TRABAJO PRÁCTICO FINAL

INTRODUCCIÓN A LA PROGRAMACIÓN

Integrantes:Francisco Batistuta y Agustina Juarez
Comisión: 11

Profesores:
Luca Velazquez
Omar Argañaras

Fecha de entrega: 23/06/25

Introducción

El siguiente trabajo consiste en el desarrollo de una aplicación web interactiva que permite visualizar, buscar y clasificar personajes, similar a una Pokédex. Los usuarios tienen la posibilidad de filtrar personajes por tipo, buscarlos por nombre y guardar sus favoritos si han iniciado sesión. Para llevar a cabo estas funcionalidades, se programaron distintas vistas y servicios que procesan los datos y controlan la forma en que se presentan en la interfaz.

Funciones desarrolladas para el funcionamiento de la aplicación:

Para una mejor percepción de las funciones desarrolladas, serán explicadas por archivo en el que se encuentran dichas funciones.

Funciones de views:

Las funciones utilizadas en las que fueron desarrolladas dentro de este archivo, fueron importadas desde el archivo de python de services, entre otros. Dicho archivo con sus funciones, será explicado más adelante.

- ★ **Función** home:

Código:

```
def home(request):
    images = services.getAllImages()
    favourite_list = services.getAllFavourites(request)  # solo devuelve favoritos si está
autenticado
    return render(request, 'home.html', { 'images': images, 'favourite_list': favourite_list })
```

Esta función solicita se activa por el usuario y recibe una petición del html la cual importa una lista de imágenes API de la función llamada (getallimages) en el módulo services y también importa todas las imágenes de la función (getallfavourites) del módulo services la cual genera una lista de imágenes favoritas del usuario si este se encuentra logueado al final en el return retorna las imágenes que encontró en cada lista

- ★ **Función** search:

Código:

```
def search(request):
   name = request.POST.get('query', '').strip()
    if name != '':
        images = services.filterByCharacter(name)
        favourite_list = services.getAllFavourites(request)
        return render(request, 'home.html', { 'images': images, 'favourite_list': favourite_list })
    else:
        return redirect('home')
```

Permite buscar personajes por nombre desde el buscador de la página. Toma lo que escribió el usuario, elimina espacios y, si hay algo escrito, filtra los resultados que coincidan. A continuación, muestra los personajes encontrados junto con los favoritos del usuario (si está logueado). En caso de que no se haya ingresado ningún texto, vuelve automáticamente a la página principal.

★ **Función** filter_by_type:

Código:

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

    if type != '':
        images = services.filterByType(type)
        favourite_list = services.getAllFavourites(request)
        return render(request, 'home.html', { 'images': images, 'favourite_list': favourite_list })
    else:
        return redirect('home')
```

Ofrece la posibilidad de filtrar personajes por tipo (como fuego, agua o planta). Toma el tipo seleccionado por el usuario, lo limpia por si tiene espacios extra, y si está completo, muestra todos los personajes que coincidan, junto con los favoritos si el usuario está autenticado. Si no se seleccionó nada, redirige al inicio.

★ **Función** register:

Código:

```
def register(request):
    if request.method == 'POST':
        form = RegisterForm(request.POST)
        if form.is_valid():
            user = form.save()
            # Enviar correo
            try:
                subject = '¡Bienvenido a Pokedex!'
```

```
                from_email = settings.EMAIL_HOST_USER
                recipient = user.email
                context ={
                    'user': user,
                      'message': f'Hola {user.first_name}! Bienvenido a Pokedex, tu cuenta ha sido creada exitosamente.'

                }

                html_content = render_to_string('registration/email.html', context)
                  text_content = f'Hola {user.first_name}! Bienvenido a Pokedex, tu cuenta ha sido creada exitosamente. link: http://127.0.0.1:8000/login'
                email = EmailMultiAlternatives(subject, text_content, from_email, [recipient])
                email.attach_alternative(html_content, "text/html")
                email.send()
            except Exception as e:
                 messages.success(request, 'Cuenta creada correctamente. Te enviamos un email de bienvenida.')
            return redirect('login')
    else:
        form = RegisterForm()
    return render(request, 'registration/register.html', {'form': form})
```

Esta vista se encarga de registrar nuevos usuarios en la aplicación. Primero verifica que el método de la solicitud sea POST, lo que indica que el formulario fue enviado. Si es así, crea una instancia del formulario RegisterForm con los datos recibidos.

Luego, valida el formulario. Si los datos están bien cargados, se guarda el nuevo usuario en la base de datos. Justo después, se prepara y envía un correo de bienvenida.

El mensaje tiene dos versiones: una en texto plano y otra en HTML. La versión HTML se genera a partir de una plantilla (email.html) que se completa con los datos del usuario (como su nombre). El correo se envía a la dirección que el usuario ingresó durante el registro.

Finalmente, se muestra un mensaje de éxito y se redirige al usuario a la página de login. En caso de que el formulario no se haya enviado aún (es decir, si el método es GET), simplemente se muestra el formulario de registro vacío.

★ **Función** getAllFavouritesByUser:

Código:

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

Muestra la página de favoritos del usuario. Solo se accede si tiene sesión iniciada, y se encarga de obtener y enviar al template la lista completa de los personajes que marcó como favoritos.

★ **Función** saveFavourite:

Código:

```
@login_required
def saveFavourite(request):
    if request.method == 'POST':
        services.saveFavourite(request)
        favourites = Favourite.objects.filter(user=request.user)
        favourite_list = [f.name for f in favourites]
        pokemons = services.getAllImages()
        return render(request, 'home.html', {
            'images': pokemons,
            'favourite_list': favourite_list,
        })
    else:
        return redirect('home')
```

Se utiliza para guardar o quitar un personaje de los favoritos. Solo funciona si el usuario está logueado y si la solicitud viene desde un formulario. Después de guardar el cambio, recarga tanto la lista de favoritos como la lista de personajes, y muestra todo actualizado en la página principal.

★ **Función** deleteFavourite:

Código:

```
@login_required
def deleteFavourite(request):
    if request.method == 'POST':
        success = services.deleteFavourite(request)
        return redirect('favoritos')
    else:
        return redirect('favoritos')
```

Elimina un personaje de la lista de favoritos. Esta acción se hace solo cuando se envía un formulario específico, y al terminar, lleva al usuario de vuelta a la página de favoritos para que vea los cambios reflejados.

Funciones de forms:

Aquí se creó el formulario necesario para el registro de nuevos usuarios, posteriormente importado al archivo de views.

```
class RegisterForm(UserCreationForm):
    email = forms.EmailField()
    class Meta:
        model = User
        fields = ('username', 'first_name', 'last_name', 'email', 'password1', 'password2')
    def save(self, commit=True):
        user = super().save(commit=False)
        user.first_name = self.cleaned_data['first_name']
        user.last_name = self.cleaned_data['last_name']
        user.email = self.cleaned_data['email']
        if commit:
            user.save()
        return user
```

Este formulario personalizado se usa para registrar nuevos usuarios. Está basado en un formulario que ya viene con Django (UserCreationForm), pero se le agregan campos extra como nombre, apellido y correo electrónico, que no vienen incluidos por defecto.

La clase Meta indica que este formulario está relacionado con el modelo User (el modelo de usuario de Django) y especifica qué campos se van a pedir en el formulario.

La función save() toma los datos que el usuario escribió en el formulario, los guarda en el objeto user, y luego lo guarda en la base de datos (si se indica commit=True). De esta forma, además del nombre de usuario y la contraseña, también se guardan el nombre, el apellido y el correo electrónico.

Funciones de services:

- ★ **Función** getAllImages:

Código:

```
def getAllImages():
    raw_images = transport.getAllImages()
```

```
    cards = []

    for raw in raw_images:
        card = translator.fromRequestIntoCard(raw)
        cards.append(card)

    return cards
```

esta función solicita un listado de imágenes API de la función (getallimages) del módulo trasport las cuales utilizando el (for raw in raw_images:) para ir transformado el raw que es la imagen en crudo y las va transformando en card para utilizar en la página las cuales se van almacenando en la lista de (card) y luego esta lista son las que se devuelven en el return de la función

★ **Función** filterByCharacter:

Código:

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

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

    return filtered_cards
```

Permite buscar personajes cuyos nombres contengan lo que el usuario escribió, sin importar si usó mayúsculas o minúsculas. Recorre todas las cartas disponibles y selecciona aquellas que coinciden parcialmente con el texto ingresado.

