Proyecto Trabajo Practico Introducción a la Programación

El siguiente proyecto está realizado por Barrionuevo Noelia y Candela Gullo.

Este proyecto consta de la modificación y la perfecta ejecución de una aplicación web, que permite buscar imágenes de Pokémon. La aplicación tiene una gran parte ya desarrollada y lo que se debe incorporar por los miembros del proyecto son algunas funciones en las capas correspondientes.

El desarrollo del trabajo práctico se dará en tres partes. En la primera tendremos los puntos principales e indispensables para ejecutar. En la segunda parte que es opcional, los puntos Estrellas los cuales son tres y por último desarrollar el Alta de nuevo usuarios, loading spinner para la carga de imágenes y la renovación de la interfaz gráfica.

Las principales ejecuciones para el funcionamiento de la página dependen de tres puntos clave.

1. En el módulo views.py hay que obtener dos listados que corresponden a las imágenes de una Application Programming Interface (API) , un interfaz que permite que una aplicación use funciones o datos de otra y una lista de los favoritos del usuario.
2. En services.py hay que conseguir un listado de imágenes convertidas en card desde la API.
3. En el último paso para completar los tres primeros requerimientos para alcanzar el objetivo principal del proyecto hay que cambiar el borde de la card según los tipos más representativos fuego , agua , planta.

En el punto(1), se requiere que completamos las listas vacías que se encontraban en el apartado “services.py”, estas listas tenían que devolver las imágenes que corresponden a cada Pokémon y asimismo tenía que convertirlas en “cards”, para esto se utilizó lo visto en clase y se llamó a las funciones definidas en otros apartados.

* La primera lista es la que se encarga de importar las imágenes desde la API, esto está definido en el apartado de “transport.py”, se llamó a la función “getAllImages” ya definida en este apartado se utilizó el nombre del mismo apartado para que no diera problemas en su implementación.
* raw\_images = transport.getAllImages()
* La segunda lista es la que va a modificar todas las imágenes y los datos que se deben implementar a cada Pokémon, se va a utilizar la función “fromRequestIntoCard” definida en el apartado de “[translator.py](http://translator.py)”, se usa la variable “img” para poder recorrer toda la lista definida en el punto anterior (raw\_images) para poder aplicarle la función y que está transforme los datos crudos sacados de la API en el formato “cards”.
* cards=[translator.fromRequestIntoCard(img) for img in raw\_images]
* Una vez definidas las listas la función va a retornar la última que ya trae los datos modificados.
* return cards

Punto(2), se deben mostrar las imágenes importadas ya modificadas anteriormente en la página correspondiente.

* Vamos a utilizar la función definida y modificada en el punto(1), que se encuentra en el apartado “[service.py](http://service.py)” esta función nos trae desde la API imágenes crudas y en la misma función se modifican para que el usuario pueda entender mejor los datos.
* images = services.getAllImages()
* Luego vamos a retornar los datos ya definidos usando funciones de DJANGO, se utilizó la función “render” que genera una página HTML a partir de un “objetivo” que sería “request” representa lo que está pidiendo el usuario para que DJANGO sepa cómo responder, una plantilla que en este caso como ya esta creada usamos la misma que seria la pagina “home.html” y los datos que debe mostrar en la misma que sería la variable “images” que contiene los datos que desea ver el usuario.
* return render(request, 'home.html', { 'images': images })

Punto(3): La información se tuvo que estudiar y obtener de la página Bootstrap con un apartado llamado Fronteras, donde había información más específica de lo que se requería para llevar adelante el trabajo.

