Universidad Nacional de General Sarmiento

Introducción a la Programación

Integrantes:

Lourdes Anabela Boc-Ho (anabelabocho06@gmail.com),

Nicole Terracciano (nicoleterracciano18@gmail.com),

Melanie Rosi (MelanieRosi25@gmail.com)

Profesores: Gonzalo Godoy, Yair Ruiz

Comisión Verano - 2025

En este informe se mostrará y explicará cada parte del código desarrollado para que el buscador web que utiliza una API de Harry Potter funcione correctamente.
Cada parte del código realizado será mostrado bajo una captura de este y una explicación que no sea tan formal. Se busca que el lector pueda entender sin la necesidad de saber tanta programación. Además, se expondrá la lógica detrás de cada código para su fácil interpretación.
Se espera que al final de la lectura, el lector pueda entender o aprender parte del desarrollo de una web que utiliza una API.

- **Views.py**

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

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

Lo que hace esta función es recibir un mensaje, en este caso lo llamaremos,‘request’.
Luego creando una variable llamada ‘images’, que lo que hace es que reciba los valores tomados de la función localizada en un módulo de Django (services.py), getAllImages() que trae toda la información de la base de datos de Harry Potter.
Lo mismo ocurre con la variable ‘favourite_list’, en este caso devuelve el listado vacío.
Finalmente, con el comando ‘return render’ lo que hace el programa es pedirle a Django que renderice el código en un HTML (lenguaje que sirve para estructurar y mostrar contenido en una web, en este caso ‘home.html’ ya que es la página principal, la información que se manda será la de ‘images’ y favourite_list’.

- **Services.py**

```python
# función que devuelve un listado de cards. Cada card representa una imagen de la API de HP.
def getAllImages():
    json_collection = transport.getAllImages()
    images = []
    for item in json_collection:
        card=translator.fromRequestIntoCard(item)
        if len(card.alternate_names)==0:
            card.alternate_names="No tiene nombres alternativos"
        else:
            card.alternate_names=random.choice(card.alternate_names)
        images.append(card)
    return images
```

Lo que hace esta función es, obtener y procesar una lista de imágenes de personajes, en este caso la función getAllImages toma los datos de la API desde el módulo transport.py que llegan en formato JSON(datos crudos; formato de intercambio de datos ligeros y fácil lectura), luego lo convertirá en una Card (un elemento de interfaz que almacena fotos,títulos,descripción;más ordenado y legible)
Se crea una lista vacía llamada Cards para almacenar los personajes procesados.
Recorremos los personajes con la variable json_collection, la cual contiene los datos de la API. Toma cada personaje de la lista y lo convierte en un formato más organizado,usando fromRequestIntoCardTranslator(item).
Importamos la librería Random que sirve para generar valores o números aleatorios. Mediante la condición If, lo que se hizo fue manejar los nombres alternativos, en el caso de que el personaje contenga multiples, se elige una opción al azar , si el personaje no tiene nombres alternativos, mostramos un mensaje indicando que no tiene nombres alternativos.
Finalmente retorna la variable Cards, con la lista de personajes e imágenes.

- **Home.html**

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

En este fragmento de HTML lo que se hizo fue poner bordes a las Cards, según los colores típicos de las casas de los personajes. Nos guiamos con la estructura de los condicionales en Django y con documentación de Bootstrap para las tarjetas y sus bordes.

border-danger (rojo): Card → Gryffindor.

border-success (verde): Card → Slytherin.

border-primary(azul): Card→ Ravenclaw.

border-warning(naranja): Card→ Hufflepuff (y cualquier otra casa que hubiese)

- **Filtrado de personajes por Casa. (Solo Gryffindor o Solo Slytherin)**

En el módulo Services.py, la función FilterByHouse(house_name), se encarga de seleccionar solo las imágenes que pertenecen a una casa en particular.
Creamos una lista vacía llamada filtered_cards = [] para almacenar las imágenes que pertenecen a la casa seleccionada.
Obtenemos completa de imágenes llamando a la función getAllImages().
Se recorre cada imagen con un ‘for´, verificando si la casa del personaje (card.house) es igual a la casa que se busca (house_name).
Se usa, lower() para hacer la comparación sin importar si el usuario escribe "GRYFFINDOR" o "gryffindor".
Si la casa coincide, la imagen se agrega a la lista filtered_cards.

