Pokedex con Django



Alumno:

Máximo Agustín Solagastua

Mail:

Solagastua14@gmail.com

DNI:

47157444

Materia y comisión:

Introducción a la Programación – comisión 09

Docentes:

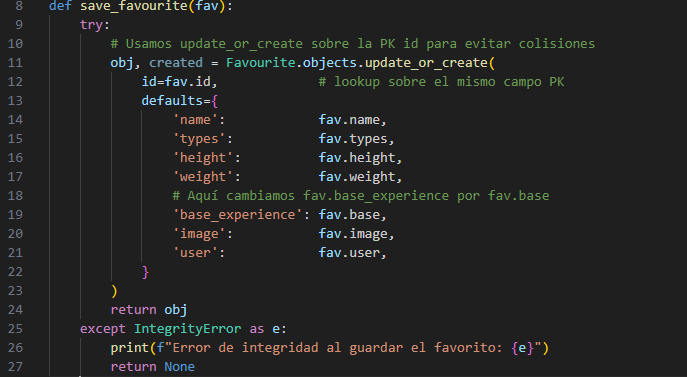
Miguel Rodríguez y Nancy Nores

Introducción:

La Pokedex desarrollada con Django ha evolucionado más allá de una simple galería de imágenes: en este informe describimos cómo se estructuró y perfeccionó la aplicación mediante un enfoque por capas para incorporar tres funcionalidades clave. En primer lugar, implementamos un “spinner” de carga que garantiza una experiencia fluida al indicar al usuario que espere mientras se recuperan datos de la API. A continuación, desarrollamos dos mecanismos de búsqueda —por nombre y por tipo (fuego, agua o planta)— que filtran dinámicamente las tarjetas de cada Pokémon. Finalmente, completamos la gestión de favoritos, permitiendo a usuarios autenticados agregar, listar y eliminar sus Pokémon preferidos sin duplicados, deshabilitando el botón cuando ya han sido marcados. Cada sección de este documento detalla los cambios realizados en la capa de persistencia, la lógica de negocio, los traductores de datos, las vistas y las plantillas, así como las lecciones aprendidas durante el proceso.

1. Repositories.Py.

Funcion modificada: Save\_favourite(fav), antes hacía Favourite.objects.create(...) y volaba si ese ID ya existía.

Actualmente:

Esta versión busca un registro con ese id, si es encontrado, lo actualiza con (defaults).

Sino, crea uno. De esta forma nunca falla por duplicado y siempre tiene datos sincronizados.

Función existente y sin cambios: delete\_favourite(fav\_id)

Simplemente hace Favourite.objects.get(id=fav\_id).delete() y si no existe, captura la excepción y devuelve False.

1. Services.Py.

GetAllImages():

Llama a la API PokeAPI (transport.getAllImages()), recibe una lista de JSON.

Para cada JSON: fromRequestIntoCard(elem) lo convierte en un objeto Card con los campos que usamos en la UI.

Devuelve la lista de Card.

FilterByCharacter(name) y filterByType(type\_filter).

Ambas funciones se encargan del buscador de la Pokedex.

Juntas obtienen todas las cards de GetAllImages, luego filtran en Python: por nombre if name.lower() in card.name.lower(). O por tipo (si type\_filter está dentro de la lista card.types).

SaveFavourite(request):

Recibe el request del form de la plantilla

Hace card=fromTemplateIntoCard(request) para convertir los campos POST (id, name, height, etc.) en una Card.

Le asigna card.user = request.user.

Llama a repo\_save(card) (el repositorio), que usa update\_or\_create.

getAllFavourites(request):

Saca el usuario con get\_user(request).

Pide al repositorio get\_all\_favourites(user), que devuelve una lista de diccionarios.

Convierte cada dict de la BD en una Card con fromRepositoryIntoCard(d)

Devuelve lista de Card.

deleteFavourite(request):

Lee el id del POST: fav\_id=int(request.POST.get('id', 0))

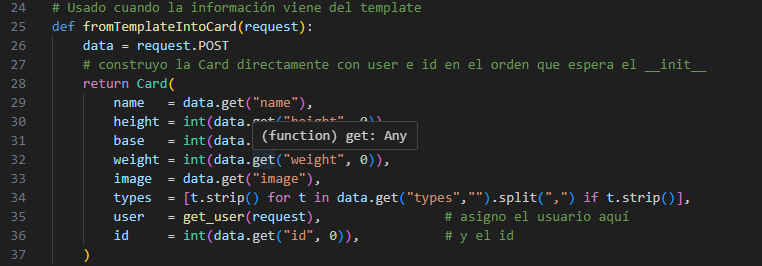
Llama a repo\_delete(fav\_id).