* En este punto del proyecto, debemos implementar condicionales en *home.htm*l. Para trabajar con la lógica de HTML debemos usar una herramienta de Python llamada Django. Las condicionales en Django no se escriben de la misma manera que en PyScripter y las detallaremos a continuación.
* Cuando se utilice el condicional if, se debe escribir de la siguiente manera {% if condición %}
* Si la condición del if resulta falsa, se prosigue con else. En Django se desarrolla de la siguiente forma {% else %}
* Para finalizar se debe colocar lo siguiente: {% endif %}, cuya etiqueta marca el final del bloque if/else, sin esto el bloque no se ejecuta.
* Una parte de la estructura del código en *home.html* ya está programada, con una sentencia de ejecución “for” donde analiza cada imagen dentro de una lista llamada Images.
* {% for img in images %}
* Esta línea lo que va a permitir es que al agregar el condicional se analiza cada img (imagen=card) y me clasifique según lo que le pida. Dentro del condicional pediremos que si la img es de tipo “fire” agregue al card (tarjeta) un borde rojo, si es de tipo “water” borde azul , y si es de tipo “grass” será verde, además si la img no entra en ninguno de las anteriores clasificaciones deberá tener un borde naranja. Para ejecutar dichas clasificación lo que hay que hacer es acceder a la propiedad type del objeto img que representa el tipo de Pokemón, y se escribirá de la siguiente forma img.type. El código ejecutado es el siguiente:
* {%if "fire" in img.types%}
* <div *class*="card border-danger mb-3 " *style*="max-width: 540px; " {%elif "water" in img.types%}
* <div *class*="card border-primary mb-3" *style*="max-width: 540px; " >
* {%elif "grass" in img.types %}
* <div *class*="card border-success mb-3 "*style*="max-width: 540px; " >
* {%else%}
* <div *class*="card border-warning mb-3" *style*="max-width: 540px; " >
* {%endif%}
* Para enmarcar los tipos de tarjetas por color utilizamos parte del código base que tenían las tarjetas antes de ser clasificadas. Estas cars inicialmente tenian un borde de color verde, entonces lo que se hizo es reutizicar ese codigo y solo cambiar una parte de *class* por los colores que requeria y ademas se elimino la class de CSS de Bootstrap ms-5, que le daba el margen del lado izquierdo a la tarjeta, esta eliminación de configuración tuvo un proposito por que cuando se realizo la modificacion del interfaz de las tarjetas provoca que las card queden unas y otras de distinto tamaño (se explicara mas adelante las configuraciones del interfaz). Los código de color fueron buscados en Bootstrap. Notar detalle importante: las tarjetas que tienen tipos variados distintos a “fire”,”water” y “grass”, no son de color naranja como se solicita en el proyecto por que dicho color no esta determinado en la paguina de codigos con la que trabajasmos, por lo tanto lo que se hizo fue enmarcar las tarjetaas de color amarillo.
* Por último explicamos porqué se decidió anular la siguiente línea:
* *<div class="card border-success mb-3 ms-5" style="max-width: 540px; "*
* Ya que se configuró que cada tarjeta tenga un color determinado según su clasificación , el margen verde era innecesario por eso se comentó el código.
* Segundo apartado con ejercicios estrellas y opcionales del trabajo. En esta parte se ejecutan dos buscadores y el desarrollo del almacenamiento de imágenes de la galería como favoritos del usuario . En la primera instancia, Estrella 1°, se debe completar la funcionalidad del buscador para filtrar según el nombre del Pokemon. En la segunda estrella , segundo buscador , se debe completar la funcionalidad para que los tres botones de filtrado completen la operación de clasificar los card por tipo , fuego, planta o agua y por último se debe completar la lógica presente para que un usuario logueado pueda almacenar imágenes en un apartado de favoritos, mediante el clic de un botón.
* Buscador I . Primer punto Estrella:
* Se implementó un Buscador (según el nombre del Pokemon), primero se va a verificar si el usuario escribió algo, si esto es verdadero va a seguir con el resto del condicional.
* if (name != ''):
* Si por el contrario es falso se va a redirigir a la pagina donde estan todas la imagenes “home.html”.
* else:
* return redirect('home')
* Dentro del condicional importamos las imágenes de la función “getAllImages()”.
* images = services.getAllImages()
* Luego le decimos mediante un bucle que recorra cada imagen “img” en la lista “images”, hacé algo con ella (en este caso, solo la agregás si cumple la condición del if) y solo muestre las imagenes si son iguales al nombre ingresado sin diferenciar mayúsculas de minúsculas con ayuda de “lower()”.
* images = [img for img in images if name in img.name.lower()]
* Por último muestra el resultado final en la página “home.html”
* return render(request, 'home.html', { 'images': images})
