Universidad Nacional de General Sarmiento

Licenciatura en Sistemas

Trabajos Práctico
Buscador Harry Potter

Introducción a la Programación

(cursada de verano)

Integrantes: González Ezequiel (ezsego18@gmail.com)
Flores Rocio (rochiflores15@gmail.com)

1. Introducción

Este trabajo tiene como objetivo desarrollar una aplicación web que permita buscar imágenes de personajes de Harry Potter. La idea es que el usuario pueda ingresar el nombre de un personaje y obtener imágenes relacionadas que muestren información como la casa, genero, etc.

Para poder realizar este trabajo utilizamos Django, un framework escrito en Python que facilita el desarrollo de aplicaciones web. Dentro del código se trabajaron funciones que filtran la búsqueda por nombre, casa del personaje y un listado de favoritos.

En este informe se detallará el código implementado, explicando el propósito de cada función desarrollada, junto con las dificultades encontradas y las decisiones tomadas durante la implementación.

2. Desarrollo

2.1 Descripción general:

Se desarrolló el buscador de imágenes de Harry Potter pudiendo filtrar por nombre y casa. También se trabajó en el desarrollo de poder iniciar sesión y tener un listado de personajes favoritos, este mismo se puede visualizar por separado y se pueden eliminar personajes de la lista. Para finalizar se agregó un loading spinner cuando se utiliza la búsqueda y se le asignó un tamaño y borde especifico a las “cards”.

2.2 Funcionalidades principales:

Código Implementado

→Función home(request)

```
# esta función obtiene 2 listados: uno de las imágenes de la API y otro de favoritos, ambos en formato Card, y los dibuja en el template 'home.ht
def home(request):
    images= services.getAllImages()
    favourite_list = services.getAllFavourites(request)

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

Propósito: Obtiene dos listados, uno con imágenes de la API y otro con imágenes marcadas como favoritas. Luego, los envía al template home.html para ser mostrados en la interfaz.

→Función getAllImages()

```python
import requests
import random
from ...config import config
from ..transport import transport
from ..persistence import repositories
from ..utilities import translator
from django.contrib.auth import get_user


# función que devuelve un listado de cards. Cada card representa una imagen de la API de HP.
    # debe ejecutar los siguientes pasos:
    # 1) traer un listado de imágenes crudas desde la API (ver transport.py)
    # 2) convertir cada img. en una card.
    # 3) añadirlas a un nuevo listado que, finalmente, se retornará con todas las card encontradas.
    # ATENCIÓN: contemplar que los nombres alternativos, para cada personaje, deben elegirse al azar. Si no existen nombres alternativos, debe mo
def getAllImages():
    json_collection = transport.getAllImages()
    imagenes = []

    for object in json_collection:
        card=translator.fromRequestIntoCard(object)
        if len(card.alternate_names)== 0:
            card.alternate_names="No tiene nombres alternativos"
        else:
            card.alternate_names=random.choice(card.alternate_names)
        imagenes.append(card)

    return imagenes
```

Propósito: Recupera una lista de imágenes desde la API de Harry Potter y las convierte en objetos tipo "card" con información procesada.

→Color del borde de las cards según su casa

```html
<div class="row row-cols-1 row-cols-md-3 g-4">
    {% if images|length == 0 %}
    <h2 class="text-center">La búsqueda no arrojó resultados...</h2>
    {% else %} {% for img in images %}
    <div class="col">
        <!-- evaluar si la imagen pertenece a Gryffindor, Slytherin u otro -->
        {% if img.house == 'Gryffindor' %}
        <div class="card border-success mb-3 ms-5" style="max-width: 540px;">
        {% elif img.house == 'Slytherin' %}
        <div class="card border-danger mb-3 ms-5" style="max-width: 540px;">
        {% elif img.house == 'Hufflepuff' or img.house == 'Ravenclaw' %}
        <div class="card border-warning mb-3 ms-5" style="max-width: 540px;">
        {% endif %}
            <div class="row g-0">
                <div class="col-md-4">
                    <img src="{{ img.image }}" class="card-img-top" alt="imagen">
                </div>
```

Propósito: en el archivo “home.html” revisa según los condicionales que tipo de casa tiene la card para al momento de mostrarla asignarle el correspondiente.

→Función filterByCharacter

```python
# función que filtra según el nombre del personaje.
def filterByCharacter(name):
    filtered_cards = []

    for card in getAllImages():
        if card.name==name:
            filtered_cards.append(card)
        # debe verificar si el name está contenido en el nombre de la card, antes de agregarlo al listado de filtered_cards.

    return filtered_cards
```

Propósito: verifica si el nombre de la card coincide con el nombre ingresado.

→Función filterByHouse

```python
# función que filtra las cards según su casa.
def filterByHouse(house_name):
    filtered_cards = []

    for card in getAllImages():
        if card.house==house_name:
            filtered_cards.append(card)
        # debe verificar si la casa de la card coincide con la recibida por parámetro. Si es así, se añade al listado de filtered_cards.

    return filtered_cards
```

Propósito: verifica si la casa de la card coincide con el de la casa que se haya solicitado para filtrar.

→Llamado a las funciones filterByCharacter y filterByHouse en views.py

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

```python
if house != '':
    images = services.filterByHouse(house) # debe traer un listado filtrado de imágenes, según la casa.
    favourite_list = []
```

Propósito: llamar a la función correspondiente para traer las imágenes filtradas.

→Loading spinner en el botón buscar y para la búsqueda especifica de Gryffindor y Slytherin

