Informe del Trabajo Práctico

Aplicación Web de Galería de Imágenes de "Rick & Morty"

Álvaro Adriano Ruiz Kuschill

Tomás Carrizo

Franco Garré

Universidad Nacional de General Sarmiento

Comisión 12

Nora Martinez

Flavia Bottino

2do semestre 2024

Introducción

Este trabajo práctico consiste en desarrollar una aplicación web que permite visualizar imágenes de los personajes de la serie "Rick & Morty" a través de su API oficial. La aplicación está construida usando el framework Django, y permite al usuario interactuar con una galería de imágenes, ver detalles sobre los personajes (como su nombre, estado, última ubicación y episodio inicial).
La aplicación permite a los usuarios buscar personajes específicos, registrarse como nuevo usuario y posteriormente iniciar sesión.

Parte obligatoria

1) En services.py se importó la función getAllImages() que viene desde transport.py para obtener todos los objetos en el json_collection y luego se recorre el json_collection para obtener los datos deseados y agregarlos a images.

```python
from ...layers.transport.transport import getAllImages as transport_getAllImages

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

    # recorre cada dato crudo de la colección anterior, lo convierte en una Card y lo agrega a images.
    images = []
    for dato in json_collection:
        images.append({
            "url": dato.get("image"),
            "name" : dato.get("name"),
            "status": dato.get("status"),
            "first_seen": dato.get("origin", {}).get("name"),
            "last_location" : dato.get("location", {}).get("name"),
        })

    return images
```

Dificultades

- Recorrer el json_collection y agregar lo necesario a images para retornarlo

Decisiones

- Saber que datos necesitamos para pedirlos y agregarlos a images

2) En el archivo views.py se importó services.py y se llamó a la función getAllImages() para usarlas en la variable images y luego enviarla al home.html.

```python
def index_page(request):
    return render(request, 'index.html')

# esta función obtiene 2 listados que corresponden a las imágenes de la API y los favoritos del usuario, y los usa para dibujar el correspondiente template.
# si el opcional de favoritos no está desarrollado, devuelve un listado vacío.
def home(request):
    images = services.getAllImages()
    favourite_list = []

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

Dificultades

- Analizar el flujo del código y entenderlo

Decisiones

- Llamar a la función desde services.py y luego enviarla al home.html

3) En el home se usaron condicionales para que dependiendo el estado del personaje, tanto el borde como la bolita cambien de color.

```
{% if images|length == 0 %}
<h2 class="text-center">La búsqueda no arrojó resultados...</h2>
{% else %} {% for img in images %}
<div class="col">
    <div class="card mb-3 ms-5" style="max-width: 540px; border: 2px solid
        {% if img.status == 'Alive' %}#00D26A
        {% elif img.status == 'Dead' %}#F8312F
        {% else %}#FF6723
        {% endif %};">
        <div class="row g-0">
            <div class="col-md-4">
                <img src="{{ img.url }}" class="card-img-top" alt="imagen">
            </div>

            <div class="col-md-8">
                <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">Última ubicación: {{ img.last_location }}</small></p>
                    <p class="card-text"><small class="text-body-secondary">Episodio inicial: {{ img.first_seen }}</small></p>
                </div>
```

Dificultades

- Entender que al principio se estaba igualando un booleano a un string cosa que no es posible y con este cambio ya es posible realizar el if

Decisiones

- Poner el código hexadecimal del color de las bolitas en el borde de la card

EXTRAS

1) **BUSCADOR**

en el views.py con la función search() ya creada, lo que se hizo es darle como parámetro a getAllImages lo que el usuario buscó que se encuentra en la variable search_msg y si existe ese personaje, lo muestra en images, sino te devuelve al home.

```
def search(request):
    search_msg = request.POST.get('query', '')

    # 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(input=search_msg)
    else:
        return redirect('home')
    return render(request, 'home.html', {'images':images})
```

Dificultades

- Entender lo que era la variable search_msg y luego usarla como propiedad

Decisiones

- Usar la variable search_msg en la función getAllImages traida desde services.py

2) INICIO DE SESIÓN

En el archivo settings.py se cambió los redirects una vez se loguea o desloguea el usuario.

```
# VARIABLES QUE INTEGRAN LOS REDIRECTS DE AUTH
LOGIN_REDIRECT_URL = '/'
LOGOUT_REDIRECT_URL = '/accounts/login/'
```

En el header.html se cambió el nombre del url que era ‘exit’ por ‘logout’ y se le puso ‘Cerrar sesión’ en vez de ‘Salir’

```
<div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
            <a class="nav-link active" aria-current="page" href="{% url 'index-page' %}">Inicio</a>
        </li>
        <li class="nav-item">
            <a class="nav-link active" aria-current="page" href="{% url 'home' %}"><strong>Galería</strong></a>
        </li>
        {% if request.user.is_authenticated %}
        <li class="nav-item">
            <a class="nav-link" href="{% url 'favoritos' %}">Favoritos</a>
        </li>
        {% endif %}
        <li class="nav-item d-flex align-center">
            {% if request.user.is_authenticated %}
            <a class="nav-link" href="{% url 'logout' %}">Cerrar sesión</a> {% else %}
            <a class="nav-link d-inline" href="{% url 'login' %}">Iniciar sesión</a>{% endif %}
        </li>
    </ul>

