Proyecto Pokedéx API (Python + Django)

Introducción a la Programación - Comisión 02

Alumnos:
Florencia Altamirano
Luis Cubilla
Alexandro Lucero

Profesores:
Bottino, Flavia
Bidart Gauna, Lucas

Introducción

Este trabajo consistió en completar una aplicación que ya estaba parcialmente desarrollada. La idea principal era mostrar una galería de personajes Pokémon y permitir al usuario interactuar con esa información de distintas maneras.

Cuando abrimos la aplicación, debía verse un listado de tarjetas, cada una con los datos principales de un personaje: su nombre, imagen, altura, peso, el nivel base al que aparece, y los tipos a los que pertenece. Sin embargo, muchas de estas funciones no estaban terminadas, por lo que el objetivo fue hacer que la aplicación funcione correctamente.

Además de mostrar esa galería, se pidió que el usuario pudiera buscar personajes por su nombre, filtrar por tipo (como fuego, agua o planta), y que las tarjetas cambiaran de color según ese tipo. También se sugirió, como una mejora, agregar un sistema de favoritos para usuarios registrados.

El trabajo implicó revisar lo que ya estaba hecho, entender cómo estaba organizada la aplicación y agregar lo que faltaba para cumplir con los requisitos. Uno de los principales desafíos fue recorrer la gran cantidad de archivos que tenía el proyecto y comprender cuáles debíamos modificar y cuáles ya contenían funciones útiles que podíamos aprovechar. Además, desde algunas computadoras no se podía acceder correctamente a la información porque la conexión con la fuente de datos (API) fallaba; en esos casos, fue necesario hacer ajustes en la configuración de red para poder continuar.

Una vez superados esos primeros obstáculos, se trabajó en los ajustes visuales, los filtros por tipo y el buscador. En el desarrollo hubo dificultades, especialmente al comienzo, cuando no estaba claro con qué funciones empezar para mostrar la información, luego al implementar los filtros, y también resultó complejo hacer que las tarjetas cambiaran de color según el tipo del personaje. A pesar de estas dificultades, todos los problemas se resolvieron y se avanzó con éxito en la implementación de las funcionalidades requeridas.

Funciones Implementadas

La función **getAllImages()** dentro de **services.py** obtiene un listado de imágenes y datos de la API que devuelve en forma de tarjetas. Entender la razón de esta función fue nuestro primer desafío a la hora de abarcar el proyecto ya que para eso, primero había que entender cómo se obtenía la API y cómo se transforma a formato JSON. Dicho proceso se lleva a cabo en el archivo **transport.py** en la función **getAllImages()**. Luego, la devolución en forma de tarjetas se da gracias a la función **fromRequestIntoCard(poke_data)** dentro de **translator.py**, la cual se encarga de extraer y organizar los datos que nos interesan del Pokémon (incluyendo la imagen) y los mete en un objeto **Card**:

```
# función que devuelve un listado de cards.
# cada card representa la información de la API de cada Pokemon
def getAllImages():
    jsonImages = transport.getAllImages()
    cards = []
    for pokemon_data in jsonImages:
        card = translator.fromRequestIntoCard(pokemon_data)
        if card:
            cards.append(card)
    return cards
```

Sin embargo, todavía no se podían ver los Pokémon en la galería (home.html), para eso fue necesario desarrollar la función **home(request)** dentro de **views.py**, gracias a la devolución de los profesores y la orientación sobre implementar la siguiente función que obtenga los datos e imágenes de la api y devuelva una lista de los mismos para utilizar luego en otra función:

```
def getAllImageAndFavoriteList(request):
    images = services.getAllImages()
    return images
```

Luego, implementamos las sugerencias para la función de **home(request)**, para que en la galería se muestre el listado de Pokémons con sus respectivas tarjetas y los favoritos en caso de que el usuario haya iniciado sesión. Esta nueva función que trabaja con la función creada anteriormente para mostrar la información de la base de datos:

```python
# esta función obtiene 2 listados: uno de las imágenes de la API y otro de favoritos
# ambos en formato Card, y los dibuja en el template 'home.html'.
def home(request):
    images = getAllImageAndFavoriteList(request)
    try:
        favourite_list = getAllFavourites(request)
    except (ImportError, AttributeError):
        favourite_list = []
    return render(request, 'home.html', { 'images': images, 'favourite_list': favourite_list })
```

