Trabajo Práctico
“Buscador de Harry Potter”

Licenciatura en Sistemas
Introducción a la Programación
(Curso de verano 2025)

Resumen:

El trabajo se basa en el desarrollo de una aplicación web que tiene funciones incompletas, pero que está desarrollada casi en su totalidad. En el cual tendremos que utilizar nuestros conocimientos adquiridos para lograr el correcto funcionamiento del mismo.

Integrantes: Gimena Ibars - ibarsgisel075@gmail.com
Leandro Barrios - barrioslg87@gmail.com
Federico Bethge - fedebethge202014@gmail.com

1-Introducción:

Vamos a comenzar buscando en los diferentes archivos, las funciones incompletas que tenemos que desarrollar.

Investigar cuál va a ser el problema específico que tiene que resolver cada una de ellas, y si tienen relación entre sí.

2-Desarrollo:

Para empezar a entender la lógica del funcionamiento del programa y de lo que debería hacer, se consultó el archivo **transport.py**, donde se encontró que la información de todos los personajes de Harry Potter se encontraba subida en un enlace API y era recopilada por una función Json.

```
def getAllImages():
    raw_data = requests.get(config.STUDENTS_REST_API_URL).json()

    json_collection = []
```

Una vez encontrado donde se guardaba la información dentro del código se procedió a utilizar ese dato para implementar los cambios necesarios y dar funcionamiento al programa.

El primer archivo que se procedió a modificar fue el **service.py** donde se llamó a la función que recopilaba los datos para poder hacer uso de ellos y arma la **card_list** que se mostrarían luego en la aplicación. Se armo una lista de los datos que se deberían tomar de la API para poder cumplir con los requisitos de las card a mostrar.

```
# función que devuelve un listado de cards. Cada card representa una imagen de la API de HP.
def getAllImages():
    # 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 mostrar un mensaje adecuado.

    raw_images = transport.getAllImages()
    card_list = []
    for img in raw_images:
            card = {
                "id": img.get("id"),
                "name": img.get("name"),
                "alternate_names": random.choice(img.get("alternate_names", []) or ["Nombre desconocido"]),
                "gender": img.get("gender"),
                "house": img.get("house", "Desconocida"),
                "image": img.get("image"),
                "actor": img.get("actor"),
            }
            card_list.append(card)

    return card_list
```

Además, se implementó la selección del **alternate_names** de forma aleatoria agregando la librería **Random**.

```
import random     #ealiza una seleccion de forma aleatoria
from ..transport import transport
from ..persistence import repositories
from ..utilities import translator
from django.contrib.auth import get_user
```

En este código también se habilito el filtro de búsqueda por palabra, el cual usa los datos ya recopilados por la función anterior (**getAllImages**) para filtrar el nombre ingresado por el usuario, realizando una comparación con el parámetro “name” dentro de la card_list ya obtenida.

```
# función que filtra según el nombre del personaje.
def filterByCharacter(name):
    all_images = getAllImages()  # Obtiene todas las imágenes de transport.py
    filtered_cards = []

    # Filtramos las imágenes que contienen el nombre ingresado, ignorando mayúsculas/minúsculas
    for img in all_images:
        if name.lower() in img['name'].lower():  # Compara el nombre de la imagen con el nombre ingresado
            filtered_cards.append(img)

    return filtered_cards
```

Por ultimo se habilito el filtro que muestra las card por la Casa. En esta parte del código se sigue obteniendo los datos de la función **getAllImages**, en este caso el filtro el se habilita mediante un botón existente en la página de la aplicación. Según la casa que elija, el código hará una comparación con el parámetro “house” mostrando solo los personajes que lo cumplan.

```
# función que filtra las cards según su casa.
def filterByHouse(house_name):
    all_images = getAllImages() #obtiene todas las imagenes de transport.py
    filtered_cards = []

    for card in all_images:
        if card.get('house') == house_name:  # Si la casa de la imagen coincide con la casa proporcionada
            filtered_cards.append(card)  # Agrega la imagen a la lista de filtradas

    return filtered_cards  # Devuelve la lista de imágenes filtradas
```

Se siguió con el código **view.py** donde el témplate **home.html** realiza el pedido y este le responde de manera dinámica mediante la función render presente en Django.

El código llama a la función **service.getAllImages** para obtener la **card_list** de todos los personajes. Reunida por el código anterior.

```python
def home(request):
    images = services.getAllImages()  #Obtiene todas las imágenes del service.py
    favourite_list = []

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

De igual forma se llama a la función **services.filterByCharacter** para la función del buscador por nombre y **services.filterByHouse** para el filtro por casa presentes en el código anterior.

```python
# 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) #Filtra imágenes por personaje desde service.py
        favourite_list = []

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

```python
# función utilizada para filtrar por casa Gryffindor o Slytherin.
def filter_by_house(request):
    house = request.POST.get('house', '')

    if house:  # Si 'house' no está vacío o nulo
            images = services.filterByHouse(house)  # Llama a la funcion filterbyhouse de service.py
            favourite_list = []

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

Y para finalizar el trabajo práctico, implementamos los siguientes cambios en: **home.html**.

Para esta parte del desarrollo fue necesario consultar documentación de Bootstrap sobre cards y cómo generar condicionales en Django. Funcionalidades que hasta el momento no implementamos nunca.

Utilizamos la sintaxis de Django ***{% if %}*** para evaluar la variable ***img.house***, en el cual, si el personaje es de Gryffindor, la ***div.card*** tendrá borde verde, si es de Slytherin será rojo y si pertenece a otra casa o no la tiene definida, el borde será naranja.

```
<!-- 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; border-width: 4px;"> <!-- Muestra borde Verde 4px de ancho -->
{% elif img.house == "Slytherin" %}
<div class="card border-danger mb-3 ms-5" style="max-width: 540px; border-width: 4px"> <!-- Muestra borde Rojo 4px de ancho  -->
{% else %}
<div class="card border-warning mb-3 ms-5" style="max-width: 540px; border-width: 4px"> <!-- Muestra bore Naranja 4px de ancho -->
{% endif %}
```

Implementamos un spinner de carga al botón de búsqueda con Bootstrap, para que al momento de dar clic en **“buscar”** se muestre dicho spinner. Y para la deshabilitación al momento de la interacción del usuario, utilizamos JavaScript. Se tomó la decisión de modificarlo para generar un clima de espera al momento de dar clic al botón **“buscar”**.

```
<!-- 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");
        // Aquí Django procesará la búsqueda. No es necesario un timeout en producción.
    });
</script>
```

3-Conclusiones:

A lo largo del trabajo práctico nos encontramos con varios desafíos, tecnologías que nunca habíamos implementado y situaciones frustrantes al momento de emplearlas.

Este trabajo nos enseñó a tener en cuenta muchos aspectos al momento de escribir código. La forma correcta de organizarse para iniciar el desarrollo e implementarlo de una manera limpia y legible.

Aprendimos tecnologías como Django, Bootstrap, Html, Css, JavaScript, y Git / Github que nos servirán en el futuro para nuestra carrera profesional.