Informe de IP: Trabajo Práctico Python

Universidad General Sarmiento

Alumnos: Alvarado Agustin, Espinola Luca, Galizia Mateo, Ayala Dilan

Profesores: Bottino Flavia, Bidart Lucas

Primer semestre 2025

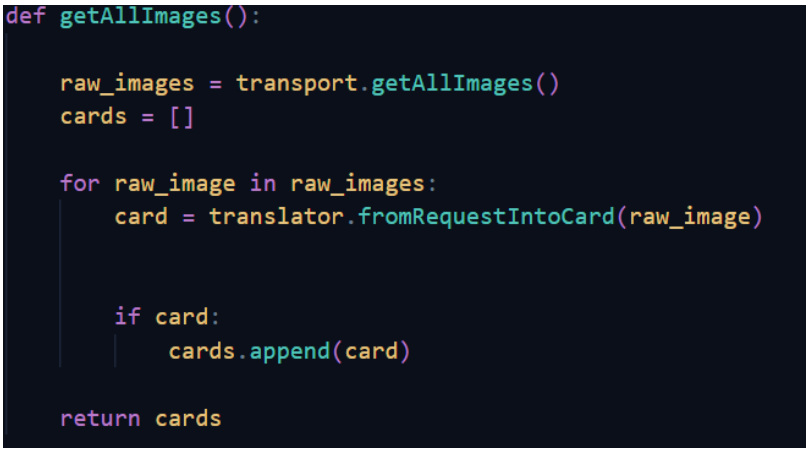
Introducción

En este trabajo desarrollaremos una página web basada en una API de Pokémon, donde lo que debemos hacer es completar código dado por los profesores y a la vez el resultado debe ser unas cards que contengan la imagen, y más datos de los pokemons de la API

Puntos Obligatorios

Dentro de los puntos obligatorios debíamos hacer lo siguiente:

- Actualizar el getAllImages() en el archivo services.py donde se pedia que esta funcion tomase los datos de la API y devolviera las cards, entonces lo que hicimos en esa función fue traer del archivo transport.py el getAllImages() que transformaba los datos de la api en .JSON para luego recorrerla con un for. Luego trajimos la función fromRequestIntoCard() la cual es la función que le daba el estilo a la card. Además hicimos un if donde si card era true, lo appendeaba a la lista cards



```python
def getAllImages():

    raw_images = transport.getAllImages()
    cards = []

    for raw_image in raw_images:
        card = translator.fromRequestIntoCard(raw_image)


        if card:
            cards.append(card)

    return cards
```

- Luego en el archivo views.py se debia modificar a la función home(request) para obtener las imágenes de la API, donde se agrego el objeto images que llama a la función antes nombrada

(getAllImages()) en el módulo services para obtener las imágenes y mostrarlas en la página. Mantuvimos toda la otra parte del código igual.

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

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

- Luego nos encargamos de hacer los bordes de cada card utilizando las etiquetas de plantilla de Django ({% %}) para realizar comprobaciones condicionales. Después de renderizar la plantilla, evalúa estas condiciones y aplica la clase de borde correspondiente. Utilizando también clases que son parte del sistema de colores de Bootstrap, que proporciona estilos predefinidos para los bordes: Si el tipo 'fire' está en img.types, se aplica la clase border-danger (borde rojo), si el tipo 'water' está en img.types, se aplica la clase border-primary (borde azul), si el tipo 'grass' está en img.types, se aplica la clase border-success (borde verde), si no es ninguno de los anteriores, se aplica la clase border-warning (borde amarillo).

  Agregamos además mb-3 la cual es una clase de Bootstrap que significa "margin-bottom: 3" que agrega un margen inferior a la tarjeta y tiene un valor de 3. Esto crea un espacio vertical entre las tarjetas cuando se apilan

  También ms-5 esta otra clase de Bootstrap. En un diseño de izquierda a derecha, esto agrega un margen izquierdo grande. El valor 5 es el máximo en la escala de Bootstrap. Esto empuja la

tarjeta hacia la derecha, creando un espacio en el lado izquierdo. Por último tiene style="max-width: 540px;" este es un estilo establece el ancho máximo de la tarjeta. Limita el ancho de la tarjeta a 540 píxeles. Esto asegura que la tarjeta no se vuelva demasiado ancha en pantallas grandes.

```
{% else %} {% for img in images %}
<div class="col">

    <div class="card
        {% if 'fire' in img.types %}border-danger
        {% elif 'water' in img.types %}border-primary
        {% elif 'grass' in img.types %}border-success
        {% else %}border-warning{% endif %}
        mb-3 ms-5" style="max-width: 540px; ">
```

Puntos Opcionales

Dentro de los puntos opcionales estaban los buscadores. Donde en el primero de los buscadores, el usuario debe poder filtrar por el nombre del pokemon, aca fue donde nos encontramos con las primeras complicaciones, en primera instancia intentamos hacer todo en el views.py pero, nos dimos cuenta que no era de esta manera, entonces entramos al archivo services.py y nos dimos cuenta que la función filterByCharacter() no estaba completa. Dentro de la función habia un

```
def filterByCharacter(name):
    tarjetas_filtradas = []

    for tarjeta in getAllImages():
        # Verifica si el nombre buscado está cont
        nombre de la tarjeta
        if name.lower() in tarjeta.name.lower():
            tarjetas_filtradas.append(tarjeta)

    return tarjetas_filtradas
```

comentario donde se nos pedía que antes de hacer el append, la función debía ver si el nombre estaba en la card o tarjeta como lo nombramos nosotros. Para luego agregar esa tarjeta a la lista vacía que en este caso sería tarjetas_filtradas. Luego lo que hicimos fue llamar la función desde el views.py donde dentro del If igualamos images a filterByCharacter() pasandole en “name” el cual es el que busca el usuario, le pusimos un .lower() para que a pesar de cómo lo escriba el usuario siempre se muestre en el programa de la misma manera y eso no genere errores.

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

    # si el usuario ingresó algo en el buscador, se deben
    filtrar las imágenes por dicho ingreso.
    if (name != ''):
        images = services.filterByCharacter(name)
        favourite_list = []

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

En el segundo buscador se desea filtrar los tipos de pokemons agua fuego y plantas. Implementando lo anterior nuestro programa hace inicio en una lista vacía la cual se crea una condición para verificar si el tipo del pokémon coincide con el tipo recibido por parámetro siguiendo con el programa si coincide lo puede añadir a la lista de filtered_cards.

```
def filterByType(type_filter):
    filtered_cards = []

    for card in getAllImages():
        # Verificar si el tipo del Pokémon coincide con el
        tipo recibido por parámetro
        if hasattr(card, 'types') and type_filter in card.
        types:
            # Si coincide, añadirlo al listado de
            filtered_cards
            filtered_cards.append(card)

    return filtered_cards
```

La función recorre todas las cartas obtenidas de getAllImages() Luego agregamos un “if” hasattr(card, 'types') que se encargará de verificar si el objeto card tiene un atributo llamado 'types' esto para evitar errores si alguna carta no tiene el atributo 'types', en caso de tener el atributo 'types', entonces se procede a verificar si type_filter está en card.types (hasattr() es una función incorporada de Python que se usa para determinar si un objeto tiene un atributo específico. Devuelve True si tiene el atributo, y False en caso de que no)

Este “if” se encargará de que solo las cartas que pasen esta verificación serán incluidas en filtered_cards. Cuando esta función retorna filtered_cards, sólo contendrá los Pokémon del tipo seleccionado.