1. Translator.Py

fromTemplateIntoCard(request)

antes usaba directamente los strings del POST.

Ahora:



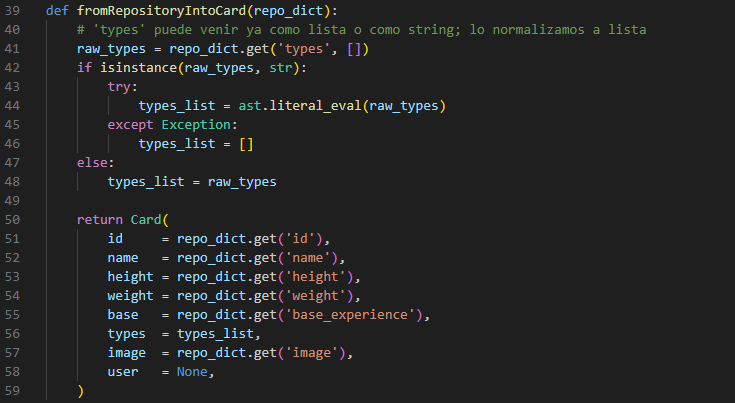
Convierte el ID y base a enteros.

Parsea la cadena “fire,Grass” a lista ["fire","grass"]

Asigna el usuario actual

fromRepositoryIntoCard(repo\_dict)

Antes usaba ast.literal\_eval(repo\_dict['types']) a secas y podía fallar si ya era lista. Por lo tanto realicé modificaciones y quedo de la siguiente forma:



Si Types viene como string JSON, lo evalúa; si ya está lista, lo deja.

1. Views.py

Realicé import y llamadas limpias, antes usaba métodos vacíos o recursivos, ahora:

**home, search, filter\_by\_type**

Obtienen images = services.getAllImages() o filtros.

Si el usuario esta logueado:

****

Agrego en todas las funciones, como aparece en la imagen de la función home, esos renglones que dice “fav\_cards=services.getAllFavourites(request) y favourite\_ids = [c.id for c in fav\_cards]”

Para que envíen solo favourite\_ids al template, para que HTML sepa qué IDs desactivar.

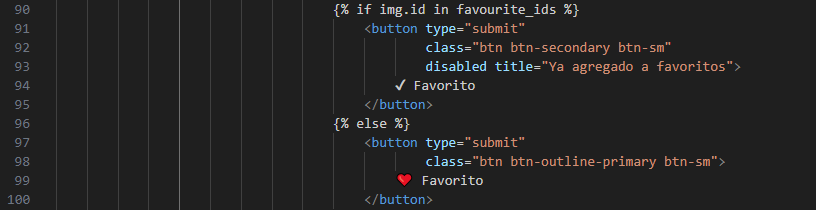
Por último saveFavourites(request) y deleteFavourites(request), llaman directamente a services.saveFavourite(request) o services.deleteFavourites(request). Para posteriormente redirigir a la página correspondiente (next o /favoritos)

1. Plantillas.

Home.html

El form de favoritos incluye ahora todos los campos (id, name, height, weight, base, types, image).

El botón:



De esta forma cumple la consigna: “si ya fue añadido, mostrar botón que impida reañadirlo”.

Favourite.html

Muestra la tabla con XP BASE y TIPOS, y un formulario de borrar que envía solo el ID.

1. El spinner de carga.

Agregué un overlay <div id="loading-spinner"> en el header.

Un pequeño script que muestra el spinner cuando envía un form o clickea un link, y se oculta al terminar de cargar la página.

