Introducción a la programación COM-02

Trabajo practico grupal

Nombre de los profesores: Bottino Flavia y Bidart Gauna Lucas.

Integrantes: Joaquín Benítez, Patricio Stivic, Roberto Duarte, Valentino Forastello.

Introducción

Este trabajo tiene como objetivo desarrollar una aplicación web que como principal característica tiene la de poder importar archivos desde una API para reutilizar datos y complementarlos en nuestro programa.

Funciones y códigos usados:

Archivo **views.py**

Este archivo contiene las vistas que es lo que maneja la interfaz para mostrarle al usuario la página.

```
def index_page(request):
    return render(request, 'index.html')
```

Función que se encarga de renderizar la página de inicio.

```
def register(request):
    if request.method == 'POST':
        # Obtener datos del formulario
        first_name = request.POST.get('first_name')
        last_name = request.POST.get('last_name')
        username = request.POST.get('username')
        email = request.POST.get('email')
        password = request.POST.get('password')
        confirm_password = request.POST.get('confirm_password')

        # Validaciones
        errors = []

        # Verificar que todos los campos estén completos
        if not all([first_name, last_name, username, email, password, confirm_password]):
            errors.append("Todos los campos son obligatorios.")

        # Verificar que las contraseñas coincidan
        if password != confirm_password:
            errors.append("Las contraseñas no coinciden.")

        # Verificar que la contraseña tenga al menos 8 caracteres
        if len(password) < 8:
            errors.append("La contraseña debe tener al menos 8 caracteres.")

        # Verificar que el email sea válido (validación básica)
        if '@' not in email or '.' not in email:
            errors.append("El email no es válido.")

        # Verificar que el nombre de usuario no exista
        if User.objects.filter(username=username).exists():
            errors.append("El nombre de usuario ya existe. Por favor, elige otro.")
```

```
        # Verificar que el email no esté en uso
        if User.objects.filter(email=email).exists():
            errors.append("El email ya está registrado. Por favor, usa otro email.")

        if errors:
            # Si hay errores, mostrar mensajes y volver al formulario
            for error in errors:
                messages.error(request, error)
            return render(request, 'registration/register.html')

        try:
            # Crear el usuario
            user = User.objects.create_user(
                username=username,
                email=email,
                password=password,
                first_name=first_name,
                last_name=last_name
            )

            # Autenticar y loguear al usuario
            user = authenticate(username=username, password=password)
            if user is not None:
                login(request, user)
                messages.success(request, f"¡Bienvenido {user.first_name}! Tu cuenta ha sido creada
exitosamente.")
                return redirect('home')
            else:
                messages.error(request, "Error al autenticar el usuario.")
                return render(request, 'registration/register.html')

        except Exception as e:
            messages.error(request, f"Error al crear la cuenta: {str(e)}")
            return render(request, 'registration/register.html')

    return render(request, 'registration/register.html')
```

Función que se encarga del renderizado del registro para nuevos usuarios.

```
def home(request):
    images = services.getAllImages(request)
    favourite_list = services.getAllFavourites(request)

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

Función que se encarga de renderizar la vista principal de la aplicación. Esta función obtiene dos listas, una de las imágenes de la api y otro de favoritos.

```
def search(request):
    name = request.POST.get('query', '')

    if name:
        images = services.filterByCharacter(name)
        favourite_list = services.getAllFavourites(request)
        return render(request, 'home.html', { 'images': images, 'favourite_list': favourite_list })

    return redirect('home')
```

Esta función sirve para mostrar la vista del buscador.

```python
def filter_by_type(request):
    type_filter = request.POST.get('type', '')

    if type_filter:
        images = services.filterByType(type_filter)
        favourite_list = services.getAllFavourites(request)
        return render(request, 'home.html', { 'images': images, 'favourite_list': favourite_list })

    return redirect('home')
```

Esta función sirve para filtrar la búsqueda por el tipo del podemos.

```python
@login_required
def getAllFavouritesByUser(request):
    favourite_list = services.getAllFavourites(request)
    return render(request, 'favourites.html', { 'favourite_list': favourite_list })
```

Una vista que muestra los favoritos del usuario registrado.

```python
@login_required
def saveFavourite(request):
    if request.method == "POST":
        services.saveFavourite(request)
    return redirect('home')
```

Una función para guardar un Pokemon como favorito.

