

Licenciatura en Sistemas

Trabajo Práctico POKÉDEX

Introducción a la Programación

1S 2025

Resumen: aplicación web usando Django que permite buscar imágenes de POKÉMON

Integrantes: Gustavo Nieva (gnieva.1410@gmail.com)
Joaquín Arce Guillem (jarceguillem@gmail.com)



Vista TP

1. Introducción

Este trabajo práctico consiste en el desarrollo de una aplicación web tipo Pokédex utilizando el framework Django. El objetivo es consumir la API pública PokeAPI para obtener datos de diferentes Pokémon y mostrarlos en una interfaz atractiva.

El problema por resolver es cómo mostrar de forma clara, dinámica y organizada información proveniente de una API externa, incluyendo imágenes, tipos, altura, peso y nivel base.

2. Desarrollo

2.1 Descripción general:

El sistema se desarrolló utilizando el framework Django, siguiendo una estructura basada en vistas, servicios y plantillas. La información de los Pokémon se obtiene a través de la API pública PokeAPI y se transforma en objetos Card que se renderizan en una galería visual.

El desarrollo incluyó funcionalidades como la obtención de datos desde una API, la conversión de esos datos a objetos internos, la implementación de filtros por nombre y tipo, un sistema de favoritos para usuarios autenticados, y la mejora de la experiencia de usuario mediante un spinner de carga. Además, se trabajó colaborativamente utilizando Git y GitHub, con un control de versiones efectivo y manejo de ramas.

2.2 Funcionalidades principales:


- Función **filterByType(type_filter)**: Esta función permite filtrar los Pokémon por tipo (como "fire", "water", "grass", etc.), permitiendo al usuario visualizar sólo aquellos Pokémon que coincidan con el tipo seleccionado.

Proyecto TP Inicio **Galeria** Favoritos Salir

Buscador de Pokemon

Pikachu, Charizard, Ditto


FUEGO **AGUA** **PLANTA**



charmander #4 🐉

FIRE ✕


Altura: 6
Peso: 85
Nivel de experiencia base: 62



chameleon #5 🐉

FIRE ✕

Altura: 11
Peso: 190
Nivel de experiencia base: 142



charizard #6 🐉

FIRE
FLYING ✕

Altura: 17
Peso: 905
Nivel de experiencia base: 240

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

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

    return filtered_cards
```

- Función **filterByCharacter (name)**: Esta función permite buscar Pokémon por nombre parcial. Por ejemplo, si el usuario ingresa "saur", la función mostrará "bulbasaur", "ivysaur" y "venusaur".

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

    for card in getAllImages():
        # debe verificar si el name está contenido en el nombre de la card, antes de agregarlo al listado de fi
        if card.name and name.lower() in card.name.lower():
            filtered_cards.append(card)

    return filtered_cards
```



- Función **getAllImages()**: Esta función es el punto de entrada para obtener los datos de los Pokémon desde la API (PokeAPI), transformarlos en objetos internos de tipo Card, enriquecerlos con iconos de tipo, y devolverlos listos para renderizar en la vista principal. Esta función es clave en la lógica general del proyecto, ya que es la que conecta la API externa con las cards que se renderizan en pantalla.

```
# función que devuelve un listado de cards. Cada card representa una imagen de la API de Pokemon
def getAllImages():
    # debe ejecutar los siguientes pasos:
    # 1) traer un listado de imágenes crudas desde la API (ver transport.py)
    json_data = transport.getAllImages()
    poke_card_list = []
    # 2) convertir cada img. en una card.
    for data in json_data:
        card = translator.fromRequestIntoCard(data)
        card.url_types = list(map(get_type_icon_url_by_name, card.types))
        #print(card)
    # 3) añadir las a un nuevo listado que, finalmente, se retornará con todas las card encontradas.
    poke_card_list.append(card)

    return poke_card_list
#pass
```

- Se implementó un **spinner de carga** para mejorar la experiencia de usuario al ingresar o recargar la página, mostrando una animación que indica que el contenido está cargando. Este spinner consiste en un div flotante de pantalla completa que se muestra al cargar la página, centrado mediante flexbox, con un fondo semitransparente blanco para cubrir el contenido hasta que termine de renderizar. Dado que el contenido se carga dinámicamente desde una API externa (PokeAPI), se quiso evitar que el usuario vea una página en blanco mientras se procesan los datos. El spinner mejora la percepción de fluidez y comunica que el sistema está trabajando.

