Radic Valentina

47.308.223

Com-4

Primer Semestre 2025

Informe del Trabajo Práctico-Aplicación Django: Galería de Pokémon

El trabajo práctico consistió en desarrollar una aplicación web sencilla utilizando Django, que permita visualizar una galería de Pokémon obtenidos desde una API externa. Además de mostrar los personajes, se agregaron funcionalidades que mejoran la experiencia del usuario, como un buscador por nombre, un filtro por tipo (fuego, agua o planta) y una animación de carga mientras se procesan los datos.

El sitio permite navegar de forma cómoda, identificar rápidamente los tipos de cada Pokémon por colores y realizar búsquedas personalizadas.

Funciones implementadas:

1. **Mostrar la galería de Pokémon**

```python
# función que devuelve un listado de cards. Cada card representa una imagen de la API de Pokemon
def getAllImages():
    raw_images= transport.getAllImages() #obtengo datos desde la api, uso transport para tarer los datos
    cards = [] #creo lista de cards para despues llenarla

    for image in raw_images:
        card = translator.fromRequestIntoCard(image)
        cards.append(card)  #hago cada imagen una card, transaltor para converit cada uno

    return cards #devuelvo lista completa
    pass
```

Esta función obtiene todos los datos crudos desde la API y transforma cada Pokémon en una Card, que contiene su información organizada. Es la base para mostrar la galería.

2. **Vista Principal**

```python
# esta función obtiene 2 listados: uno de las imágenes de la API y otro de favoritos, ambos en formato Car
def home(request):
    images = services.getAllImages() #uso la funcion que ya complete, obtengo los pokemon
    #si mas adelante hago favoritos, dejo la funcion ya hecha
    favourite_list = services.getAllFavourites(request) if request.user.is_authenticated else []

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

Esta vista obtiene las imágenes y (si el usuario está logueado) sus favoritos, en mi caso lo agregue para no dejar la función vacía por más que no implemente favoritos. Usa los datos para mostrar la galería

3. **Card con bordes de colores según tipo:**

```
<!-- armo las condiciones, segun el tipo de pokemon que dice que es -->
<div class="card mb-3 mb-5"
style="border-width: 3px; border-style: solid; border-color: {% if 'grass' in img.types %}green{% elif 'fire' in img.types %}red{% elif 'wat
```

Se usaron condicionales en Django para cambiar el borde de cada card según el tipo principal de pokmon.

4. Buscador I

Archivo: services.py

```python
# función que filtra según el nombre del pokemon.
def filterByCharacter(name):
    filtered_cards = []

    for card in getAllImages():
        # debe verificar si el nombre de la card contiene el nombre recibido por parámetro
        if name.lower() in card.name.lower(): #verifico si el nombre de la card tiene el nombre que recibe
            #en minusculas para que no haya problemas
            filtered_cards.append(card)

    return filtered_cards
```

Archivo: views.py

```python
# función utilizada en el buscador.
def search(request):
    name = request.POST.get('query', '')

    # si el usuario ingresó algo en el buscador, se deben filtrar las imágenes por dicho ingreso.
    if (name != ''):
        images = services.filterByCharacter(name) #llama a la función que cree en services.py
        # y devulve una lista de pokemons filtrada por el texto que se ingresó
        favourite_list = services.getAllFavourites(request) if request.user.is_authenticated else []

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

Si el usuario ingresa texto, se filtran las cards por nombre. Se hace sin importar mayúsculas o minúsculas.

5. Buscador II- Filtra por tipo

Archivo: services.py

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

    for card in getAllImages():
        # debe verificar si la casa de la card coincide con la recibida por parámetro. Si es así, se añade al listado de filtered_cards.
        if type_filter.lower() in [t.lower() for t in card.types]:
            filtered_cards.append(card)

    return filtered_cards
```

Archivo: views.py

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

    if type != '':
        images = services.filterByType(type) #llama a la función que cree en services.py
        #devulve lista de pokemons filtados por el tipo que se ingreso
        #me aseguro que devuelva lista vacia ya que no va a haber ingreso de usuario.
        favourite_list = services.getAllFavourites(request) if request.user.is_authenticated else []

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

Permite al usuario filtrar por los tipos principales de pokémon (fuego, agua o planta) con botones en la página. Se comparan los tipos de cada Pokémon por el valor recibido.

6. Loading Spinner

Archivo: home.html

Visible mientras carga

```html
    <!-- spinner de carga, lo pongo al comienzo antes del buscador del pokemon-->
<div id="loading" class="text-center mt-5">
  <div class="spinner-border text-primary" role="status" style="width: 4rem; height: 4rem;">
    <span class="visually-hidden">Cargando...</span>
  </div>
  <p class="mt-3">Cargando imágenes, por favor espere...</p>
</div>
```

Sección oculta con las cards

```html
        <div id="content" style="display: none;">
            <div class="row row-cols-1 row-cols-md-3 g-4">
            {% if images|length == 0 %}
            <h2 class="text-center">La búsqueda no arrojó resultados...</h2>
            {% else %}
             {% for img in images %}
```

Script en el final del template:

```html
<!-- script que oculta el spinner, y muetsra la pagina cuando termina de cargar -->
<script>
  window.addEventListener("load", function () {
    document.getElementById("loading").style.display = "none";
    document.getElementById("content").style.display = "block";
  });
```

Se oculta la galería y se muestra un spinner mientras la página carga. Cuando termina de cargar, se oculta el spinner y aparece el contenido.

Dificultades y decisiones

- Se decidió no implementar favoritos porque dependía de tener implementado el sistema de usuarios con inicio de sesión. Como este no era obligatorio y demandaba más tiempo, se optó por enfocarse en las funcionalidades pedidas.
- El filtrado por tipo y por nombre se implementó utilizando los contenidos vistos en la materia. Se analizaron los datos que devuelve la API y se respetó la estructura dada. Se reutilizó el código cuando fue necesario, por ejemplo usando getAllImages() en los filtros.
- El spinner de carga fue el componente que más trabajo implicó. Para hacerlo, se investigó cómo estructurar el HTML con una sección visible al inicio que muestre el spinner, y otra oculta con las imágenes. Luego se usó un pequeño script en JavaScript para ocultar el spinner y mostrar las imágenes cuando la página termina de cargar. Esta implementación mejora la experiencia del usuario, sobre todo si la conexión es lenta o hay muchas imágenes.
- Se implementó también el cambio de color en el borde de cada card según el tipo del Pokémon, usando condicionales de Django dentro del archivo home.html. Como nunca se había trabajado directamente con HTML+Jinja (el motor de plantillas de Django), fue necesario aprender cómo aplicar la lógica aprendida en Python en el estilo de cada elemento. Se utilizaron instrucciones if/elif/else para aplicar diferentes colores.