* Buscador II:
* Para desarrollar este buscador tenemos que saber de donde proviene la información necesaria para poder ejecutar los botones de FUEGO , AGUA y PLANTA los cuales se ubican en *home.html* y visualmente en la página web, botón Galería. Para saber si una tarjeta pertenece a alguno de los botones de clasificación se tuvo primero clasificar las card y esto se logra con la función llamada filterByType ubicada en services.py
* def filterByType(type\_filter):
* filtered\_cards = []
* for card in getAllImages():
* if type\_filter in card.types:
* filtered\_cards.append(card)
* return filtered\_cards
* La estructura de esta función ya estaba determinada , lo que se incorporó fue el condicional
* if type\_filter in card.types: que analiza el parámetro ya determinado. Sí, cuyo parámetro se encuentra en la lista de card.types= [“fire”, “water”,”grass”...] entonces me agrega la tarjeta/card a una nueva lista llamada filtered\_cards. Por lo tanto la función filterByType retornara las tarjetas clasificadas. Con el código anterior ya resuelto pasamos a la capa *views.py* donde utilizaremos la función filter\_by\_type la cuál terminará de clasificar los Pokemon por tipo. Todo esto será posible si importo desde services.py la función filterByType a views.py, que traerá un listado filtrado de imágenes.
* from app.layers.services.services import filterByType
* Por último la función filter\_by\_type se usará en *home.html* la estructura de la página, que va a permitir la correcta ejecución de los botones AGUA, PLANTA y FUEGO.
* Favoritos, Estrella II:
* Se implementó la página de Favoritos según lo que el usuario haya elegido, en esta página se veran los pokemones elegidos por el usuario, se podrán agregar desde la galería y así mismo se podrán eliminar si así lo desea el usuario desde la misma página de favoritos. Vamos a crear una cadena en la función home(request)donde llamamos a otra función que verifica los datos del usuario:
* favourites = getAllFavouritesByUser()
* Retorna los datos de todos los pokemones favoritos seleccionados por el usuario y guardados en la base de datos, para después mostrarlos en la página de home.html:
* return render(request, 'home.html', {'images': images,'favourite\_ids': favourite\_ids })
* En la función saveFavourite(request)en esta función vamos a guardar lo favoritos seleccionados por el usuario, para eso vamos a utilizar el método POST para pedir los datos:
* if request.method == 'POST':
* Extraemos los datos del formulario HTML:
* poke\_id = request.POST.get('id')
* name = request.POST.get('name')
* height = request.POST.get('height')
* weight = request.POST.get('weight')
* base\_experience = request.POST.get('base\_experience')
* types = request.POST.getlist('types')
* image = request.POST.get('image')
* Verificamos si el pokemon ya fue seleccionado por el usuario:
* exists = Favourite.objects.filter(user=request.user, name=name).exists()
* Si este no fue seleccionado se crea uno nuevo con los datos del pokemon y el usuario para mostrarse después:
* if not exists:
* Favourite.objects.create(
* id=poke\_id,
* name=name,
* height=height,
* weight=weight,
* base\_experience=base\_experience,
* types=types,
* image=image,
* user=request.user )
* Si ya existe le muestra al usuario un mensaje y luego lo redirige a la galería:
* else:
* messages.info(request, 'Ese Pokémon ya está en tus favoritos.')
* return redirect('home')
* En la función getAllFavouritesByUser(request)se van a obtener todos los favoritos que selecciono el usuario filtrandolos con filter para que no se obtengan datos de otros usuarios y se van a retornar en la página favourites.html :
* favoritos = Favourite.objects.filter(user=request.user)
* return render(request, 'favourites.html', {'favourite\_list': favoritos})
* La función deleteFavourite(request)se va a usar para cuando el usuario quiere eliminar algún pokemon de la lista de favoritos, para esto definimos nuevamente el método POST, para obtener el id del dato que se quiere eliminar:
* if request.method == 'POST':
* fav\_id = request.POST.get('id')
* Primero buscamos la id del favorito que el usuario quiere eliminar:
* fav = Favourite.objects.filter(id=fav\_id, user=request.user).first()
* Si se encontró el pokemon con el id que se desea eliminar se eliminará y aparte le mostrará al usuario un mensaje de verificación:
* if fav:
* fav.delete()
* *#Candela Gullo: Mensaje de éxito si se elimina correctamente*
* messages.success(request, 'El favorito fue eliminado.')
* Si por el contrario no se encontró dicha id se mostrará un mensaje de error para que el usuario verifique si tuvo algún error y luego redirige al usuario a la página de favoritos:
* else:
* *# Candela Gullo: Mensaje de error si no se encuentra el favorito*
* messages.error(request, 'No se encontró el favorito o no te pertenece.')
* return redirect('favoritos')