Buscador de Pokemon

Pikachu, Charizard, Ditto

Buscar

FUEGO

AGUA

PLANTA



bulbasaur #1

GRASS
POISON

Altura: 7

Peso: 69

Nivel de experiencia base: 64



ivysaur #2

GRASS
POISON

Altura: 10

Peso: 130

Nivel de experiencia base: 142



venusaur #3

GRASS
POISON

Altura: 20

Peso: 1000

Nivel de experiencia base: 236

```
{% extends 'header.html' %} {% block content %}
```

```
<div id="spinner" style="
```

```
    position: fixed;
```

```
    top: 0; left: 0;
```

```
    width: 100vw;
```

```
    height: 100vh;
```

```
    background-color: rgba(255,255,255,0.8);
```

```
    display: flex;
```

```
    justify-content: center;
```

```
    align-items: center;
```

```
    z-index: 9999;
```

```
">
```

```
<div class="loader"></div>
```

```
</div>
```

```
<style>
```

```
.loader {
```

```
    border: 16px solid #f3f3f3;
```

```
    border-top: 16px solid #3498db;
```

```
    border-radius: 50%;
```

```
    width: 120px;
```

```
    height: 120px;
```

```
    animation: spin 1s linear infinite;
```

```
}
```

```
@keyframes spin {
```

```
    0% { transform: rotate(0deg); }
```

```
    100% { transform: rotate(360deg); }
```

```
}
```

```
</style>
```

```
<script>
```

```
    window.addEventListener('load', function () {
```

```
        setTimeout(function () {
```

```
            document.getElementById('spinner').style.display = 'none';
```

```
        }, 1500);
```

```
    });
```

```
</script>
```

```
{% endblock %}
```

2.3 Funcionalidad Favoritos

- En esta sección se definen las funciones que corresponden a un usuario logueado. Hace uso de la capa Service, que conecta con distintos servicios. Las imágenes son de [View.py](#) y [Service.py](#).

Archivo [View.py](#)

- **getAllFavouritesByUser(request):** Obtiene la lista de favoritos y la envía para desplegarla en el formulario de la página favourites.html.
- **saveFavourite(request):** Guarda en el repositorio el objeto seleccionado en el html y hace un redireccionamiento a la ruta 'home'.
- **deleteFavourite(request):** Borra en el repositorio el objeto seleccionado en el html y hace un redireccionamiento a la ruta 'favoritos'.
- **exit(request):** Realiza un logout del usuario y hace un redireccionamiento a la ruta 'index-page'.

```
# Estas funciones se usan cuando el usuario está logueado en la
aplicación.
@login_required
def getAllFavouritesByUser(request):
    #pass
    favourite_list = services.getAllFavourites(request)
    return render(request, 'favourites.html' ,{'favourite_list' :
    favourite_list})

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

@login_required
def deleteFavourite(request):
    #pass
    borrado = services.deleteFavourite(request)
    print("borrado" if borrado else "no borrado")
    return redirect('favoritos')

@login_required
def exit(request):
    logout(request)
    return redirect('index-page')
```

View.py

Archivo [Service.py](#)

- **saveFavourite(request):** usando el objeto translator y su metodo fromTemplateIntoCard(), le pasamos como argumento el request y lo transforma en un objeto de tipo Favourite(Modelado para la DB). Se le asigna un User desde request y luego con el objeto repositories y su metodo save_favourite() le pasamos el objeto a ser guardado en la DB.
- **getAllFavourites(request):** Si el usuario está autenticado, busca los objetos en la DB, los transforma a tipo Card y los agrega a una nueva lista que se retorna.
- **deleteFavourite(request):** Mediante la request, se obtiene el id del objeto a eliminar en la DB. Usa el objeto repositories y su metodo delete_favourites para eliminar.

```
# añadir favoritos (usado desde el template 'home.html')
def saveFavourite(request):
    fav = translator.fromTemplateIntoCard(request) # transformamos
    un request en una Card (ver translator.py)
    fav.user = get_user(request) # le asignamos el usuario
    correspondiente.

    return repositories.save_favourite(fav) # lo guardamos en la BD.

# usados desde el template 'favourites.html'
def getAllFavourites(request):
    if not request.user.is_authenticated:
        #print("no entro fav")
        return []
    else:
        #print("entro fav")
        user = get_user(request)

        favourite_list = repositories.get_all_favourites(user) #
        buscamos desde el repositories.py TODOS Los favoritos del
        usuario (variable 'user').
        mapped_favourites = []

        for favourite in favourite_list:
            card = translator.fromRepositoryIntoCard(favourite) #
            convertimos cada favorito en una Card, y lo almacenamos
            en el listado de mapped_favourites que luego se retorna.
            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
```

Service.py

3. Conclusiones:

Este trabajo no sólo permitió ejercitar el uso del framework Django, sino también comprender la importancia de una buena arquitectura de código, la interacción entre componentes, la reutilización de funciones y la experiencia de usuario en el desarrollo de aplicaciones web modernas.

El uso de Git como sistema de control de versiones y GitHub como plataforma de colaboración remota facilitó el trabajo en equipo, el control de cambios, el seguimiento de versiones y la integración de funcionalidades desarrolladas por distintos integrantes del grupo de manera ordenada y eficiente. Esto también agilizó las reuniones de equipo ya que no era menester reunirnos físicamente al tener estas herramientas a nuestra disposición.

Anexo: El trabajo consiste en implementar una aplicación web usando Django que permita buscar imágenes de POKÉMON. La información será proporcionada mediante una API y luego renderizada por el framework en distintas cards que mostrarán -como mínimo- la imagen del Pokémon, los tipos del mismo, altura, peso y el nivel base en el que estos existen.

Fuentes:

<https://www.shivatutorials.com/2020/09/how-to-implement-site-loader-in-django.html>
<https://youtu.be/3GymExBkKjE>