```python
@login_required
def deleteFavourite(request):
    if request.method == "POST":
        services.deleteFavourite(request)
    return redirect('home')
```

Una función para eliminar un Pokemon de favorito.

```python
@login_required
def exit(request):
    logout(request)
    return redirect('home')
```

Vista para cerrar la sesión del usuario registrado.

Archivo **services.py**

Este archivo contiene toda la lógica interna de la aplicación a su vez como intermediario entre el archivo transport.py y views.py

```python
def getAllImages(request=None):
    raw_images = transport.getAllImages()
    favourite_ids = []

    if request and request.user.is_authenticated:
        favourites = getAllFavourites(request)
        favourite_ids = [str(card.id) for card in favourites]

    card_list = []
```

```
    for raw in raw_images:
        card = translator.fromRequestIntoCard(raw)
        card.is_favourite = str(card.id) in favourite_ids
        card_list.append(card)

    return card_list
```

Esta función obtiene un listado de las cards de los Pokemon desde la API. Cada cards representa la imagen de un Pokemon.

En caso de que el usuario este registrado agrega la opción de agregar a favoritos.

```
def filterByCharacter(name):
    all_cards = getAllImages()
    filtered_cards = []

    for card in all_cards:
        if name.lower() in card.name.lower():
            filtered_cards.append(card)

    return filtered_cards
```

Función que filtra por nombre o por caracteres que coincidan.

```
def filterByType(type_filter):
    all_cards = getAllImages()
    filtered_cards = []

    for card in all_cards:
        if type_filter.lower() in [type.lower() for type in card.types]:
            filtered_cards.append(card)

    return filtered_cards
```

Función que filtra por tipo especifico.

```
def saveFavourite(request):
    fav = translator.fromTemplateIntoCard(request)
    fav.user = get_user(request)

    return repositories.save_favourite(fav)
```

Función que guarda un Pokemon como favorito para el usuario registrado.

```
def deleteFavourite(request):
    fav_id = request.POST.get('id')
    return repositories.delete_favourite(fav_id)
```

Elimina un Pokemon de la lista de favoritos del usuario.

```
def get_type_icon_url_by_name(type_name):
    type_id = config.TYPE_ID_MAP.get(type_name.lower())
    if not type_id:
        return None
    return transport.get_type_icon_url_by_id(type_id)
```

Obtiene la URL del icono de un tipo de Pokemon por su nombre.

Archivo **transport.py**

Este archivo maneja toda la comunicación con APIs externas.

```
def getAllImages():
    with _cache_lock:
        if _cache:
            print("[transport.py]: Usando cache de Pokémon")
            return list(_cache.values())

    print("[transport.py]: Obteniendo datos de la API...")
    json_collection = []

    with concurrent.futures.ThreadPoolExecutor(max_workers=MAX_WORKERS) as executor:
        future_to_id = {
            executor.submit(fetch_single_pokemon, pokemon_id): pokemon_id
            for pokemon_id in range(POKEMON_RANGE[0], POKEMON_RANGE[1])
        }

        for future in concurrent.futures.as_completed(future_to_id):
            pokemon_id = future_to_id[future]
            try:
                pokemon_data = future.result()
                if pokemon_data:
                    json_collection.append(pokemon_data)
            except Exception as exc:
                print(f"[transport.py]: Error al obtener Pokémon {pokemon_id}: {exc}")

    json_collection.sort(key=lambda x: x.get('id', 0))

    with _cache_lock:
        for pokemon in json_collection:
            _cache[pokemon['id']] = pokemon

    print(f"[transport.py]: Obtenidos {len(json_collection)} Pokémon")
    return json_collection
```

Una función que obtiene todos los datos de los Pokemon desde la API. Implementaba para optimizar el código y evitar múltiples peticiones simultáneas a la API.

```
def fetch_single_pokemon(pokemon_id):
    """Función auxiliar para obtener un solo Pokémon"""
    try:
        response = requests.get(
            config.STUDENTS_REST_API_URL + str(pokemon_id),
            timeout=REQUEST_TIMEOUT
        )

        if not response.ok:
            print(f"[transport.py]: Error al obtener datos para el id {pokemon_id}")
            return None

        raw_data = response.json()

        if 'detail' in raw_data and raw_data['detail'] == 'Not found.':
            print(f"[transport.py]: Pokémon con id {pokemon_id} no encontrado.")
            return None

        return raw_data
```

```
    except requests.exceptions.Timeout:
        print(f"[transport.py]: Timeout para el id {pokemon_id}")
        return None
    except requests.exceptions.RequestException as e:
        print(f"[transport.py]: Error de red para el id {pokemon_id}: {e}")
        return None
    except Exception as e:
        print(f"[transport.py]: Error inesperado para el id {pokemon_id}: {e}")
        return None
```

Función auxiliar para obtener un solo Pokemon de la API. Realiza una petición individual a la API para obtener los datos de un Pokemon específico.