FLUJO FINAL RESUMIDO

1. Usuario hace click en buscar/filtro/favorito.
2. Vista recibe el request y llama a un servicio.
3. Servicio transforma datos crudos a Card, aplica lógica y toca la BD vía repositorio.
4. Repositorio guarda o borra con update\_or\_create y delete().
5. Traductor asegura que todos los datos estén en el tipo adecuado (enteros,lista, etc.)
6. Plantilla pinta las cards y muestra o deshabilita botones según favourite\_ids.
7. Spinner avisa al usuario mientras todo sucede.

Para realizar este trabajo estuve informándome por varios medios, información dada por profesores (como [pagina1](https://getbootstrap.com/docs/4.0/components/card/), [pagina2](https://medium.com/powered-by-django/if-else-conditions-in-django-templates-5b3658cbf287)), vía Youtube e inclusive chatgpt.

Videos:

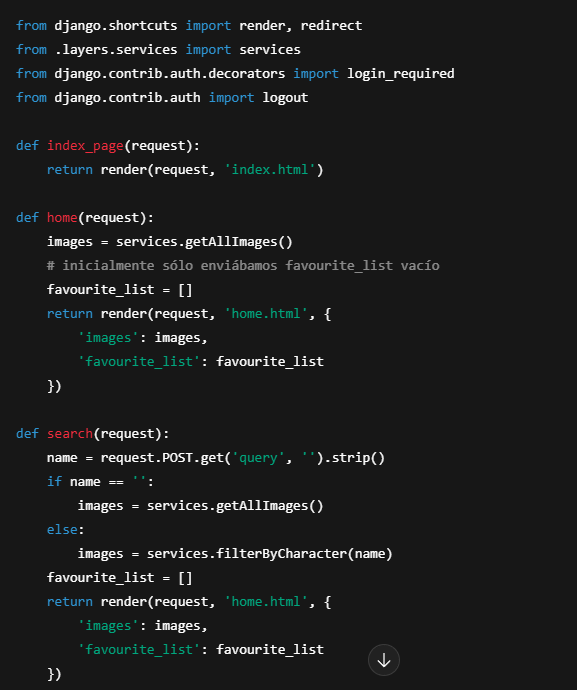
[COMO USAR GIT](https://youtu.be/vlCXdvcgiE0?si=Gn-cyIL5kPWuVF64), este video me sirvió mucho para terminar de comprender GIT.

[COMO USAR GITHUB](https://youtu.be/44ziZ12rJwU?si=PzAMVF61vDp8QF56), este video me sirvió para empezar a familiarizarme un poco con github, uso de forks, etc.

[DEBUG](https://youtu.be/VL5xJmdAHes?si=wbLrQh1DvjVV-6fo), este me enseñó una buena forma de verificar el flujo, dectectar errores, trazabilidad, etc.

[JSON](https://youtu.be/FGG-UTCwlJw?si=9IxeFhRZ2QoowaWm), para introducirme un poco en que es JSON.

Y con chatgpt, necesité ayuda con lo que es recursionError en views.py



Esta era la views.py que chat gpt me recomendó en un momento, la cual evidentemente estaba mal y daba el error mencionado anteriormente.

Y también necesité ayuda con el logueo, chat gpt me dijo que mediante el comando “Python manage.py createsuperuser” ya podía loguearme.

En conclusión, este Trabajo Práctico, pese a los varios dolores de cabeza que me causó, ha resultado una experiencia muy enriquecedora. Más allá de cumplir con la consigna, aprendí a:

1. **Diseñar por capas**: separar el transporte, la lógica de negocio, la persistencia y la presentación me obligó a pensar en responsabilidades claras y a aislar problemas, lo que sin duda hará más sencillo escalar y mantener la aplicación en el futuro.
2. **Manejar errores reales**: enfrentarme a un RecursionError, a fallos por clave primaria duplicada y a conversiones mal hechas, me enseñó la importancia de validar y normalizar datos desde el formulario hasta la base de datos.
3. **Dar feedback al usuario**: implementar el spinner de carga y deshabilitar el botón de favorito cuando ya existía esa entrada me ayudó a comprender cómo mejorar la experiencia de uso y evitar acciones redundantes.
4. **Trabajar con Django y buenas prácticas**: usar update\_or\_create, login\_required, django.contrib.messages y un enfoque de servicio centralizado mejoró mi aprendizaje sobre el ecosistema de Django y la manera recomendada de estructurar un proyecto real.