TRABAJO PRÁCTICO POKEDEX

Introducción a la programación

Primer semestre - 2025

García Sol y Antonucci Leonel

Contenido

Introducción

En este trabajo practico desarrollamos una página web utilizando Django. El objetivo principal es completar y mejorar una app parcialmente desarrollada, agregando las funcionalidades necesarias para que sea totalmente funcional y visualmente atractiva para el usuario.

A lo largo del desarrollo se trabajó en equipo, distribuyendo las tareas y haciendo uso de GitHub.

Desarrollo

En esta sección se detallan las funciones desarrolladas, los templates modificados y los ajustes realizados en las distintas capas del proyecto para lograr el funcionamiento completo de la aplicación. A medida que avanzó el trabajo, se fueron implementando componentes en los archivos views.py, services.py, y en los templates, con el objetivo de mostrar correctamente la información proveniente de la API en formato de tarjetas (cards). Además, se aplicaron condicionales para mejorar la visualización según el tipo de Pokémon y se preparó la estructura para incorporar funcionalidades opcionales.

Services.py

A continuación, se detallarán las funciones implementadas en services.

services.getAllImages: Esta función se encarga de conseguir la información de los pokemones mediante la función getAllImages en transport.py, para luego devolverlos en forma de lista. Primero se intentó desarrollar la función para obtener los datos desde la misma, pero luego se la corrigió para que usara la función presente en el archivo transport.py.

```
def getAllImages():
    json_collection = transport.getAllImages()
    cards = []

    for pokemon in json_collection:
        card = translator.fromRequestIntoCard(pokemon)
        cards.append(card)

    return cards
```

services.filterByCharacter: Se implementó una función que permite filtrar un listado de todas las tarjetas cuya propiedad name (nombre del Pokémon) contenga una subcadena determinada, sin distinguir entre mayúsculas y minúsculas. De esta manera se puede buscar tarjetas cuyo nombre coincida total o parcialmente con el buscado.

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

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

    return filtered_cards
```

services.filterByType: Se implementó una función que permite filtrar un listado de todas las tarjetas cuya propiedad type (tipo del Pokémon) contenga una cadena determinada, sin distinguir entre mayúsculas y minúsculas. De esta manera se puede buscar tarjetas cuyo tipo coincida con el ingresado.

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

    for card in getAllImages():
        for type in card.types:
            if type_filter in type.lower():
                filtered_cards.append(card)

    return filtered_cards
```

services.getAllFavourites: Se completó la función que permite recuperar a través del uso del translator los favoritos del usuario loggeado desde la base de datos.

```
def getAllFavourites(request):
    if not request.user.is_authenticated:
        return []
    else:
        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 pokemon in favourite_list:
            card = translator.fromRepositoryIntoCard(pokemon)
```

```
            mapped_favourites.append(card)
        return mapped_favourites
```

services.register_user: Esta función es la encargada de la creación de nuevas cuentas, si tanto el nombre de usuario como la dirección de correo electrónico están disponibles guarda los datos, y sino devuelve los errores.

```
def register_user(usuario):
    username = usuario['username']
    email = usuario['email']
    password = usuario['password']
    name = usuario['name']
    surname = usuario['surname']

    errores = []

    if User.objects.filter(username = username).exists():
        errores.append("Ese nombre de usuario ya está en uso.")

    if User.objects.filter(email = email).exists():
        errores.append("Esa dirección de correo electrónico ya está asociada a
otra cuenta.")

    if errores:
        return errores

    User.objects.create_user(
        username = username,
        email = email,
        password = password,
        first_name = name,
        last_name = surname,
    )

    return errores
```