</div>
```

En el urls.py se cambio el enlace a ‘logout/’

```
urlpatterns = [
    path('', views.index_page, name='index-page'),
    path('login/', views.index_page, name='login'),
    path('home/', views.home, name='home'),
    path('buscar/', views.search, name='buscar'),

    path('favourites/', views.getAllFavouritesByUser, name='favoritos'),
    path('favourites/add/', views.saveFavourite, name='agregar-favorito'),
    path('favourites/delete/', views.deleteFavourite, name='borrar-favorito'),

    path('logout/', views.logout, name='logout'),
```

En el views.py se cambió el nombre de la función por ‘logout’

```
@login_required
def logout(request):
    pass
```

Dificultades

- Cambiar los nombres de ‘exit’ por ‘logout’

Decisiones

- Guiarnos por el video recomendado en el repositorio

3) **Registro de usuarios**

En el signup.html se creó el formulario con los datos requeridos para el registro de nuevos usuarios

```
{% extends 'header.html' %} {% block content %}
<div class="signup-form" style="text-align: center;">
    <form method="POST" style="display: inline-block;">
        {% csrf_token %}
        <h2 class="text-center">Registrarse</h2>
        <div class="form-group" style="margin-bottom: 5%;">
            <input type="text" name="name" id="name" class="form-control" placeholder="Nombre" required="required">
        </div>
        <div class="form-group" style="margin-bottom: 5%;">
            <input type="text" name="surname" id="surname" class="form-control" placeholder="Apellido" required="required">
        </div>
        <div class="form-group" style="margin-bottom: 5%;">
            <input type="text" name="username" id="username" class="form-control" placeholder="Usuario" required="required">
        </div>
        <div class="form-group" style="margin-bottom: 5%;">
            <input type="email" name="email" id="email" class="form-control" placeholder="Email" required="required">
        </div>
        <div class="form-group" style="margin-bottom: 5%;">
            <input type="password" name="password1" id="password1" class="form-control" placeholder="Contraseña" required="required">
        </div>
        <div class="form-group" style="margin-bottom: 5%;">
            <input type="password" name="password2" id="password2" class="form-control" placeholder="Verificar contraseña" required="required">
        </div>
        <div class="form-group">
            <button type="submit" class="btn btn-primary btn-block">Ingresar</button>
        </div>
    </form>
</div>
{% endblock %}
```

En el urls.py se agregó la url para que el usuario pueda ver el formulario de registro

```
#RESGISTRAR
path('accounts/signup/', views.signup, name='signup')
```

En el views.py se desarrolló la función para registrar nuevos usuarios y guardarlos en la base de datos, además se agregó la funcionalidad para enviar email de registro exitoso al usuario.

```python
#REGISTRO DE NUEVOS USUARIOS
def signup(request):
    if request.method == 'GET':
        return render(request, 'signup.html', {'form': UserCreationForm})
    else:
        #pregunta si las contraseñas coinciden
        if request.POST['password1'] == request.POST['password2']:
            #pregunta si el usuario existe (devuelve True o False)
            if existeUser(request.POST['username']):
                return HttpResponse("el usuario ya existe")
            else:
                # se registra el usuario
                user = User.objects.create_user(
                    username=request.POST['username'],
                    email=request.POST['email'],
                    password=request.POST['password1'],
                    )
                user.first_name=request.POST['name'],
                user.last_name=request.POST['surname'],
                user.save()
                # se envia el email
                email = request.POST['email']
                asunto = 'Registro Exitoso'
                mensaje = f'Hola {request.POST["name"]},\n\nTe registrase exitosamente en nuestro sitio.'
                send_mail(
                    asunto,
                    mensaje,
                    settings.EMAIL_HOST_USER,
                    [email],
                    fail_silently=False
                )
                messages.success(request, 'Correo de confirmación enviado correctamente.')
                return redirect('home')
        else:
            return HttpResponse("no coinciden las contraseñas")

def existeUser(username):
    return User.objects.filter(username=username).exists()
```

En el settings.py se agregaron los datos necesarios para que se pueda enviar el email al usuario que se registró

```python
#REGISTRO DE NUEVOS USUARIOS Y ENVIAR EMAIL
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = 'ipalvaro06rk@gmail.com'
EMAIL_HOST_PASSWORD = 'clnhqnvblltpclba'
```

Dificultades

- Al crear el usuario la clase User solo espera username, email y password, nosotros necesitábamos agregarle el first_name y el last_name
- No se enviaba el email porque no habíamos puesto el email del remitente

Decisiones

- Nos guiamos con el video del repositorio para poder enviar los mails
- Nos guiamos de tutoriales para poder guardar el usuario en la base de datos