Como último apartado del proyecto , desarrollaremos:

* Segundo, propone implementar en pantalla una pantalla de carga con el objetivo de informar al usuario que debe esperar mientras se completa la tareas de carga de la galería de imágenes. Esta funcionalidad resulta especialmente útil en situaciones donde la aplicación puede presentar demoras al procesar o traer una gran cantidad de datos.
* Alta de Usuario:
* Se implementó el registro/alta de nuevos usuarios para esto se tuvo que agregar un botón en la página Login.html:
* <a *href*="{% url 'register' %}">Registrarse</a>
* Este botón nos llevará a un nuevo formulario implementado en la carpeta “templates-registration”, este formulario es similar al del login, con algunos cambios que nos permiten saber más cosas sobre el usuario para poder registrarlo.
* En la funcion register(request)definida en la pagina Views.py vamos a poder realizar todo el registro del usuario, empezamos definiendo el método POST y extrayendo los datos del formulario que llenó el usuario:
* if request.method == 'POST':
* first\_name = request.POST['first\_name']
* last\_name = request.POST['last\_name']
* username = request.POST['username']
* password = request.POST['password']
* email = request.POST['email']
* Luego vamos a verificar si el nombre del usuario está en uso, ya que este mismo no se puede repetir:
* if User.objects.filter(username=username).exists():
* Si el nombre del usuario está en uso sale un mensaje de error y lo redirige al formulario de registro:
* messages.error(request, 'El nombre de usuario ya existe. Elegí otro.')
* return redirect('register')
* En cambio si el usuario no existe se crea uno nuevo con los parámetros que se ingresaron en el formulario:
* user = User.objects.create\_user(
* username=username,
* password=password,
* email=email,
* first\_name=first\_name,
* last\_name=last\_name
* )
* Una vez creado el usuario se le enviará un correo electrónico de bienvenida para el usuario, con el siguiente contenido:
* mensaje = f"Bienvenido {first\_name}!\n\nTus credenciales son:\nUsuario: {username}\nContraseña: {password}"
* send\_mail('Registro exitoso en la aplicación',*#Asunto del correo.*
* mensaje, *#Cuerpo del correo.*
* settings.DEFAULT\_FROM\_EMAIL,#*Dirección del remitente. (definida en settings.py).*
* [email], *#Lista con el destinatario.*
* fail\_silently=False)#*Si ocurre un error, lo muestra.*
* messages.success(request, 'Registro exitoso. Revisa tu correo electrónico.')
* Este mismo se tuvo que definir o configurar en el apartado settings.py:
* *# Esta línea define el backend de envío de correo.*
* *# Para permitir enviar correos reales a través de un servidor (Gmail).*
* EMAIL\_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
* *# Especifica el servidor Gmail que se va a usar para enviar los correos.*
* EMAIL\_HOST = 'smtp.gmail.com'
* *# Puerto que usa Gmail para la conexión segura TLS*
* EMAIL\_PORT = 587
* *# Indica que se debe usar el protocolo TLS para enviar el correo de forma segura (cifrada).*
* EMAIL\_USE\_TLS = True
* *# Correo desde el cual se enviarán los mensajes (puse el mio como ejemplo).*
* EMAIL\_HOST\_USER = 'candegullo22@gmail.com'
* *# Esta contraseña permite que Django se conecte a Gmail para enviar correos.*
* EMAIL\_HOST\_PASSWORD = 'jtpf oqdn waui puor'
* *# Dirección de correo que se mostrará como remitente por defecto al enviar emails.*