```
def get_type_icon_url_by_id(type_id):
    base_url = 'https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/types/generation-iii/colosseum/'
    return f"{base_url}{type_id}.png"
```

Función que construye la URL del icono de un tipo de Pokemon por su ID.

```
def clear_cache() -> None:
    """
    Limpia el cache de Pokémon.

    Útil para forzar una nueva obtención de datos de la API.
    """
    with _cache_lock:
        _cache.clear()
    print("[transport.py]: Cache limpiado")
```

Limpia el cache. Útil para forzar una nueva obtención de datos de la API.

```
def get_cache_size() -> int:
    """
    Obtiene el tamaño actual del cache.

    Returns:
        int: Número de Pokémon en cache
    """
    with _cache_lock:
        return len(_cache)
```

Obtiene el tamaño actual del cache.

Archivo **Models.py**

Este archivo define la estructura base de datos para almacenar los Pokemon favoritos de los usuarios registrados.

```
    def __str__(self):
        return f"{self.name} (ID: {self.pokemon_id}) - {self.user.username}"
```

Representación en string del objeto.

Archivo **repositories.py**

Este archivo contiene rodas las operaciones de base de datos para la aplicación.

```
def save_favourite(fav):
    try:
        fav = Favourite.objects.create(
            name=fav.name,  # Nombre del personaje
            pokemon_id=fav.id,  # ID de pokeapi
            types=fav.types,  # tipos
            height=fav.height,  # altura
            weight=fav.weight,  # peso
            image=fav.image,  # Imagen
            user=fav.user  # Usuario autenticado
        )
        return fav
    except IntegrityError as e:
        print(f"Error de integridad al guardar el favorito: {e}")
        return None
    except KeyError as e:
        print(f"Error de datos al guardar el favorito: Falta el campo {e}")
        return None
```

Guarda un Pokemon favorito en la base de datos.

```
def get_all_favourites(user):
    return list(Favourite.objects.filter(user=user).values(
        'id', 'pokemon_id', 'name', 'height', 'weight', 'types','base_experience', 'image'
    ))
```

Obtiene todos los favoritos de un usuario específico.

```
def delete_favourite(fav_id):
    try:
        favourite = Favourite.objects.get(id=fav_id)
        favourite.delete()
        return True
    except Favourite.DoesNotExist:
        print(f"El favorito con ID {fav_id} no existe o no pertenece al usuario.")
        return False
    except Exception as e:
        print(f"Error al eliminar el favorito: {e}")
        return False
```

Elimina un favorito específico de la base de datos.

Archivo **card.py**

Clase card, actua como un modelo de datos unificado que representa un Pokemon en diferentes contextos de la aplicación.

```
class Card:
    def __init__(self, name, height, base, weight, image, types, user=None, id=None, type_images=None,
is_favourite=False):
```

```
        self.name = name  # Nombre del pokemon
        self.height = height  # ALTURA
        self.weight = weight  # PESO
        self.base = base  # NIVEL BASE
        self.image = image  # URL de la imagen
        self.user = user  # Usuario asociado (si corresponde)
        self.id = id  # ID único (si corresponde)
        self.types = types or []  # Asegura que sea una lista por defecto
        self.type_images = type_images or []
        self.is_favourite = is_favourite
```

Inicializa una nueva instancia de Card.

```
    def __str__(self):
        return (f'name: {self.name}, height: {self.height}, weight: {self.weight}, '
                f'base: {self.base}, image: {self.image}, user: {self.user}, id: {self.id}')
```

Representación en string del objeto Card.

```
    def __eq__(self, other):
        if not isinstance(other, Card):
            return False
        return (self.name, self.height, self.weight, self.id) == \
               (other.name, other.height, other.weight, other.id)
```

Método para comprar dos objetos Card.

```
    def __hash__(self):
        return hash((self.name, self.height, self.weight, self.id))
```

Metodo para permitir usar Card en conjuntos y diccionarios.

Archivo **translator.py**

Archivo para mapear datos de un formato o estructura a otro.

