# Trabajo práctico
# Introducción  a la programación

Alumnos:

- Alvarez Magali
- Cameroni Sebastian

Docentes:

- Mieres Jose Luis
- Rodriguez Miguel
- Santa Cruz Sergio

Segundo Cuatrimestre 2024
UNGS

# Informe Buscador Rick and Morty

Como trabajo final de *Introducción a la programación* tuvimos que implementar una aplicación web usando Django. Esta aplicación debe permitirnos acceder a los personajes de la serie *Rick & Morty.* En el presente informe vamos a describir el proceso que llevamos a cabo para poder completar los códigos para que la aplicación funcione correctamente.

Una vez que nos entregaron la guía para realizar el trabajo práctico, comenzamos siguiendo una serie de pasos que nos permitían acceder a la aplicación. Gran parte de la aplicación ya estaba resuelta, a nosotros nos correspondía implementar las funcionalidades más importantes.

Antes de empezar a escribir código tuvimos que entrar a cada plantilla para tratar de entender el código que había en ellas y cómo funcionaban. Aunque los profesores nos habían explicado el propósito de algunas, tuvimos que verlas una por una para entender cómo se relacionaban entre sí. Por ejemplo, en una plantilla se generaba un código que servía para complementar a otra y así. Un aspecto que resultó confuso fue que las funciones tienen igual nombre. Por ejemplo, la función *getAllImages* está definida en repositories.py y en services.py. Si bien entendíamos que debíamos usar una función dentro de la otra, fue confuso ver cómo hacíamos esto, hasta que notamos en los demás códigos cómo llamaban funciones que estaban definidas en otra plantilla.

- Código de funciones implementadas

```python
def getAllImages(input=None):
    # obtiene un listado de datos "crudos" desde la API, usando a transport.py.
    json_collection = transport.getAllImages(input) #Función getAllImages de la carpeta transport (trae

    # recorre cada dato crudo de la colección anterior, lo convierte en una Card y lo agrega a images.
    images = []

    for o in json_collection:
        card = translator.fromRequestIntoCard(o) #Convierte los objetos de la variable json_collection
        if card: #Si existe una card.
            images.append(card) #Agrega la card en la lista images.

    return images
```

Empezamos en la carpeta *services.py* donde encontramos la función *getAllImages* con el parámetro de entrada (input). En la función había dos variables que contenían listas vacías (*json_collection* e *images*) y retornaba la variable *images*. En la variable *json_collection* hay un comentado que nos indica:  # Obtiene un listado de datos "crudos" desde la API, usando a transport.py.
Es así que nos dirigimos a la carpeta *transport*.py y encontramos la función *getAllImages* que se encarga de traer los objetos crudos desde la API y los almacena en la variable *json_collection* y retorna esta misma variable.
Volvimos a la carpeta *service.py*, importamos todo lo que hay en la carpeta *transport.py* con el siguiente import: `from ..transport import transport`. Luego, modificamos la variable json_collection con el código `transport.getAllImages(input)`, el cual llama a la función *getAllImages* de *transport.py*  y realizamos un ciclo for para iterar los datos de esta variable. Lo que ocurre en este ciclo es que para cada iteración llamamos a la función

*fromResquestIntoCard,* que ya estaba desarrollada en *translator.py.* Esta función se encarga de transformar la información en formato de tarjetas. Esto lo almacenamos en la variable que creamos llamada *card*, luego de ello realizamos un simple condicional para consultar si existe la variable card (esto es por las dudas que se genere algún error) y guardamos la información en la variable images usando la función append.
Para completar la función existe un *return* de la variable *images* que en conclusión retorna todos las imágenes en forma de tarjetas.
Luego de completar la función nos dirigimos al módulo *views.py*, en la función *home* y completamos la variable *images* llamando a la función *getAllimages* del módulo *services.py* con el parámetro *request*. Luego de esto, se realiza un *return* con el *render* donde carga en '*home.html*' si es que encuentra el campo *'images'* con la variable images que completamos.

```python
# 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 = services.getAllImages()                    #Función getAllImages que retorna una lista con las card.
    favourite_list = services.getAllFavourites(request)     #función getAllFavourites de services. esta función retorna los favs del usuario en formato card.

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

Para terminar las funcionalidades básicas para la aprobación del trabajo solo nos quedaba que las *cards* cambien su borde de color dependiendo del estado del personaje, para esto investigamos el template *'home.html´* indicado en el readme. Modificamos el siguiente código.

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

Inicialmente, en el lugar de *img.status* decía *TRUE*; esto hacía que el condicional no funcionara. En su lugar colocamos *img.status*, que leía el estado del personaje en la carpeta.

Por otro lado, para adicionar al trabajo, implementamos los opcionales: Buscador, Inicio de Sesión y Favoritos.

Vamos a comenzar contando cómo completamos la función para el buscador. Dentro del módulo *views.py* ya había definida una función llamada *search*. Si bien la estructura ya estaba armada, faltaba completar el condicional.