Al final, se devuelve la lista filtrada con todas las imágenes de la casa seleccionada.

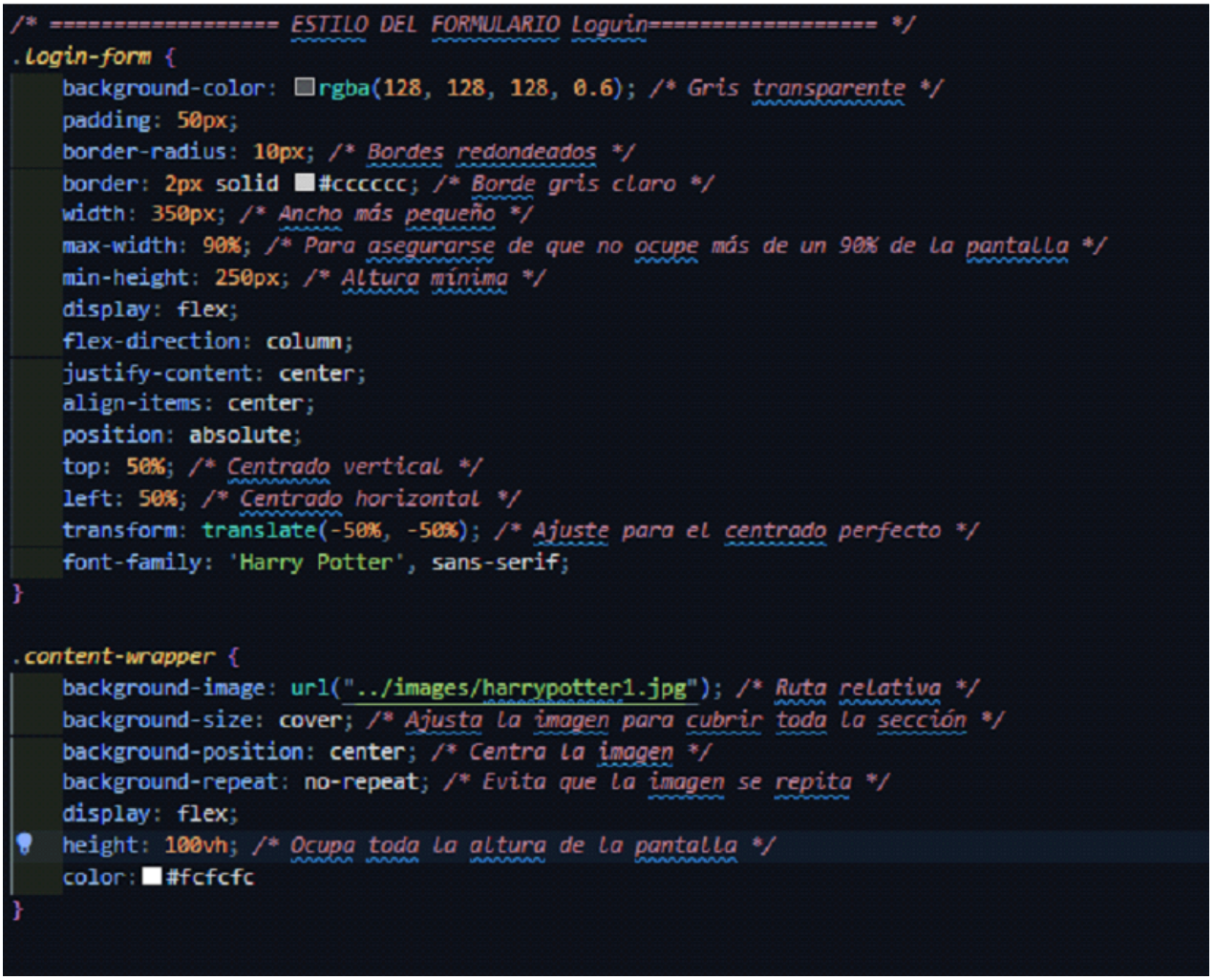
```python
# función que filtra las cards según su casa.
Codeium: Refactor | Explain | Generate Docstring | ×
def filterByHouse(house_name):
    filtered_cards = []
    for card in getAllImages():  # Obtener todas las imágenes
        if card.house and card.house.lower() == house_name.lower():
            # Si la casa del personaje coincide con la seleccionada, se añade a la lista
            filtered_cards.append(card)

    return filtered_cards
```