```
def fromRequestIntoCard(poke_data):
    types = getTypes(poke_data)
    type_images = [services.get_type_icon_url_by_name(type_name) for type_name in types]

    card = Card(
        id=poke_data.get('id'),  # id de pokeapi, corresponde a pokemon_id en la base de datos
        name=poke_data.get('name'),
        height=poke_data.get('height'),
        weight=poke_data.get('weight'),
        base=poke_data.get('base_experience'),
        image=safe_get(poke_data, 'sprites', 'other', 'official-artwork', 'front_default'),
        types=types,
        type_images=type_images
    )
    return card
```

Convierte dato de la API a un objeto Card.

```python
def getTypes(poke_data):
    types = []
    for type in poke_data.get('types'):
        t = safe_get(type, 'type','name' )
        types.append(t)
    return types
```
Extrae la lista de tipos de un Pokemon desde los datos de la API.

```python
def fromTemplateIntoCard(templ):
    card = Card(
        name=templ.POST.get("name"),
        id=templ.POST.get("id"),  # id de pokeapi, corresponde a pokemon_id en la base de datos
        height=templ.POST.get("height"),
        weight=templ.POST.get("weight"),
        types=templ.POST.get("types"),
        base=templ.POST.get("base"),
        image=templ.POST.get("image")
    )
    return card
```
Convierte datos en un formulario HTML en un objeto Card.

```python
def fromRepositoryIntoCard(repo_dict):
    types_list = repo_dict.get('types', [])

    if isinstance(types_list, str):
        try:
            types_list = ast.literal_eval(types_list)
        except (ValueError, SyntaxError) as e:
            print(f"Error al evaluar types: {e}, usando lista vacía")
            types_list = []

    return Card(
        id=repo_dict.get('pokemon_id'),  # id de pokeapi, corresponde a pokemon_id en la base de datos
        name=repo_dict.get('name'),
        height=repo_dict.get('height'),
        weight=repo_dict.get('weight'),
        base=repo_dict.get('base_experience'),
        types=types_list,
        image=repo_dict.get('image')
    )
```
Convierte datos de la base de datos en un objeto Card.

```python
def safe_get(dic, *keys):
    for key in keys:
        if not isinstance(dic, dict):
            return None
        dic = dic.get(key, {})
    return dic if dic else None
```
Obtiene un valor de un diccionario anidado de forma segura.

Dificultades encontradas

Al principio nos costó comprender exactamente qué era lo que había que hacer. La frase "¿por dónde empiezo?" seguramente rondó por nuestras cabezas más de una vez, hasta que finalmente le consultamos a un compañero que nos explicó cómo importar funciones desde otros archivos.

La cantidad de información disponible al inicio fue abrumadora y confusa, pero a medida que avanzamos en la programación, todo comenzó a tener más sentido.

Una de las mayores dificultades fue lograr que un valor de app_favourite.id pudiera estar asociado a más de un usuario. También tuvimos varios problemas al momento de actualizar las migraciones, lo cual generó errores y nos llevó bastante tiempo resolverlo.

Aspectos pendientes

Hubiera sido interesante incorporar al sistema de registro una función para enviar un correo electrónico de confirmación. Sin embargo, debido a la falta de tiempo, no pudimos implementarla en esta etapa.

Conclusión final

Este trabajo práctico representó un salto importante en cuanto a nivel de dificultad. Para muchos integrantes del grupo, fue la primera experiencia con un proyecto de estas características, mientras que algunos ya contaban con experiencia previa que resultó de gran ayuda para resolver ciertos problemas.

Lo más complejo fue, sin duda, lograr que una misma función pudiera asociar correctamente un identificador a varios usuarios. En contraste, algunas partes del diseño HTML resultaron más sencillas gracias a conocimientos previos que ya teníamos.

Especificaciones:

El integrante Patrick Stivic realizo gran parte del código necesario para el funcionamiento del registro. Durante el desarrollo del código se lo paso al integrante Joaquín Benítez para que este pudiera implementarlo en su código y terminar el register.html por lo que al final el commit fue realizado desde su computadora.

Los integrantes Roberto Duarte y Valentino Forastello no contaron con el uso de computadoras propias o se les dificulto ciertos accesos. Pero pudieron trabajar desde la biblioteca para realizar la importación de las imágenes de los tipos de los Pokemon para que fueran visibles en lugar de solo mostrar texto.