```python
def search(request):
    search_msg = request.POST.get('query', '')

    # si el texto ingresado no es vacío, trae las imágenes y favoritos desde services.py,
    # y luego renderiza el template (similar a home).
    if (search_msg != ''):
        images = services.getAllImages(input=search_msg) #Usa la función getAllImages de la carpeta service usando
        #como parametro la variable search_msg.
        return render(request, 'home.html', {'images': images})
    else:
        return redirect('home')
```

Nosotros queremos que si *search_msg* es distinto de vacío nos devuelva todos los personajes que coinciden con el parámetro ingresado. Para ello, definimos la variable *images*. Esta variable trae desde el módulo *services.py* la función *getAllImages* que se encarga de traer una lista con las *cards*. El parámetro de esta función será *search_msg*, así podemos filtrar la búsqueda según el *search_msg*. Luego retorna en la plantilla *home.html* las imágenes según la búsqueda. En caso contrario, si *search_msg* está vacío nos redirige al *home.*

En cuanto al inicio de sesión, en un principio creímos que debíamos crear una función para poder iniciar sesión. Luego, nos dimos cuenta de que esto ya estaba implementado y lo que debíamos hacer era implementar una función para cerrar sesión. Para ello, en la carpeta *views.py* vimos que ya estaba importada la función *logout* de Django, con el siguiente código: `from django.contrib.auth import logout.` Con este pudimos completar la función *exit*, definida en *views*.

```python
@login_required
def exit(request):
    logout(request)                         #Función de Django para cerrar sesion
    return render(request, 'index.html')    #Vuelve al index.html
```

Dentro de esta función, llamamos a la función de Django para cerrar sesión. Luego, nos retorna a *index.html.*

Por último, vamos a describir cómo implementamos el opcional Favoritos.
Para añadir a favoritos comenzamos completando la función *saveFavourite. En* ella teníamos un código con indicaciones para orientarnos con el código a completar. La sugerencia era, transformar un request del template en una Card. Por lo tanto, llamamos a la función *fromTemplateIntoCard* definida en *translator,* esta función se encargaba de hacer esto. Luego, en otra variable *fav.user*, asignamos el usuario correspondiente con el código *request.user*, este código se encarga de representar al usuario que actualmente se encuentra autenticado en la aplicación. El return ya estaba completo, se llama a la función *saveFavourite*, definida en *repositories*. Esta función se encarga de guardar un favorito en la base de datos. En este caso, el parámetro fue *fav,* la variable definida en la función.

```python
# añadir favoritos (usado desde el template 'home.html')
def saveFavourite(request):

    fav = translator.fromTemplateIntoCard(request) # transformamos un request del template en una Card.
    fav.user = request.user # le asignamos el usuario correspondiente.

    return repositories.saveFavourite(fav) # lo guardamos en la base de datos.
```

Por último, para terminar esta funcionalidad, en el módulo *views.py,* completamos la función *saveFavourite.* En ella, definimos una variable llamada *fav* que se encarga de guardar el personaje en la base de datos. Esto fue posible llamando a la función *saveFavourite* desde *services.py.*

```python
@login_required
def saveFavourite(request):
    fav = services.saveFavourite(request) #Desde services traigo la función que me permite guardar
                                          #el personaje en la base de datos.
    return redirect('home')  #Lo redirigo a la plantilla de home.
```

Por otro lado, para mostrar los favoritos de un usuario necesitamos completar un código en *services.py* y otro en *views.py.* En *services.py* contábamos con la estructura de una función *getAllFavourites*, la cual estaba incompleta. En caso de que el usuario no esté identificado, esta función retorna una lista vacía. En caso contrario, en la variable *user* almacena la identificación del usuario. Luego, en la variable *favourite_list* almacena todos los favoritos del usuario. A este listado de favoritos accede llamando a la función *getAllFavourites* definida en *repositories.py*. Luego, recorre por elemento la lista de favoritos del usuario, a cada uno los transforma en una card y los agrega a la lista *mapped_favourites,* que definimos inicialmente vacía. Para transformar a cada favorito en una card, llamamos la función *fromRepositoryIntoCard* desde *translator.py.* Si bien en el módulo *translator.py* habían distintas funciones para transformar la información en *card*, elegimos esta función ya que seguimos la sugerencia que decía que se usaba desde el *template ´favorites.html´.* Por último, la función retorna a todos los favoritos en formato *card*.

```python
# usados desde el template 'favourites.html'
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: #recorro cada favorito del usuario.
            card = translator.fromRepositoryIntoCard(favourite) # transformamos cada favorito en una
                                                                #Card, y lo almacenamos en card.
            mapped_favourites.append(card) #a cada favorito en formato card lo agrego a
                                           #la lista mapped_favourites.


        return mapped_favourites
```

Para terminar de implementar esta función, desde *views.py* también completamos un código. Este código en un principio tenía una lista vacía, nosotros cambiamos esto y decidimos que esta lista sea la lista de favoritos del usuario. Para ello, trajimos desde *services.py* la función *getAllFavourites* que comentamos anteriormente. Luego, la función va a retornar en la plantilla *favourites.html,* todos los favoritos del usuario.

```python
@login_required
def getAllFavouritesByUser(request):
    favourite_list = services.getAllFavourites(request) #traigo la función que retorna la lista de favs
                                                        #del usuario desde services.
    return render(request, 'favourites.html', { 'favourite_list': favourite_list }) #Lo muestro en el templates favourites.html
```

Finalmente, para eliminar un favorito tuvimos que completar el código en el módulo *views.py.* Si bien la función estaba definida, no hacía nada. Nosotros definimos la variable *delete*, en ella llamamos a la función *deleteFavourite*. Esta función está definida en *services.py* y se encarga de borrar un favorito por su ID.

```python
@login_required
def deleteFavourite(request):
    delete = services.deleteFavourite(request) #Desde services traigo la función que me
                                               #permite borrar un favorito por su ID.
    return redirect('/favourites/') #Lo redirigo a la plantilla de favoritos.
```

De esta forma, implementamos una aplicación web que nos permite buscar imágenes de los personajes de la serie *Rick & Morty,* filtrar la búsqueda, iniciar sesión y añadir a favoritos a distintos personajes.