* DEFAULT\_FROM\_EMAIL = EMAIL\_HOST\_USER
* Interfaz gráfica:
* Primer apartado explicativo de cómo se renovó el interfaz de la pantalla principal del Proyecto, cuyos datos están ubicados en el archivo de página header.html. Para éstas modificaciones se utilizaron códigos ya determinados por la pág. Bootstrap.
* Utilizando un código extraído de Bootstrap, apartado Flex, se modifica la posición de los botones de enlace, Inicio, Galería e Iniciar Sesión. El código implementado es el siguiente
* <div *class*="d-flex justify-content-center w-100"
* Quiero además que el botón de cada enlace tenga un recuadro y esté pintado. El código base es el siguiente:
* <a *class*="nav-link active" *aria-current*="page" *href*="{% url 'index-page' %}"><strong>Inicio</strong></a>
* Simplemente modificamos la class=”nav-link active” con los colores que voy a elegir para asemejar mi página visualmente a un Pokédex.
* *class*="btn btn-warning" <!-- Para color amarillo>
* Esta modificación se aplicó para cada uno de los botones que aparecen en el <body> del archivo header.html. Se modificará visualmente los siguientes enlaces:
* Inicio
* Galería
* Inicio sesión
* Favoritos
* Salir
* También se agregó un recuadro a la palabra Proyecto TP para que vaya acorde a la gama de colores que queremos que tenga la página. Se agregó a *class* el color. El fragmento de código quedó de la siguiente forma:
* <a *class*="navbar-brand btn btn-danger" *href*="#">Proyecto TP</a>
* *class* previamente tenía solo la indicación *class=”navbar-brand”* y dentro de las comillas se agregó la característica del color que se quiere visualizar. Se borro la etiqueta <strong> de la palabra Galería , para que no resalte del resto y se vean todas iguales .
* En el archivo index.html modificamos el saludo principal y agregamos “Bienvenidos a Pokédex!”. Utilizamos el código que ya está implementado y solo agregamos a *class* una etiqueta que hace que el saludo se vea con un recuadro de un extremo de la página a la otra, denominado por Bootstrap como Alerta. Además se agregó un resaltador a las letras, ésto fue copiado de la línea del button Galería, del archivo de header.html. <Strong> es la etiqueta de marcado que usamos para hacer el resaltador.
* <h2 *class*="text-center alert alert-success" *role*="alert"> <strong>Bienvenido a Pokedex!</strong></h2>
* alert alert-success role=”alert” hace que el texto se vea recuadrado por una franja de color. Esta misma *class* y *role* se ejecutaron en home.html con el texto Buscador de Pokemon para tener una continuidad del estilo del proyecto.
* Siguiendo con la idea de que los colores de la pantalla principal se asemeja al juego, cambie el color del link que te dirige a mirar las imágenes y también le agregue dos emojis para hacerlo más “divertido” visualmente. El código que sume para el color del link fue el siguiente:
* *class*="link-danger link-offset-2 link-underline-opacity-25 link-underline-opacity-100-hover"
* Este código lo extraje del apartado Enlace de la página. Bootstrap.
* En archivo de la página de loguin.html se modifica el interfaz de Iniciar sesión. Se cambia la estructura del grupo de entrada, Usuario y Contraseña con una “etiqueta flotante” esto hará que al escribir o al hacer click, la etiqueta del formulario se “eleve” hacia arriba. El código siguiente hace que la interfaz se vea más elegante y moderna. Este código permite que se “eleve” los recuadros:
* <div *class*="form-floating">
* Se le agregó un icono al comienzo del recuadro del formulario, tanto en la parte donde se pide el dato del usuario como donde se solicita la contraseña.
* <span *class*="input-group-text">@</span>.
* <label *for*="floatingInputGroup1">Nombre del usuario </label>
* Con ésta línea estamos mostrando que usamos un flotante, lo que hace que cuando marquemos con el cursor para poder completar el formulario el recuadro se iluminará, dando de ese modo un mejor efecto hacia la vista del usuario y efecto a la página. para el recuadro de contraseña se realizó lo mismo y el código completo es el siguiente:
* <div *class*="form-group" *style*="margin-bottom: 5%;" >
* <div *class*="input-group ">
* <span *class*="input-group-text">🗝️</span>
* <div *class*="form-floating">
* <input *type*="password" *name*="password" *id*="password" *class*="form-control" *id*="floatingInputGroup1" *placeholder*="contraseña" *required*="required">
* <label *for*="passwordInput" > Contraseña </label>
* Se incorporó una modificación de cambio de lugar a toda la estructura de Inicio de sesión junto con los botones de usuario y contraseña para que todo el conjunto se centre en medio de la pantalla para ello se agregó a *class*  el código siguiente:
* position-absolute top-50 start-50 translate-middle
* Además de los cambios anteriores, se le cambió el color al botón de “Ingresar” pasó del celeste estándar con el que ya estaba programado a verde para estar acorde a lo que deseamos realizar. Y como último se agregó un emoji sacado de internet ,al link registrarse.
* Se agregó en el archivo de la página home.html, en cada tarjeta, donde salen los datos del Pokémon como altura, peso y nivel, un recuadro estilo lista. Además se agregó un código etiqueta <strong>, a cada dato para que la letra salga en negrita. El código de lista fue el siguiente y se extrajo de Bootstrap. El código fue el siguiente:
* *class*="card-text list-group-item
* A la leyenda del Buscador de pokemon se le agregó una “alerta” para que pueda verse resaltado
* <h1 *class*="text-center alert alert-success " *role*="alert">Buscador de Pokemon</h1>
* Los emojis agregados en cada botón fueron extraídos de internet .

* En el archivo de la página de login.html se modifica el interfaz de Iniciar sesión. Se cambia la estructura del grupo de entrada, Usuario y Contraseña con una “etiqueta flotante” esto hará que al escribir o al hacer click, la etiqueta del formulario se “eleve” hacia arriba. Esto hará que la interfaz se vea más elegante y moderna.
* <span *class*="input-group-text">@</span>
* Se le agregó un icono al comienzo del recuadro del formulario, tanto en la parte donde se pide el dato del usuario como donde se solicita la contraseña.
* <label *for*="floatingInputGroup1">Nombre del usuario </label>
* Con ésta línea estamos mostrando que usamos un flotante, lo que hace que cuando marquemos con el cursor para poder completar el formulario el recuadro se iluminará dando de ese modo un mejor efecto hacia la vista del usuario y efecto a la página.
* Además de los cambios anteriores, se le cambió el color al botón de “Ingresar” paso del celeste estándar con el que ya estaba programado a verde para estar acorde a lo que deseamos realizar. Se agregó al link de registrarse un emoji para hacerlo acorde al estilo, tal emoji fue extraído de internet.