El módulo views.py maneja las solicitudes del usuario y devuelve las respuestas. En este caso, la función filter_by_house(request) se encarga de recibir el nombre de la casa desde un html llamado home y mostrar solo las imágenes de esa casa en la página.
Recibe la solicitud del usuario, específicamente el valor del campo house desde html home (request.POST.get('house', '')).
Verifica que la casa sea "Gryffindor" o "Slytherin" antes de continuar. Si la casa no es válida, redirige a la página principal (redirect('home')).
 Llama a filterByHouse(house) (desde el módulo services.py) para obtener solo las imágenes de la casa seleccionada.
Crea una lista vacía llamada favourite_list, que podría usarse más adelante para marcar imágenes favoritas.
 Renderiza la página home.html pasando las imágenes filtradas (images) y la lista de favoritos (favourite_list).

```python
# función utilizada para filtrar por casa Gryffindor o Slytherin.
Codeium: Refactor | Explain | Generate Docstring | ×
def filter_by_house(request):
    house = request.POST.get('house', '')

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

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

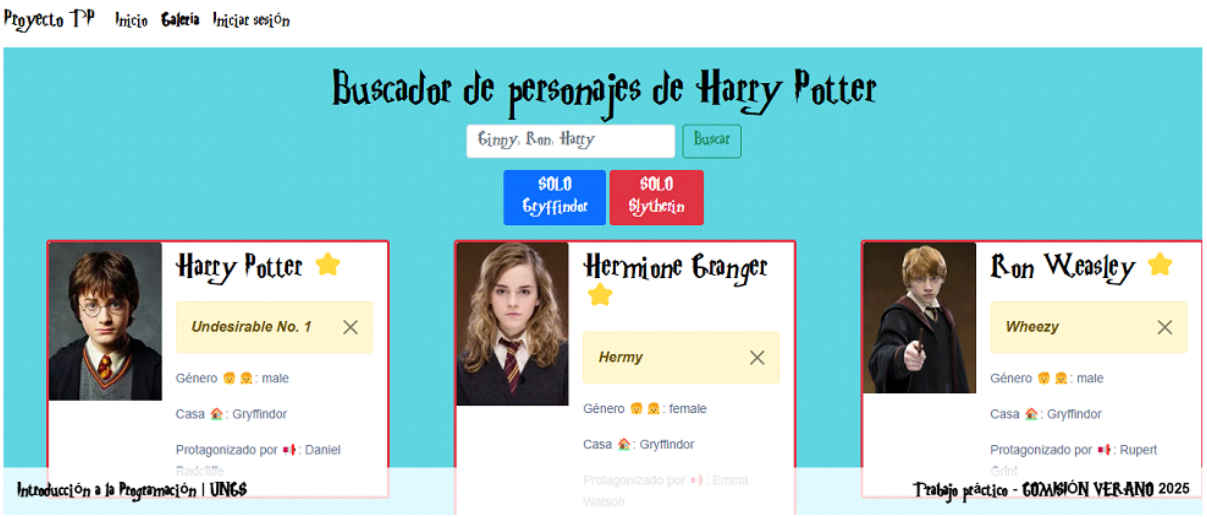
- **Diseño**

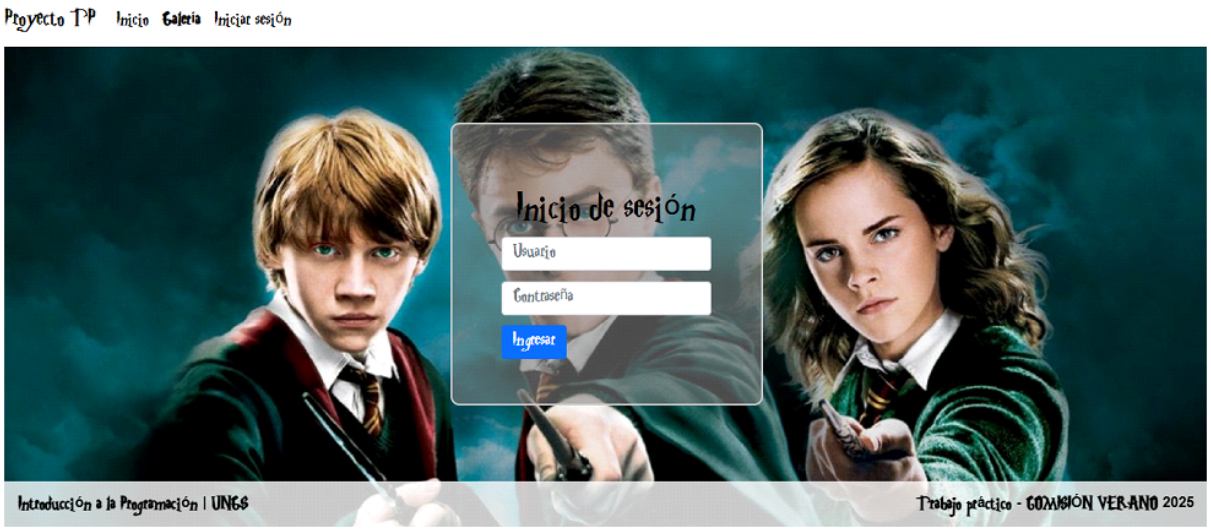
Se modificó todo el diseño de la página por uno propio, en la imagen se muestra una parte de la modificación que se encuentra en #static/styles.css.
Cada sección cuenta con notas que explican qué sección se modifica.
Finalmente la página queda así con los cambios mencionados.