★ **Función** filterByType:

Código:

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

    for card in getAllImages():
        if type_filter.lower() in [t.lower() for t in card.types]:
            filtered_cards.append(card)

    return filtered_cards
```

Sirve para filtrar personajes según su tipo (como fuego, agua, etc.). Compara el tipo seleccionado por el usuario con la lista de tipos de cada personaje, teniendo en cuenta que algunos pueden tener más de uno. Si coincide, lo incluye en el resultado.

★ **Función** saveFavourite:

Código:

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

Se encarga de guardar un personaje como favorito del usuario. Primero convierte los datos enviados desde el formulario en un formato que el sistema entiende, los asocia al usuario actual y luego realiza el guardado en la base de datos.

★ **Función** getAllFavourites:

Código:

```
def getAllFavourites(request):
    if not request.user.is_authenticated:
        return []

    user = get_user(request)
    favourite_list = repositories.get_all_favourites(user)
    mapped_favourites = []

    for favourite in favourite_list:
        card = translator.fromRepositoryIntoCard(favourite)
        mapped_favourites.append(card)

    return mapped_favourites
```

Recupera todos los personajes marcados como favoritos por el usuario que tiene la sesión iniciada. Para hacerlo, primero verifica que esté logueado, luego busca los datos almacenados y los convierte en un formato que se puede mostrar en pantalla.

★ **Función** deleteFavourite:

Código:

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

Quita un personaje de la lista de favoritos. Para lograrlo, toma el identificador enviado desde el formulario y lo utiliza para eliminar el registro correspondiente en la base de datos.

Código implementado en Home.html:

Botón de agregar favoritos:

```
<input type="hidden" name="image" value="{{ img.image }}">
{% if img.name in favourite_list %}
    <button type="submit" class="btn btn-primary btn-sm" disabled>✔️ Favoritos</button>
{% else %}
    <button type="submit" class="btn btn-primary btn-sm">❤️ Favoritos</button>
{% endif %}
```

Se encarga de mostrar un botón para agregar un Pokémon a favoritos. Si el personaje ya está en la lista de favoritos del usuario, el botón aparece desactivado y con un tilde para indicar que ya fue marcado. En cambio, si todavía no lo está, se muestra con un corazón para que el usuario lo pueda agregar. Gracias a esta lógica, se evita que el mismo Pokémon se agregue dos veces.

Borde de las cards:

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

Modifica el borde de color de cada tarjeta según el tipo del Pokémon. Por ejemplo:

- ★ Si es de tipo fuego, el borde se pone rojo (border-danger).
- ★ Si es de agua, azul (border-primary).
- ★ Si es de planta, verde (border-success).
- ★ Para cualquier otro tipo, se usa un borde amarillo (border-warning).

Esto ayuda a distinguir visualmente los tipos desde el primer vistazo, haciendo la interfaz más intuitiva y atractiva.

- ★ **Spinner de carga:**

```
<div id="spinner"></div>

<style>
    /* Spinner siempre visible mientras el body tenga la clase 'preloading' */
    #spinner {
        position: fixed;
        top: 50%;
        left: 50%;
```

```
        z-index: 9999;
        width: 50px;
        height: 50px;
        margin: -25px 0 0 -25px;
        border: 5px solid #f3f3f3;
        border-top: 5px solid #3498db;
        border-radius: 50%;
        animation: spin 1s linear infinite;
        display: block;
    }

    @keyframes spin {
        0% { transform: rotate(0deg); }
        100% { transform: rotate(360deg); }
    }

    /* Ocultar el spinner cuando el body ya está cargado */
    body.loaded #spinner {
        display: none;
    }
</style>

<script>
    window.addEventListener('load', () => {
        document.body.classList.remove('preloading');
        document.body.classList.add('loaded');
    });

    // Mostrar spinner en cada envío de formulario
    document.querySelectorAll('form').forEach(form => {
        form.addEventListener('submit', () => {
            document.body.classList.remove('loaded');
            document.body.classList.add('preloading');
        });
    });
</script>
```

Se trata de una pequeña animación en forma de ruedita giratoria que aparece en el centro de la pantalla mientras la página está cargando o cuando se envía un formulario. Sirve para que el usuario sepa que el sistema está procesando información, y así evitar que piense que la aplicación está congelada o que haga clics múltiples por error.

Dificultades encontradas y decisiones para afrontarlas:

Implementar código externo a nuestro código y poder adaptarlo para que funcione correctamente, establecer variables en conjunto para las funciones, implementar nuevas tecnologías al proyecto. Para la resolución de estas dificultades, nos organizamos con comunicación, búsqueda de información sobre las tecnologías y los temas que desconocemos.

Cosas que no hubiera gustado agregar:

Lo único que consideramos que nos faltó para poder haber terminado el trabajo al 100% es la confirmación de los datos del usuario a través del correo ingresado, para poder cargarlo en la página y que quede registrado como nuevo usuario.

Conclusión:

Para el trabajo desarrollado, aprendimos distintas tecnologías y aspectos a desarrollar con python, las cuales dificultaron en parte el desarrollo del mismo que con la información y comunicación entre nosotros, se pudo resolver y llegar a obtener el resultado esperado.