```html
<div class="d-flex justify-content-center" style="margin-bottom: 1%">
    <!-- Buscador del sitio -->
    <form class="d-flex" action="{% url 'buscar' %}" method="POST" id="searchForm">
        {% csrf_token %}
        <input class="form-control me-2" type="search" name="query" placeholder="Ginny, Ron, Harry" aria-label="Search">
        <button class="btn btn-outline-success" type="submit" id="searchButton">
            <span id="spinner" class="spinner-border spinner-border-sm d-none" role="status" aria-hidden="true"></span>
            Buscar
        </button>
    </form>
    <script>
        document.getElementById("searchForm").addEventListener("submit", function(event) {
            let btn = document.getElementById("searchButton");
            let spinner = document.getElementById("spinner");

            // Deshabilita el botón y muestra el spinner
            btn.disabled = true;
            spinner.classList.remove("d-none");
        });
    </script>
```

Propósito: mostrar por pantalla una animación de un logo de carga.

→Tamaño para las cards y ancho de su borde.

```html
<!-- evaluar si la imagen pertenece a Gryffindor, Slytherin u otro -->
{% if img.house == 'Gryffindor' %}
<div class="card border-success mb-3 ms-5 h-100" style="max-width: 540px;border-width: 2.5px;">
{% elif img.house == 'Slytherin' %}
<div class="card border-danger mb-3 ms-5 h-100" style="max-width: 540px;border-width: 2.5px;">
{% elif img.house == 'Hufflepuff' or img.house == 'Ravenclaw' %}
<div class="card border-warning mb-3 ms-5 h-100" style="max-width: 540px;border-width: 2.5px;">
{% endif %}
```

→**Logging**

```python
# Estas funciones se usan cuando el usuario está logueado en la aplicación.
@login_required
def getAllFavouritesByUser(request):
    favourite_list = services.getAllFavourites(request)
    return render(request, 'favourites.html', {'favourite_list': favourite_list})


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


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


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

Propósito del código: Permite que un usuario logueado pueda almacenar una o varias imágenes de la galería como favoritos

→**Favoritos**

```python
# # añadir favoritos (usado desde el template 'home.html')
def saveFavourite(request):
    fav = translator.fromTemplateIntoCard(request) # transformamos un request en una Card (ver translator.py)
    fav.user = get_user(request) # le asignamos el usuario correspondiente.

    return repositories.save_favourite(fav) # lo guardamos en la BD.

# usados desde el template 'favourites.html'
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 favourite in favourite_list:
            card = translator.fromRepositoryIntoCard(favourite) # convertimos cada favorito en una Card, y lo almacenamos en el listado de mapped_favourites que luego se retorna.
            mapped_favourites.append(card)

        return mapped_favourites

def deleteFavourite(request):
    favId = request.POST.get('id')
    return repositories.delete_favourite(favId) # borramos un favorito por su ID
```

Propósito del código: El código define tres funciones que guarden y leen datos desde una base de datos para ejecutar "favoritos". Estas funciones están diseñadas para que los usuarios pueden guardar elementos como favoritos, ver todos sus favoritos y eliminar

favoritos.

Una de las primeras decisiones fue aprovechar las funciones ya creadas en transport.py y translator.py en lugar de escribir nuevas. Esto ayudó a mantener el código más ordenado y reutilizable.

Para obtener imágenes de la API, simplemente llamamos a la función que ya existía en transport.py, ya que su propósito era precisamente ese. No tenía sentido reescribir algo que ya funcionaba bien. Para transformar esas imágenes en cards, seguímos la lógica de translator.py sin modificarla. Así, aseguramos que las cards se generaran correctamente sin afectar otras partes del código.

Cuando tuvimos que manejar los nombres alternativos, optamos por random.choice() porque era la opción más simple y clara. Su implementación en una sola línea hizo que el código fuera más fácil de leer y mantener.

Para el trabajo implementado sobre el templae 'home.html' se optó por utilizar un condicional IF para el borde las casas en 3 líneas ya que con un if podíamos cubrir una posible rama y con otros dos elif cubrir las restantes. Sobre el loading spinner podemos decir que fue lo más complicado de implementar debido a que es algo externo al código que ya teníamos, pero mediante documentación de html y bootstrap pudimos implementarlo.

Al momento de desarrollar el filtro por personaje y casa vimos diferentes opciones, pero tras charlarlo con los docentes optamos por la realizada, en la misma encontramos que debíamos utilizar los elementos de las cards para poder compararlos con las funciones implementadas.

Finalmente, en views.py, decidímos llamar directamente a las funciones de services.py para obtener las imágenes y la lista de favoritos. Como ya las había trabajado antes, sabía que funcionaban correctamente y que seguir esta estructura hacía que el código fuera más organizado.

3. Conclusiones:

Este trabajo nos permitió aprender más sobre cómo estructurar una aplicación en Django y cómo aprovechar el código ya existente en lugar de escribir todo desde cero.

Al principio, nos costó entender cómo funcionaban los distintos archivos y cómo interactuaban entre sí, pero con ayuda y práctica logramos comprender mejor su lógica.

También aprendimos a manejar la API correctamente y a solucionar problemas como la selección aleatoria de nombres alternativos, lo cual nos ayudó a mejorar la capacidad para investigar y encontrar soluciones eficientes.

Aprendimos que leer algo lleva a otra cosa y esa otra a otra, y así sucesivamente. Nos dimos cuenta de que la lógica en todo lo que se trabajo fue siempre la misma: leer código, identificar qué hace cada cosa y luego completar el código. Además, entendimos que saber interpretar código ya armado es clave. Nos encantó la experiencia de crear una página web.