Informe del Trabajo Práctico

Introducción

El trabajo práctico tiene como objetivo desarrollar una aplicación web basada en la serie *Rick & Morty*, que permita a los usuarios explorar personajes a través de una galería interactiva. Los usuarios pueden buscar personajes, agregar favoritos, y administrar sus listas personales. La aplicación integra diversas funcionalidades, como la visualización de personajes según su estado, la búsqueda con filtros y la gestión de favoritos, garantizando una experiencia dinámica e intuitiva.

Desarrollo

A continuación, se describen las funciones implementadas, su propósito y las decisiones tomadas durante su desarrollo.

Funciones Implementadas

Funciones en `views.py`

- `home(request)`
  - **Propósito:** Renderiza la galería principal mostrando imágenes obtenidas de la API y la lista de favoritos del usuario si está logueado.
  - **Código:**

```
def home(request):
    images = services.getAllImages()
    favourite_list = services.getAllFavourites(request)

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

  - **Dificultades:** Manejar casos en los que el usuario no esté logueado y garantizar que la lista de favoritos se muestre correctamente.

- **search(request)**
  - **Propósito:** Filtrar imágenes de personajes según el texto ingresado en el buscador.
  - **Código:**

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

  - **Decisiones:** Implementar un control para mostrar todos los personajes en caso de búsquedas vacías.
  - **Dificultades:** Asegurar que el flujo sea intuitivo y funcional tanto para búsquedas vacías como para resultados inexistentes.

- **getAllFavouritesByUser(request)**
  - **Propósito:** Recupera la lista de favoritos del usuario logueado y los renderiza en la plantilla favourites.html.
  - **Código:**

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

  - **Decisiones:** Se verificó la autenticación del usuario con el decorador @login_required antes de procesar la solicitud.

    Se centralizó la lógica de obtención de datos en el servicio services.getAllFavourites.

  - **Dificultades:** Asegurar que los datos de favoritos se pasen al template en un formato consistente y utilizable.

- **saveFavourite(request)**
  - **Propósito:** Guarda un elemento como favorito en el perfil del usuario logueado y redirige a la página principal.
  - **Código:**

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

- **Decisiones:** Delegar la lógica de negocio al servicio services.saveFavourite para mantener la función controladora ligera.

  Utilizar redirect('home') para optimizar la experiencia del usuario tras guardar el favorito.

- `deleteFavourite(request)`
  - **Propósito:** Elimina un elemento favorito del perfil del usuario logueado mediante una solicitud POST.
  - **Código:**

```
@login_required
def deleteFavourite(request):
    if request.method == 'POST':
        services.deleteFavourite(request)
    return redirect('favoritos')
```

  - **Decisiones:** Restringir la eliminación solo a solicitudes POST para prevenir acciones no deseadas.

    Centralizar la lógica de eliminación en el servicio services.deleteFavourite.

  - **Dificultades:** Garantizar la seguridad del proceso de eliminación y proporcionar retroalimentación visual al usuario.

- `exit(request)`
  - **Propósito:** Cierra la sesión del usuario logueado y redirige a la página de inicio de sesión.
  - **Código:**

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

  - **Decisiones:** Usar la función logout de Django para gestionar de manera segura el cierre de sesión.

    Redirigir al usuario a la página de inicio de sesión para facilitar una nueva autenticación.

Funciones en `services.py`

- **`getAllImages(input=None)`**
  - **Propósito:** Obtiene un listado de imágenes desde la API y las convierte a un formato estándar (*Card*).
  - **Código:**

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

    return images
```

  - **Decisiones:** Utilizar una estructura de lista para transformar los datos crudos en objetos utilizables en los templates.
  - **Dificultades:** Validar la integridad de los datos para evitar errores en el mapeo.

- **`saveFavourite(request)`**
  - **Propósito:** Guarda un personaje como favorito del usuario actual.
  - **Código:**

```
def saveFavourite(request):
    fav =  translator.fromTemplateIntoCard(request)
    fav.user = get_user(request)

    return repositories.saveFavourite(fav)
```

  - **Dificultades:** Asegurar que no se dupliquen favoritos y manejar errores de persistencia.

- **`getAllFavourites(request)`**
  - **Propósito:** Recupera la lista de favoritos del usuario logueado.
  - **Código:**

```
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
```

    - **Dificultades:** Mapear correctamente los datos desde la base de datos a objetos útiles en el template.
- **`deleteFavourite(request)`**
    - **Propósito:** Eliminar el favorito de un determinado ID del repositorio.
    - **Código:**

```
def deleteFavourite(request):
    favId = request.POST.get('id')
    return repositories.deleteFavourite(favId)
```

    - **Decisiones:** Centraliza la logica de la eliminacion en repositories.deleteFavourite(favId).

Modificaciones en `home.html`

- **Propósito:** Mostrar las tarjetas de personajes con un borde que indique su estado (`Alive`, `Dead` o `Unknown`).
    - **Código:**

```
<div class="card mb-3 ms-5 {% if img.status == 'Alive' %} border-alive {% elif img.status == 'Dead' %} border-dead {% else %} border-unknown {% endif %}"  style="max-width: 540px;">
```

- **Decisiones:** Usar condicionales en el template para simplificar el control visual del estado.
- **Dificultades:** Integrar la lógica de estilos con los datos dinámicos provenientes de la API.