Views.py

views.home: Esta función hace uso de los servicios getAllImage y getAllFavourites para enviárselos al template para que dibuje los todos los pokemones y además identifique si están o no en los favoritos.

```
def home(request):
    images = []
```

```
    favourite_list = []
    favourite_list_name = []

    images = services.getAllImages()
    favourite_list = services.getAllFavourites(request)
    for pokemon in favourite_list:
        favourite_list_name.append(pokemon.name)
    return render(request, 'home.html', { 'images': images,
'favourite_list_name': favourite_list_name })
```

views.search y views.filter_by_type: Estas funciones hacen uso de los servicios para filtrar el listado por nombre y tipo respectivamente y devuelven una lista de los pokemones filtrados para que estos puedan ser dibujados por el template.

```
# función utilizada en el buscador.
def search(request):
    name = request.POST.get('query', '')

    # si el usuario ingresó algo en el buscador, se deben filtrar las imágenes
por dicho ingreso.
    if (name != ''):
        images = services.filterByCharacter(name)
        favourite_list = []

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

# función utilizada para filtrar por el tipo del Pokemon
def filter_by_type(request):
    type = request.POST.get('type', '')

    if type != '':
        images = services.filterByType(type) # debe traer un listado filtrado
de imágenes, segun si es o contiene ese tipo.
        favourite_list = []

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

views.subscribe: Toma los datos del formulario y si son válidos llama a la función register_user, si hubo algún problema se le muestra al usuario por pantalla, y si no se le envía un mail con las credenciales de inicio de sesión.

```
def subscribe(request):
    form = SubscribeForm()
    if request.method == 'POST':
        form = SubscribeForm(request.POST)
        if form.is_valid():
            usuario = form.cleaned_data
            errores = register_user(form.cleaned_data)

            if errores:
                for error in errores:
                    messages.error(request,error)
            else:
                username = usuario['username']
                email = usuario['email']
                password = usuario['password']
                subject = 'Registro exitoso'
                message = f'¡Gracias por registrarte {username}!\nEstas son
tus credenciales de inicio de
sesión:\nUsuario:{username}\nContraseña:{password}'
                send_mail(subject, message, settings.EMAIL_HOST_USER, [email],
fail_silently=False)
                return redirect('loading_home')
    return render(request, 'registration/register.html', {'form': form})
```

views.getAllFavouritesByUser: Utiliza el servicio getAllFavourite para recuperar los favoritos y que el template pueda dibujarlos.

```
def getAllFavouritesByUser(request):
    favourite_list = []
    images = []
    favourite_list = services.getAllFavourites(request)


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

views.saveFavourite y views.deleteFavourite: Estas dos funciones se encargan de llamar al servicio correspondiente encargado de guarda o quitar un Pokémon de favoritos y luego te redirecciona a la lista de favoritos para poder ver los cambios reflejados.

```
@login_required
def saveFavourite(request):
    services.saveFavourite(request)
    return redirect('favoritos')
```

```
@login_required
def deleteFavourite(request):
    services.deleteFavourite(request)
    return redirect('favoritos')
```

Forms.py

forms.subscribeForm: Define los campos que deben completarse en el formulario al momento de registrarse.

```
class SubscribeForm(forms.Form):
    username = forms.CharField()
    name = forms.CharField()
    surname = forms.CharField()
    password = forms.CharField()
    email = forms.EmailField()
```

Templates

Se modificó home.html para que este muestre los marcos de los pokemones de color según su tipo, variando el tipo de marco.

```
<div class="card mb-3 ms-5
                    {% if img.types.0 == 'Fire' %}
                        border-danger
                    {% elif img.types.0 == 'Water' %}
                        border-primary
                    {% elif img.types.0 == 'Grass' %}
                        border-success
                    {% else %}
                        border-warning
                    {% endif %}"
                    style="max-width: 540px;">
```

Se agregó al header.html un spinner de carga y un script para se muestre antes descargar la página, de esta manera al descargase las páginas si la transición a la siguiente tardaba, este se mostraría durante ese proceso.

```
<div id="loading_home" style="display: none; position: fixed; top: 0; left: 0;
width: 100vw; height: 100vh; background-color: rgba(255,255,255,0.8); z-index:
9999; text-align: center; padding-top: 200px;">
        <span class="spinner-border text-primary" role="status"></span>
        <p>Cargando...</p>
</div>

    <script>
```

```
        window.addEventListener("beforeunload", () => {
        const spinner = document.getElementById("loading_home");
        if (spinner) {
            spinner.style.display = "block";
            }
        });
    </script>
```

Además, se agregó un template utilizado para el registro de usuarios (register.html) y se hicieron cambios en el archivo main/settings.py para implementar el envío de mails al momento de registrar la cuenta.