Además, implementamos las funciones que se nos sugirieron añadir para que el usuario pueda operar con el listado de favoritos en la interfaz, esto fue exclusivamente sugerido en la devolución por lo que las implementamos tal cuál nos fueron sugeridas y funcionaron correctamente.

```python
# añadir favoritos (usado desde el template 'home.html')
def saveFavourite(request):
    fav = translator.fromTemplateIntoCard(request)
    # le asignamos el usuario correspondiente.
    fav.user = get_user(request)
    return repositories.save_favourite(fav)

# usados desde el template 'favourites.html'
def getAllFavourites(request):
    if not request.user.is_authenticated:
        return []
    else:
        user = get_user(request)
        # buscamos desde el repositories.py TODOS los favoritos del usuario (variable 'user').
        favourite_list = repositories.get_all_favourites(user)
        mapped_favourites = []
        for favourite in favourite_list:
            # convertimos cada favorito en una Card, y lo almacenamos en el listado de mapped_favourites.
            card = translator.fromRepositoryIntoCard(favourite)
            mapped_favourites.append(card)

        return mapped_favourites

def deleteFavourite(request):
    favId = request.POST.get('id')
    return repositories.delete_favourite(favId) # borramos un favorito por su Id
```

Lo siguiente era hacer que funcionen los botones que filtran los pokemons según el tipo. Esto resultó complejo ya que surgió un problema, el cuál era que la función que utilizaba nuestra función para filtrar los pokemons no tenía en cuenta que un pokemon puede tener más de un tipo. Para ello tuvimos que hacer modificaciones en la función original y así la función que creamos pudo funcionar y filtrar los pokemons según el tipo de pokemon que son, aludiendo a los tres tipos principales (planta, fuego, agua):

```python
# función utilizada para filtrar por el tipo del Pokemon
def filter_by_type(request):
    type = request.POST.get('type', '')

    if type != '':
        favourite_list = []
        images = filterByType(type) if type else []

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

Para que la función anterior logre hacer un correcto filtro, tuvimos que modificar la siguiente función **filterByType(type_filter)** dentro de **services.py**, que filtra según el tipo de pokemon teniendo en cuenta que puede tener más de un tipo, terminó resultando de la siguiente manera:

```python
# función que filtra las cards según su tipo.
def filterByType(type_filter):
    filtered_cards = []

    for card in getAllImages():
        if type_filter in card.types:
            filtered_cards.append(card)

    return filtered_cards
```

Lo siguiente que realizamos fue investigar html para poder definir los bordes de las tarjetas según los tipos de pokémon que teníamos, haciendo que los de fuego tuvieran un marco rojo, los de agua uno azul, los de planta uno verde y los restantes de color naranja. Esto resultó complejo ya que la mayor parte de nuestro grupo no había visto o aprendido html hasta el momento pero con ayuda de herramientas de inteligencia artificial pudimos desarrollar el siguiente código html:

```
<!— evaluar si la imagen pertenece al tipo fuego, agua o planta —>
{% with tipo=img.types.0|lower %}

    <div class="card mb-3 ms-5"
        style="max-width: 540px;
                border: 3px solid;
                {% if tipo == 'water' %}blue
                {% elif tipo == 'grass' %}green
                {% elif tipo == 'fire' %}red
                {% else %}orange
                {% endif %};">

     <div class="row g-0">···
    </div>
</div>

{% endwith %}
```

Finalmente, se implementó la función para buscar por nombre. Para ello se modificó la función **search(request)**, la cuál ya existía una base en los archivos provistos, para que cuando se ingrese en el formulario de búsqueda un nombre de un pokémon, este responde mostrando la tarjeta del correspondiente:

```
def search(request):
    if request.method == 'POST':
        name = request.POST.get('query', '').lower().strip()

        if name:
            all_cards = getAllImages()
            images = [card for card in all_cards if name in card.name.lower()]
            favourite_list = []

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

    return redirect('home')
```

Cierre

En conclusión, este trabajo resultó muy complejo al principio, pero con la investigación y la orientación de los profesores, logramos comprender el funcionamiento de las funciones involucradas. Se completaron todas las consignas obligatorias e incluso se implementaron algunas funcionalidades adicionales. A lo largo del desarrollo, se aprendió en términos generales cómo es el flujo de trabajo de una aplicación web con Python y Django.

Si bien aún queda mucho por aprender sobre funciones avanzadas, librerías y bases de datos, este proyecto fue una experiencia muy valiosa, ya que permitió afrontar un código ya iniciado y adaptarlo para que funcione correctamente.