

 Comision:COM-09

Docentes**:** Miguel Rodríguez y Nancy Nores

Integrantes del Grupo: Valentín Galván

Introducción a la Programación

**INDICE**

**INTRODUCCION- 2**

**DESARROLLO- 2**

**ESTRUCTURA GENERAL- 2**

**FUNCIONALIDADES IMPLEMENTADAS- 2**

**DIFICULTADES ENCONTRADAS- 8**

**CONCLUSION- 9**

**Introducción**

Este trabajo práctico consiste en el desarrollo de una aplicación web con estructura en capas, que actúa como una Pokédex. el objetivo es hacer uso de una API pública llamada POKEAPI para obtener los datos de los pokémon y mostrarlos en una interfaz. Por lo tanto, deberemos lograr que se muestre toda esa información de la API que contiene imágenes, tipos, altura, peso y nivel base del pokémon.

**Tecnologías utilizadas**

**Python 3.10 / Django** – Framework principal del backend

**HTML5 + CSS3+ Bootstrap5** – Estructura y diseño visual responsivo

**PokeAPI** – Fuente externa RESTful de datos sobre Pokémon

**Git** – Control de versiones

**VSCODE** – Editor de código fuente utilizado para programar, gestionar archivos y conectar con GIT

**JAVASCRIPT** – Lenguaje utilizado para animaciones y comportamiento dinámico.

**Desarrollo**

**Estructura general**

La aplicación está dividida en capas:

**Vista** (views.py): gestiona entradas del usuario y renderiza los templates.

**Servicio** (services.py): lógica de negocio y coordinación entre componentes.

**Persistencia** (repositories.py): acceso y manipulación de datos en la base.

**Transporte** (transport.py): conexión con la API externa.

**Utilidades** (translator.py, card.py): transformación de datos y modelo interno.

**Funcionalidades implementadas**

**Visualización general (home, views.py)**

Carga desde la API un conjunto de 29 Pokémon, los transforma en objetos Card y los muestra en cards visuales usando Bootstrap. También indica cuáles ya están guardados como favoritos.



Es el punto de entrada más importante de la app. Cada vez que el usuario vuelve al inicio o recarga la Pokédex, esta función reactiva la carga de tarjetas y asegura que estén estilizadas correctamente.

**Parámetros:**

Request: objeto que representa la solicitud HTTP del usuario.

**Return:**

Renderiza el template home.html con tres variables:

images: lista de objetos Card que representan Pokémon.

favourite\_list: lista de favoritos del usuario autenticado.

favourite\_names: lista de nombres para marcar cuáles están guardados.

**Búsqueda por nombre (search, views.py)**

Filtra los Pokémon por coincidencia parcial de nombre. Se realiza a través de un formulario HTML enviado por POST.



Agrega una capa de interacción personalizada al sistema. Aporta una experiencia de exploración flexible y controlada.

**Parámetros**

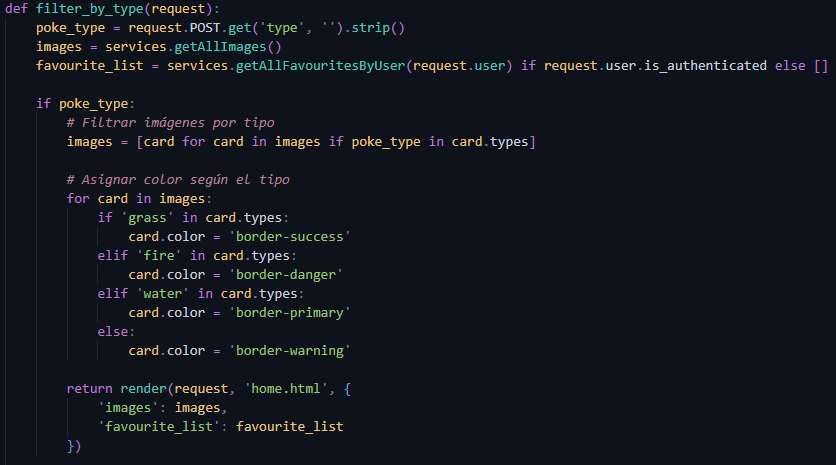
Request: solicitud del usuario con el valor ingresado en el campo de búsqueda (POST['query']).

**Return:**

Muestra los resultados filtrados en home.html junto con los favoritos actuales del usuario.

**Filtro por tipo (filter\_by\_type, views.py)**

Permite al usuario filtrar por tipo elemental (fuego, agua, planta). La lógica valida el tipo, obtiene todas las imágenes, las filtra por los tipos que contiene y asigna una clase de color visual (por ejemplo, border-danger para fuego).

****

Mejora la organización visual y la exploración temática dentro de la Pokédex. Además, permite experimentar con datos dinámicos y filtros multiplataforma.

**Parámetros:**

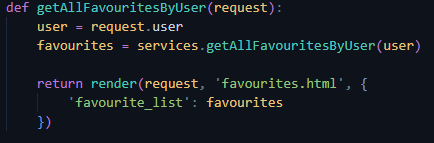
Request: solicitud con tipo enviado desde un formulario (POST['type']).

**Return:**

Si encuentra coincidencias, renderiza home.html con los resultados filtrados; si no, redirige al home.

**Vista de favoritos (getAllFavouritesByUser, views.py)**

Esta función renderiza la plantilla favourites.html, mostrando al usuario todos los Pokémon que guardó como favoritos.



