Universidad Nacional de General Sarmiento

Trabajo Final

Introducción a la Programación

Comisión: 01 intensivo de verano 2026

Profesores:

Gonzalo Godoy

Yair Ruiz

Alumnos:

Aldana, Lisette (listhv16@gmail.com)

Luna, Christian (chrisitanezequielluna@gmail.com)

Vergara, Benjamín (benjagunz11@gmail.com)

Este trabajo consiste en una aplicación web que permite a un usuario interactuar con un conjunto de personajes de la serie animada *Los Simpsons*. La aplicación muestra una galería de personajes, permite buscar personajes por nombre, filtrar por estado (por ejemplo “vive” o “muerto”), y además guarda y elimina personajes favoritos de cada usuario.

La idea detrás de esta aplicación es practicar cómo interactuar con datos que vienen desde un servicio externo o una fuente de datos, presentarlos al usuario a través de páginas web, y permitir que el usuario personalice su experiencia guardando o eliminando favoritos.

Código de las funciones implementadas y su explicación

A continuación se detalla cada función de la capa de presentación (es decir, las que reciben y responden a las acciones del usuario), junto con una breve explicación:

`index_page(request)`

```
def index_page(request):
    return render(request, 'index.html')
```

Explicación:
Esta función muestra la página principal de bienvenida o bien una pantalla inicial cuando un usuario ingresa a la aplicación sin estar aún en la sección principal.

Justificación:
Es útil para tener una pantalla de entrada antes de que el usuario interactúe con la parte principal de la aplicación.

home(request)

```
def home(request):
    favourite_list = []
    if request.user.is_authenticated:
        favourite_list =
Favourite.objects.filter(user=request.user)
    return render(request, 'home.html', {
        'images': services.getAllImages(),
        'favourite_list': favourite_list})
```

Explicación:
Esta función obtiene todos los personajes disponibles y la lista de favoritos del usuario autenticado para mostrarlos en la galería principal.

Justificación:
Permite personalizar la vista con las imágenes de personajes y mostrar cuáles ya han sido marcados como favoritos por el usuario.

search(request)

```
def search(request):
    search_msg = request.POST.get('query', '')
    if not search_msg:
        return redirect('home')
    images = services.filterByCharacter(search_msg)
    favourite_list = services.getAllFavourites(request)
    return render(request, 'home.html', {
        'images': images,
        'favourite_list': favourite_list})
```

Explicación:
Busca personajes por nombre basado en lo que el usuario escribió. Si no se puso nada, vuelve a la galería principal.

Justificación:
Brinda funcionalidad de búsqueda para que el usuario encuentre personajes que coincidan con el texto que ingresó.

filter_by_status(request)

```
def filter_by_status(request):
```

```
    status = request.POST.get('status')
    images = services.getAllImages()
    personajes_filtrados = []
    for personaje in images:
        if personaje.status.lower() == status.lower():
            personajes_filtrados.append(personaje)
    favourite_list = []
    if request.user.is_authenticated:
        favourite_list =
Favourite.objects.filter(user=request.user)
    return render(request, 'home.html', {
        'images': personajes_filtrados,
        'favourite_list': favourite_list})
```

Explicación:
Filtra la lista de personajes por su estado (por ejemplo “alive” o “dead”), basado en lo que el usuario seleccionó.

Justificación:
Permite que el usuario reduzca los personajes a mostrar según su estado, haciendo la experiencia más personalizada.

getAllFavouritesByUser(request)

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

Explicación:
Obtiene y muestra la lista de personajes que el usuario ha marcado como favoritos.

Justificación:
Necesario para dar una vista dedicada al usuario donde vea solo sus favoritos.

saveFavourite(request)

```
@login_required
def saveFavourite(request):
```

```
    services.saveFavourite(request)
    return redirect('home')
```

Explicación:
Guarda un personaje como favorito para el usuario y vuelve a la pantalla principal.

Justificación:
Permite que usuarios registren nuevos favoritos de forma rápida y sin recargar manualmente.

`deleteFavourite(request)`

```
@login_required
def deleteFavourite(request):
    services.deleteFavourite(request)
    return redirect('home')
```

Explicación:
Elimina un personaje de favoritos.

Justificación:
Control de favoritos para que el usuario pueda quitar lo que ya no desea en su lista.

`exit(request)`

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

Explicación:
Cierra la sesión del usuario y vuelve a la galería sin usuario.

Justificación:
Permite que el usuario termine su sesión en la aplicación de forma simple.

Conclusión:

El desarrollo de este trabajo permitió construir una aplicación interactiva donde el usuario puede visualizar personajes de *Los Simpsons*, buscarlos por nombre, filtrarlos según su estado y administrar una lista personal de favoritos. A lo largo del proyecto se organizaron claramente las responsabilidades para que cada parte cumpliera una función específica.

Por un lado, las rutas permiten que cada acción del usuario esté correctamente conectada con la respuesta adecuada dentro del sistema. Gracias a ellas, la aplicación puede interpretar qué quiere hacer el usuario y dirigirlo a la sección correspondiente.

Por otro lado, las vistas se encargan de recibir esas acciones y devolver una respuesta visible en pantalla. Son el punto de contacto entre el usuario y el sistema. Sin embargo, el procesamiento de la información no se realiza directamente allí, sino que se delega en los servicios.

Los servicios concentran la lógica del trabajo con los datos: obtener personajes, aplicar búsquedas, filtrar información y administrar favoritos. Esta separación fue una decisión importante, ya que permite mantener el código ordenado, evitar repeticiones innecesarias y facilitar futuras mejoras.

En conjunto, la estructura adoptada demuestra una organización clara del proyecto, donde cada parte tiene una función definida. Esto no solo mejora la comprensión del código, sino que también facilita su mantenimiento y expansión en el futuro.

En conclusión, el trabajo no solo cumple con las funcionalidades solicitadas, sino que también presenta una estructura coherente y bien justificada, permitiendo una experiencia clara para el usuario y una implementación ordenada desde el punto de vista del desarrollo.