```css
/* ================== ESTILO DEL FORMULARIO Loguin================== */
.Login-form {
    background-color: rgba(128, 128, 128, 0.6); /* Gris transparente */
    padding: 50px;
    border-radius: 10px; /* Bordes redondeados */
    border: 2px solid #cccccc; /* Borde gris claro */
    width: 350px; /* Ancho más pequeño */
    max-width: 90%; /* Para asegurarse de que no ocupe más de un 90% de la pantalla */
    min-height: 250px; /* Altura mínima */
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    position: absolute;
    top: 50%; /* Centrado vertical */
    left: 50%; /* Centrado horizontal */
    transform: translate(-50%, -50%); /* Ajuste para el centrado perfecto */
    font-family: 'Harry Potter', sans-serif;
}

.content-wrapper {
    background-image: url("../images/harrypotter1.jpg"); /* Ruta relativa */
    background-size: cover; /* Ajusta la imagen para cubrir toda la sección */
    background-position: center; /* Centra la imagen */
    background-repeat: no-repeat; /* Evita que la imagen se repita */
    display: flex;
    height: 100vh; /* Ocupa toda la altura de la pantalla */
    color: #fcfcfc
}
```

```css
/* ================== ESTILOS GENERALES ================== */

body {
    color: #566787;
    background-color: #5fd2dd;
    font-family: 'Roboto', sans-serif;
    padding-bottom: 60px;  /* Ajusta el padding para que el contenido no quede tapado por el footer */

}
/* Titulo de las Cards con la fuente de Harry Potter */
.card-title,.navbar-brand{
    font-family: 'Harry Potter', sans-serif;
    color: black
}
/* Footer, Inicio y Home con la fuente de Harry Potter */
.text-center{
    font-family: 'Harry Potter', sans-serif;
    padding-top: 30px; /* Agrega espacio interno arriba */
    text-align: center;
    color:black

}


/*  Botones con la fuente de Harry Potter */
.d-flex{
    font-family: 'Harry Potter', sans-serif;
}

/*Fuente del header  */
.navbar {
    background-color: #fff;
    font-family: 'Harry Potter', sans-serif;
}
.navbar .nav-link ,.nav-item{
    color: black !important; /* Texto negro para contraste */
    font-family: 'Harry Potter', sans-serif;
}
```





Conclusión:

Este trabajo consistió en terminar de codear una aplicación web usando Django que permita buscar imágenes de los personajes de Harry Potter, usando una API.

Esta información proveniente de la API, se verá reflejada en Cards que muestran la imagen del personaje, sus apodos de forma aleatoria (en el caso de que tengan) y casas a las que pertenecen.

Logramos implementar un filtrado por casas de los personajes y colocarles un borde de color acorde a la que pertenian.

Nos sirvió para desarrollar un ambiente de trabajo respecto a un repositorio remoto, donde los integrantes vivimos y tenemos horarios distintos en los cuales trabajar/estudiar, por lo cual aprender el uso de

GitHub para tener un ambiente compartido en el cual ir subiendo avances logrados fue necesario para finalizar el trabajo.
Fue una buena experiencia para aprender programación un poco más avanzada.