Introducción a la Programación

Com.: Virtual - 2do semestre del 2024

Docentes: Godoy Gonzalo, Montiel Santiago, Rial Jorgelina y Ruiz Yair.

Grupo 6: ***Delgado Solange Edith, Lema Jacinto, Michael Anthony y Quintana, Axel Brian.***

Informe

Trabajo Práctico Grupal

Introducción

El presente trabajo consiste en explicar el desarrollo de una aplicación web usando un entorno de trabajo (framework) llamado “Django” utilizando una API (o interfaz de programación de aplicaciones) homónima para la búsqueda de imágenes de los personajes de la serie **Rick & Morty**. Para esto, la información proveniente de esta API será transformada por el framework mencionado en distintas “*cards” (tarjetas)* que mostrarán la imagen del personaje, su nombre y apellido, el estado (alive, dead, unknown), la última ubicación y el episodio inicial. Además, será posible visualizar 20 cards (limite) por página siendo en total tres.

A continuación se procederá a explicar el código de las funciones implementadas y una breve explicación de cada una de ellas junto con las dificultades de implementación y decisiones tomadas.

Códigos de las funciones implementadas

Puntos Obligatorios

Views.py

home(request): en esta función decidimos cambiar las listas “images” y “favourite_list” dadas, las cuales estaban vacías y a cada una de ellas decidimos asignarle llamados a

una función determinada. Por ejemplo: para “images= services.getAllImages()”que llama la función obtener todas las imágenes y para “favourite_list=services.getAllFavourites(request)”. En la última línea decidimos no modificarla, de esta manera la función quedo completada.

Para no tener *dificultades en la implementación* es necesario no olvidarse de realizar estas modificaciones en las funciones que le siguen y llaman a estas funciones o a las necesarias. Por lo tanto después de la línea:

- ✓ “if (search_msg != ''):” en la función search (request) le adicionamos

images = services.getAllImages(search_msg) # llamamos a la funcion de obtener todas las imagenes usando el input del cuadro de busqueda

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

y luego quedaría el “else:” y todo lo demás sin modificación.

- ✓ “def getAllFavouritesByUser(request):” solo le adicionamos

favourite_list = services.getAllFavourites(request) # llamamos a la funcion de obtener todos los favoritos de un usuario para mostrarlos en otra plantilla.

El resto del código queda igual.

- ✓ def saveFavourite(request):Modificamos

services.saveFavourite(request) # llamamos a la funcion guardar favoritos desde servicios

return redirect('home') # redireccionamos a la pagina 'home' para seguir viendo las imágenes

- ✓ def deleteFavourite(request):Agregamos

services.deleteFavourite(request) # llamamos a la funcion eliminar favoritos

return redirect('favoritos') # redireccionamos a la pagina de favoritos para seguir viendo el listado de favs

- ✓ def exit(request):aquí cambiamos

logout(request) # llamamos a la funcion logout para desloguear

return redirect('login') # redireccionamos a la pagina de 'login' una vez deslogueado

Básicamente, lo que hicimos en esta sección fue modificar las funciones de manera tal que se llame a las funciones pertinentes.

Services.py

getAllImages(input=None): Lo primero que decidimos fue *agregar el* “from ..transport import transport” puesto que es necesario para en la función “def getAllImages(input=None=” para obtener los datos de la API por lo tanto adicionamos allí un “json_collection = transport.getAllImages(input)”. Luego, en la sección comentada “recorre cada dato crudo de la colección anterior, lo convierte en una Card y lo agrega a

images.” abajo de la lista vacía de “images” coincidimos que era conveniente utilizar un “for” para recorrer los datos obtenidos y sumarlo a dicha lista en formato de card haciendo uso de translator (“for object in json_collection: images.append(translator.fromRequestIntoCard(object))”). Más adelante, para añadir favoritos (usado desde el template 'home.html') en la función “saveFavourite(request)” transformamos un request del template en una Card mediante “fav = translator.fromTemplateIntoCard(request)” y además le asignamos el usuario correspondiente, de la siguiente manera: “fav.user = get_user(request)”. Finalmente, en la función “getAllFavourites(request)” que lo utiliza desde el template 'favourites.html' adicionamos después de la línea “user = get_user(request)” la búsqueda desde el repositories.py **todos** los favoritos del usuario (variable 'user')y creaos una lista vacia llamada “mapped:favourites” y por ultimo después del for que recorre “favourite in favourite_list” a la variable card la transformamos en cada favorito en una card y la guardamos en card (“card = translator.fromRepositoryIntoCard(favourite)) y con lo que resta del código no lo cambiamos ya que no se presentan dificultades en la implementación.

Home.html

***Card*:** debe cambiar su border color dependiendo del estado del pensonaje. Si está **vivo** (alive), mostrará un borde verde; si está **muerto** (dead) mostrará rojo y si es **desconocido** (unknown), será naranja.

Claramente, en esta sección no podemos dejar de pensar la solución con condicionales, lo que modificamos/agregamos en el código fue colocar por una parte, para el bloque de borde de tarjeta, luego de , el condcional if y comparar si la imagen (img.status) correspondia a alive (verde), después de la línea ya utilizamos el condición elif si esta correspondia a dead(rojo) y en caso que no ocurra ninguna de las dos el condicional else para unknown(naranja) adiconado después de la línea “<div class="card mb-3 ms-5

border-danger" style="max-width: 540px;">” después agregamos las líneas y {% endif %} para finalizar las condiciones de esta sección.

Por otra parte, para que no se presenten dificultades en la implementacion en el bloque de estado del personaje también lo cambiamos los condicionales para que el circulo cambie de color dependiendo del estado estado y aquí nuevamente utilizamos los mismos condicionales (if, elif, else)con las mismas relaciones, quedando la parte modificada de la siguiente forma: “(…)

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

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

{% else %} □ {{ img.status }}

{% endif %}

 (…)”

Finalmente, el resto del código lo dejamos tal cual y se ha podido llegar a lo pedido.

Conclusión

El presente trabajo se trato de desarrollar una aplicación web usando un framework llamado “Django” utilizando una API la cual brinda información que es transformada en cards que muestran los personajes de Rick & Morty, se nos otorgó un código el cual faltaba completarlo, para la sección *Views.py* a grandes rasgos lo logramos completar reemplazando las listas vacías con llamadas de funciones necesarias a cada una de las funciones sin estas modificaciones se tendrían dificultades para su implementación. En el apartado que solicitaba completar la parte de *Services.py* hemos notado que era necesario introducir un “from ..transport import transport”, pisar algunas variables como por ejemplo, images con llamadas de funciones, utilizar un json_collection para así poder recorrerlo con un for, entre otras. Para *Home.html,* hemos observado que eran necesarios los condicionales (if, elif, else) para los requerimientos pedidos (colores) tanto en la línea bloque de borde de tarjeta y en el en el bloque de estado del personaje. Finalmente, podemos concluir que hemos podido realizar los puntos obligatorios sin errores de implementación.