': images, 'favourite\_list': favourite\_list ,'page': pagina,"next\_page" : next\_page,"prev\_page" : str(int(pagina) - 1)})

En este código, con la variable page obtenemos el valor de la página.

Después al momento de renderizarlo, utilizamos next\_page y prev\_page, que lo que hace es pasarlo a entero y al valor que nos devuelva page le suma o resta 1 en base a la acción que quisiéramos realizar.

En el template de home.hmtl se realizaron los cambios correspondientes para el correcto funcionamiento del paginado

<ul class="pagination">

<li class="page-item {% if prev\_page<'1' %}oculto{% endif %}">

<a class="page-link " href="?page={{prev\_page}}">⬅️</a>

</li>

<li class="page-item {% if page == '1' %} active {% endif %}" aria-current="page">

<a class="page-link " href="?page=1">1</a>

</li>

<li class="page-item {% if page == '2' %} active {% endif %}">

<a class="page-link " href="?page=2">2</a>

</li>

<li class="page-item {% if page == '3' %} active {% endif %}">

<a class="page-link" href="?page=3">3</a>

</li>

<li class="page-item ">

<a class="page-link" href="?page={{ next\_page }}">➡️</a>

</li>

</ul>

En este template modificamos link de referencia con los números de las páginas correspondientes, Ejemplo:

<a class="page-link " href="?page=1">1</a>

Cada vez que se intente ir a la página anterior o a la posterior mediante

<li class="page-item ">

<a class="page-link" href="?page={{ next\_page }}">➡️</a>

o

<a class="page-link " href="?page={{ prev\_page }}">⬅️</a>

Se activa el sistema de paginación previamente visto. Para concluir esta parte agregamos restricciones para que no se pueda retroceder si la página es la numero 1

<li class="page-item {% if prev\_page == '0' %} disabled xd {% endif %}">

Conclusión:

En este trabajo se retocaron y ajustaron muchas funciones claves, que nos permitieron poner en práctica conocimientos previos al mismo tiempo que veíamos lenguajes que no habíamos visto previamente en clase. El mayor desafío que se nos presento era adaptarnos a un framework como lo es Django, que si bien no era difícil, era algo nuevo. Pero que a través de las prácticas fuimos comprendiendo mejor.