TP - Buscador Rick & Morty

Ivan Rossini

Este proyecto consiste en crear una aplicación web con django con la que podemos buscar imágenes de los personajes de la serie de **Rick y Morty** a través de su api. Esta información obtenida muestra en formato de tarjetas la imagen, el estado (vivo, muerto, desconocido), su ultima ubicación y su primer episodio. Ademas tiene un sistema de búsqueda de personajes, y a futuro se planea implementar autenticación de usuarios para guardar personajes como favoritos, entre otras cosas.

Views.py

- Home

```python
def home(request):
    images = [] # lista de imagenes de la api

    images = transport.getAllImages() # obtiene las imagenes desde transport

    favourite_list = [] #lista favoeritos

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

En esta funcion se obtienen las imágenes de los personajes, para luego pasarlas a la web y mostrarlos en cada tarjeta.

- Search

```python
def search(request):
    search_msg = request.POST.get('query', '') #agarra el texto ingresado. si no hay nada, devuelve una cadena vacia

    # si el texto ingresado no es vacío, trae las imágenes y favoritos desde services.py, y luego renderiza el template (similar a home).
    if (search_msg != ''):
        images = services.getAllImages(search_msg) # obtiene las imagenes de la api

        return render(request, 'home.html', {'images': images})
    else:
        return redirect('home') # si esta vacio redirige al home
```

En esta función se maneja la búsqueda de imagenes por medio de un buscador. Se obtienen las imágenes que coincidan con el texto ingresado y las muestra. Si no se ingresa nada, se redirige automáticamente a la pagina de inicio.

Services.py

- getAllImages

```python
def getAllImages(input=None):
    # obtiene un listado de datos "crudos" desde la API, usando a transport.py.
    json_collection = []

    # recorre cada dato crudo de la colección anterior, lo convierte en una Card y lo agrega a images.
    images = []

    json_collection = transport.getAllImages(input) #obtiene los datos de la funcion getallimages
    for character in json_collection: # itera con cada personaje
        images.append({ # creo un objeto con los datos del personaje
            "name": character["name"],
            "image": character["image"],
            "status": character["status"],
            "location": character["origin"]["name"],
            "origin": character["origin"]["name"],
        })

    return images
```

Creé esta función porque no pude lograr que el codigo original me devolviera los datos de la api. Luego de la correccion del profe Franco en clase, intente volver a hacerlo pero utilizando el codigo original y no me funcionaba, por lo que volví a lo que yo habia hecho.

Home.html

1-

```
<div class="card mb-3 ms-5
    {% if img.status == 'Alive' %}border-success
    {% elif img.status == 'Dead' %}border-danger
    {% else %}border-warning{% endif %}"
        style="max-width: 540px; border-width: 2px;">
```

En este div modifique la condición para cada estado de los personajes, ya que no me funcionaba con los nombres originales. Si el estado del personaje es “alive” entonces pinta el borde de color verde. En cambio si el estado es “dead” lo hace de color rojo. Y por ultimo si no es ninguno de los anteriores, lo hace de color naranja.

2-

```
<div class="col-md-4">
    <img src="{{ img.image }}" class="card-img-top" alt="imagen">
</div>
```

En este otro div, modifique el nombre de dónde se busca la imagen, ya que en la plantilla original era “img.url” y eso no coincide con como esta definida en la api. Luego de hacer este cambio pude cargar cada foto a las tarjetas.

3-

```
<div class="card-body">
    <h3 class="card-title">{{ img.name }}</h3>
    <p class="card-text">
        <strong>
            {% if img.status == 'Alive' %} 🟢 {{ img.status }}
            {% elif img.status == 'Dead' %} 🔴 {{ img.status }}
            {% else %} 🟠 {{ img.status }}
            {% endif %}
        </strong>
    </p>
    <p class="card-text"><small class="text-body-secondary">Episodio inicial: {{ img.origin.name }}</small></p>
    <p class="card-text"><small class="text-body-secondary">Última ubicación: {{ img.location.name }}</small></p>
</div>
```

En este caso, modifique las condiciones de el “status” de cada tarjeta, ya que nuevamente no coincidían los nombres originales de la plantilla con los nombres de la api. Originalmente estaba como “if true == …”, asi que lo puse “if img.status == …”. Además de modificar también los nombres de el episodio incial y ultima ubicación, los cuales no coincidían con la api, pasaron de “last_location” y “first_seen” a “img.origin.name” y “img.location.name” respectivamente.

4-

```
<div id="spinner" class="spinner-border text-primary" role="status"
    style="position: fixed; top: 50%; left: 50%; display: none;"></div>
```

```
<script>
    // muestra el spinner mientras carga
    document.getElementById('spinner').style.display = 'block';

    // oculta el spinner luego
    window.onload = function() {
      document.getElementById('spinner').style.display = 'none';
    };
</script>
```

Por ultimo agregue este spinner de carga, utilizando bootstrap y javascript, con ayuda de tutoriales y la documentacion oficial de botstrap.

Conclusion

En conclusion, este trabajo me sirvio para recordar algunos conceptos que habia olvidado de la secundaria, ya que estudie Tecnicatura en Informatica en el secundario, y ademas, aunque no lo haya terminado e implementar los cambios visuales, fue algo divertido ya que a mi me gusta dedicarme mas al front que al backend.