Universidad Nacional de General Sarmiento

Licenciatura en Sistemas

Trabajo Práctico
Pokedex

Introducción a la Programación

(primer semestre 2025)

Resumen: Este trabajo consiste en poner en funcionamiento una aplicación web para buscar imágenes utilizando Django.

Integrantes: Pablo Rosset - *pablorosseth@gmail.com*
Victoria Rodríguez - vickrodriguez0018*@gmail.com*

Se presentó una aplicación la cual ya estaba mayormente desarrollada, excepto ciertas funciones. Para completar su funcionalidad, se precisa completar y desarrollar estas mismas. De esta manera, se hizo uso de condicionales, ciclos y listas para que la aplicación funcione correctamente.

En general, se desarrollaron funciones en services.py para ser utilizadas en views.py. De esta manera, las funciones en views.py retornan la respuesta necesaria a los templates, permitiendo la visualización y funcionamiento correcto de la aplicación.

Las siguientes son las principales funcionalidades:

- **Mostrar la galería**
  Se completo utilizando las funciones *home(request)* y *getAllImages()*

  ```
  def getAllImages():
      API_list = transport.getAllImages()
      card_list = []
      for image in API_list:
          card = translator.fromRequestIntoCard(image)
          card_list.append(card)
      return card_list
  ```

  Esta función devuelve una lista de cards con datos de la API.
  Para hacer esto, hace uso de la función getAllImages de transport.py para traer una lista de objetos Json de la API.
  Luego recorre esa misma lista y "transforma" a cada objeto en una card, utilizado la función fromRequestIntoCard.

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

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

  Esta función le asigna la lista de cards a una variable y la devuelve al template home, para que las imágenes sean visibles en la página.

- **Favoritos**
  Esta funcionalidad permite al usuario elegir imágenes favoritas y guardarlas en la sección "Favoritos", de esta manera se desactiva el botón de favoritos de dicha imagen. Al eliminar el favorito, el botón vuelve a activarse.

  Las funciones utilizadas son:

  En services.py:

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

      return repositories.save_favourite(fav)
  ```

Recibe una request desde el home template, transforma la información de este request en una card, y consigue el user para asociarlo al favorito. Luego retorna fave_favourite(fav) de repositories.py, y asi guarda el favorito en la DB.

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

Recibe una request y de esta request consigue el user. Busca todos los favoritos guarados en la DB de ese user, y los agrega en una lista. Luego transforma cada favorito en una card y los agrega en una lista vacia, que luego se retorna.

En views.py:

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

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

Crea una lista con todos los favoritos en formato card (usando la función anterior) y los retorna a la sección de favoritos, para que sean visibles en pantalla.

```
def saveFavourite(request):
    services.saveFavourite(request)
    return redirect ('home')
```

Implementa la función saveFavourite de services.py en el template home.

- Buscador(nombre):

En services.py

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

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

```
    return filtered_cards
```

Recibe un nombre ingresado por el usuario, luego busca entre todas las imágenes, cuales tienen el mismo nombre, y las agrega a un listado. Luego devuelve el listado.

En views.py:

```
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 })
    else:
        return redirect('home')
```

Recibe una request y de esta consigue el nombre, luego filtra las imágenes usando la función anterior, y las devuelve al template home.html

Cambio de border color:

```
{% if img.types and "fire" in img.types %}
            <div class="card border-danger mb-3 ms-5" style="max-width: 540px; ">
              <div class="row g-0">
                <div class="col-md-4">
                  <img src="{{ img.image }}" class="card-img-top"alt="imagen">
                </div>
```

En home.html se usan esas líneas de código sacadas de Bootstrap, donde con border –(color) se cambia el color del borde de la card, en este caso danger es rojo para los tipo fuego.

Filtrado por tipo:

En services.py:

```
def filterByType(type_filter):
        filtered_cards = []
        for card in getAllImages():
                if type_filter.lower() in (type.lower() for type in card.types):
                filtered_cards.append(card)
return filtered_cards
```

Esta función agarra el type_filter (agua, tierra, fuego), hace un ciclo for con todas las cartas de getAllImages(): y las compara una a una con el type_filter sin importar si están en mayúsculas o minisculas. Devuelve un listado con las cartas que tengan ese type_filter.

En views.py:

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

Eecibe una peticion y filtra un listado de imágenes según el tipo solicitado.
Para esto llama a la función filterByType(type) que le devuelve un listado con las cartas que tienen el tipo, y renderiza el listado en home.html
Alta de usuarios:
Register.html:
Se crea el archivo register.html donde se usa una plantilla que muestra un formulario de registro básico, y utiliza un post para enviar los datos a view.py.