Mejora gráfica

Se agregó la carpeta media en static, en donde se agregaron las imágenes utilizadas tanto para el cursor como para el fondo de la página.

Se modificó home.html para que este muestre el fondo del tipo de pokemon con su color correspondiente.

```
<div class="alert alert-warning alert-dismissible fade show" role="alert">
     {% for poketype in img.types %}
          <span class="badge type-{{ poketype|lower }}">{{ poketype }}</span>
     {% endfor %}
</div>
```

Para esto se debieron además agregar en el archivo styles.css, colores para cada uno de los tipos que aparecen:

```
.type-fire {
    background-color: #f08030;
}

.type-water {
    background-color: #6890f0;
}

.type-grass {
    background-color: #78c850;
}

.type-poison {
    background-color: #a040a0;
}

.type-electric {
```

```
    background-color: #f8d030;
}

.type-ground {
    background-color: #ad8e37;
}

.type-flying {
    background: linear-gradient(to right, #6890f0, #d1d1d1);
}

.type-bug {
    background-color: #a8b820;
}

.type-normal {
    background-color: #9e9e9e;
}
```

Por otro lado, con el objetivo de que la página fuera responsive se hizo uso de Media Queries para que se adaptara a dispositivos con pantallas grandes, dispositivos con pantallas medianas y dispositivos con pantallas pequeñas:

```
@media (min-width: 992px) {
    .navbar-expand-lg .navbar-nav .nav-link {
        padding-right: 1rem;
        padding-left: 1rem;
    }
}

@media (min-width: 992px) {
    .navbar .nav-link {
        height: 100%;
        display: flex;
        align-items: center;
        justify-content: center;
        line-height: 1;
        transition: background-color 0.3s ease;
    }
    .navbar .navbar-nav,
    .navbar .nav-item {
        height: 100%;
    }
    .navbar {
        height: 60px;
        padding-top: 0;
```

```
        padding-bottom: 0;
    }
}

@media (max-width: 992px) {
    .navbar .nav-link {
        text-align: center;
        padding-top: 0.7rem;
        padding-bottom: 0.7rem;
        padding-bottom: .5rem;
        width: 100%;
    }
}
```

Aparte de eso se realizaron cambios menores en el header, en las tarjetas de los pokemones y en el sitio en general con el objetivo que quedara más estético. Como lo son el cambio en la fuente utilizada, efecto hover en la navbar y demás.

Dificultades

Durante el desarrollo de este trabajo surgieron diversas dificultades que fueron resolviéndose progresivamente. En un principio, no utilizamos la función translate, lo que permitió mostrar las imágenes correctamente, pero generó inconvenientes posteriormente al intentar detectar qué Pokémon ya se encontraban guardados como favoritos. Esta situación fue corregida implementando el uso adecuado de translate.

Además, con el avance del proyecto, identificamos que algunas funcionalidades que inicialmente estaban definidas en las views resultaban más apropiadas para ser gestionadas desde el módulo de services, lo cual permitió mejorar la organización, legibilidad y mantenimiento del código.

Conclusión

El desarrollo de esta aplicación web utilizando Django nos permitió profundizar en el uso de distintos componentes del framework, como las vistas, servicios, formularios y templates, además de integrar información proveniente de una API externa. A lo largo del trabajo, fuimos enfrentando y resolviendo diversos desafíos que nos ayudaron a mejorar la estructura del proyecto y la organización del código.

Implementar buenas prácticas como la separación de responsabilidades entre las capas (*views*, *services*, *utilities*, etc.) nos permitió tener un código más legible.

El uso de GitHub fue fundamental para facilitar el trabajo colaborativo, permitiéndonos organizarnos como equipo, distribuir tareas y mantener un control de versiones

efectivo durante todo el desarrollo. Esta dinámica de trabajo nos permitió adaptarnos ante los distintos obstáculos que surgieron y aplicar mejoras de forma ordenada y progresiva. Como resultado, logramos una aplicación funcional, visualmente atractiva y alineada con los requerimientos del trabajo práctico.