**Parámetros:**

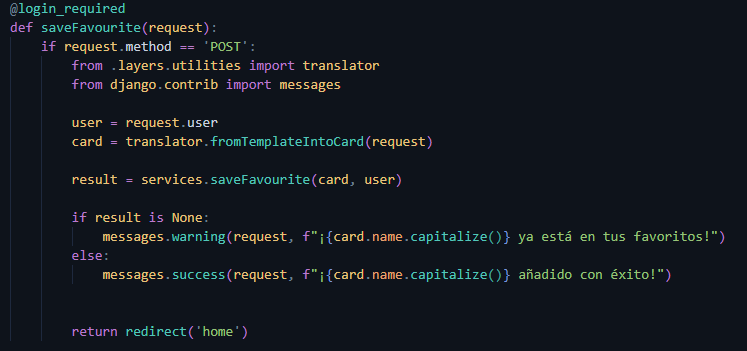
* Request: incluye la sesión del usuario.

**Return:**

* Renderiza el template favourites.html, que recibe como contexto la lista de favoritos del usuario (favourite\_list).

**Guardar favoritos (saveFavourite, views.py)**

Permite almacenar un Pokémon como favorito mediante el formulario integrado en cada tarjeta. El sistema valida que no haya duplicados por usuario antes de guardar.

****

**Parámetros:**

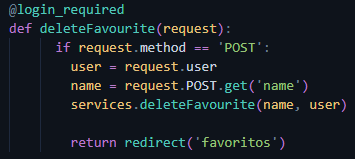
Request: contiene los datos enviados por el formulario (nombre, tipo, imagen, etc.) y el usuario actual.

**Return:**

Redirección a la vista home, con un mensaje (éxito o advertencia) que informa si se agregó correctamente o si ya era favorito.

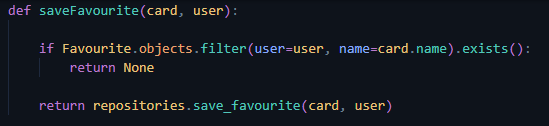
**Borrar favoritos (deleteFavourite,views.py)**

Esta función elimina el favorito seleccionado mediante un botón en el template. Se asegura de que el Pokémon exista y pertenezca al usuario que realiza la operación.

****

**Guardar favoritos (saveFavourite, services.py)**

Guarda un Pokémon como favorito si no fue guardado antes por ese usuario.

****

**Parámetros:**

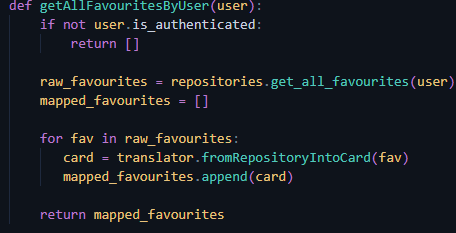
card: objeto con los datos del Pokémon

user: usuario autenticado

**Return:** el objeto guardado o None si ya existía.

**Vista de favoritos (getAllFavouritesByUser, services.py)**

Devuelve los favoritos del usuario autenticado en formato Card.

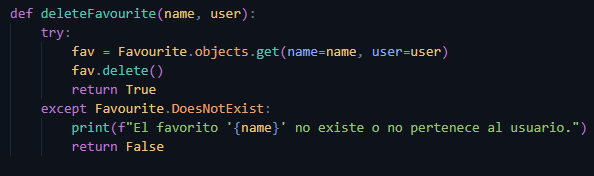
****

**Parámetros:** user.

**Return:** lista de Pokémon favoritos transformados desde la base.

**Borrar favoritos (deleteFavourite, services.py)**

Elimina el favorito si pertenece al usuario.

****

**Parámetros:**

**name:** nombre del Pokémon

**user:** usuario autenticado

**Return:** True si se borró con éxito, False si no existía o no pertenecía al usuario.

**Dificultades encontradas**

**Spinner de carga**: al principio resulto bueno ver que el spinner se mostraba en pantalla, pero ocurrió un problema, el cual era que el spinner se mostraba en una esquina de la página e incluso si cargaba la página, no había modo de que lo pudiera ocultar.

**Aplicar un Fondo a la Página:** en un comienzo intente aplicarle un fondo con un color distinto a la página, había logrado cambiar solo el borde de abajo, pero no sabía cómo cambiar el fondo en general de toda la página por lo tanto busque en internet y pude solucionar ese problema

**Detección y prevención de favoritos duplicados:** Al permitir que el usuario agregue un Pokémon a favoritos, podía ocurrir que se guarde varias veces el mismo Pokémon si no se validaba correctamente. Para solucionarlo implemente una verificación en services.saveFavourite() con una consulta previa:



Esta línea de código me permitió Evalúar si ya existe un registro de favorito con ese nombre de Pokémon asociado al **usuario actual**. Si existe, se evita guardarlo otra vez y retorna None, lo cual permite mostrar un mensaje como “ya está en favoritos”.

Entre las cosas que me hubieran gustado implementar en el trabajo tenemos, un mejor diseño de interfaz con más tipos y pokémon, por lo menos todos los 151 que había en la primera generación. Otra de las funciones que no pude implementar, pero por falta de tiempo fue el alta de nuevos usuarios.

30/6 Volví a revisar el código, y pude incluir fondo nuevo, Card con borde más visible y el spinner lo pude fixear para que funcione correctamente.

**Conclusión**

Este trabajo me permitió aplicar conceptos de arquitectura multicapa, integrar una API externa real, trabajar con modelos personalizados, formularios protegidos y sesiones. Aprendí a organizar una app en capas independientes pero conectadas entre sí y además profundizar sobre las funcionalidades y uso de HTML, JSON, GIT, DJANGO entre otros. Al principio el trabajo se veía complejo a pesar de tener gran variedad de las funcionalidades ya implementadas, porque había que relacionar lo ya implementado con lo que faltaba implementar, pero con la ayuda de internet, poco a poco pude ir progresando.