```
<div class="register-form" style="text-align: center;">
  <form method="POST" style="display: inline-block;">
    {% csrf_token %}
    <h2 class="text-center">Registro de usuario</h2>
    {{ form.as_p }}
    <button type="submit" class="btn btn-success btn-block">Registrarse</button>
  </form>
</div>
```

Forms.py:

se añaden (from django.contrib.auth.forms import UserCreationForm) y
(from django.contrib.auth.models import User) para poder crear, modificar y borrar usuarios, y el primer import para poder modificar la plantilla de registrom agregar nombre y apellido, mail y confirmación de contraseña; esto ultimo con

```
class CustomUserCreationForm(UserCreationForm):
```

```
    class Meta:
        model = User
        fields = ('first_name','last_name','username','email', 'password1', 'password2').
```

Views.py:

Se añade la función:

```
def register(request):
    if request.method == 'POST':
        form = CustomUserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            #mail de registro
            subject = 'Registro'
            message = 'Felicitaciones!! Tu usuario fue registrado correctamente en la Pokedex'
            recipient = form.cleaned_data.get('email')
            send_mail(subject, message, settings.EMAIL_HOST_USER, [recipient],
fail_silently=False)
            return redirect('login')
    else:
        form = CustomUserCreationForm()


    return render(request, 'registration/register.html', {'form': form})
```

La función register(request) se encarga de registrar usuarios en Django. Si la petición es un POST, arma el formulario con los datos que se mandaron, revisa que todo esté bien, guarda el usuario en la base y manda un mail avisando que el registro fue exitoso. Si salió todo bien, lo manda directo al login. Si no es un POST, simplemente muestra el formulario vacío para que lo completen. Al final, siempre carga la plantilla registration/register.html con el formulario.

Urls.py:

Se añade
path('accounts/register/', views.register, name='register')

Loading Spinner:

El spinner, es decir el código html junto con el código css se sacaron de la pagina anexada al final del informe.

Estos fragmentos de código fueron incrustados en header.hmtl y styles.css
En styles.css se crearon las clases centrado y hidden que centran el spinner, le dan color al fondo y lo ocultan cuando carga la pagina.

Footer.html:

Se instala jquery con :
<script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
Y se ejecuta el archivo código.js con:
<script src="{% static 'codigo.js' %}"></script>

Código.js:

```
$(document).ready(function() {
    $('#onload').fadeOut();

    $('a.nav-link').on('click', function(e) {
        if (!$(this).attr('target') && this.href && this.href.indexOf('#') === -1) {
            $('#onload').fadeIn();
        }
    });

    $('form').on('submit', function() {
        $('#onload').fadeIn();
    });
});
```

Muestra el spinner en pantalla y cuando termina de cargar la pagina lo oculta.

Conclusiones:

Este proyecto representó el primer acercamiento real al uso de Django junto con HTML y CSS, lo que generó una cierta complejidad al inicio, principalmente por la necesidad de entender cómo se conectaban las diferentes partes del sistema y como se organizaban los archivos y las vistas. Al principio costó adaptarse a la estructura de Django y a la logica de los templates, pero a medida que se avanzaba en el proyecto la comprencion de las herramientas fue mejorando notablemente, permitiendo terminar el trabajo con mayor soltura y entendiendo el funcionamiento general de la Pokedex. En

definitiva, la experiencia fue desafiante al principio pero terminó siendo enriquecedora, ya que permitió adquirir nuevas habilidades y familiarizarse con tecnologias que no se habian usado antes.

Loading Spinner