INFORME DE TRABAJO PRÁCTICO, INTRODUCCIÓN A LA PROGRAMACIÓN

¿De qué consiste el Trabajo?

El trabajo consiste en el desarrollo de una página web utilizando Django, la cual nos ayuda a crear mediante funciones qué nosotros implementemos, una página de la serie "*Rick & Morty*". La aplicación muestra información sobre los personajes, como su imagen, estado, última ubicación y episodio en el que aparecieron por primera vez. Además, se implementó un módulo de autenticación de usuarios, donde los usuarios pueden almacenar como favoritos aquellos personajes que deseen. Estos favoritos pueden ser consultados posteriormente al iniciar sesión en la aplicación.

Codigos usados:

views.py

Función: Obtiene dos conjuntos de datos: uno con las imágenes extraídas de la API y otro con los personajes favoritos del usuario, y los emplea para generar el template adecuado

Cambios realizados:

Agregada lógica para manejar la paginación.

Devuelve una lista de páginas para iterar en la plantilla.

services.py

Función: Recibe una lista de imágenes en formato de Card provenientes de la API. Si se incluye un parámetro de entrada, este especifica los personajes o imágenes que deben ser filtrados

Cambios realizados:

Manejo de la lógica de paginación y obtención de imágenes de la página solicitada.

```
def getAllImages(page=1, query=''):

    json_collection, total_pages = repositories.getAllImages(page, query)

    images = [translator.fromRequestIntoCard(obj) for obj in json_collection]

    return images, total_pages
```

repositories.py

Función: gestiona la obtención de datos de la API de Rick and Morty y la manipulación de objetos Favourite en la base de datos

Cambios realizados:

Petición a la API de Rick & Morty para obtener los resultados de búsqueda con paginación.

```
import requests

def getAllImages(page=1, query=''):
    url = f"https://rickandmortyapi.com/api/character/?page={page}&name={query}"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        results = data.get('results', [])
        total_pages = data['info']['pages']
        return results, total_pages
    else:
        return [ ], 0
```

home.html

Cambios realizados:

Inclusión del paginador en la plantilla.

```
<div class="d-flex justify-content-center mt-4">
    <nav aria-label="Page navigation">
        <ul class="pagination">
            {% for i in pages %}
            <li class="page-item {% if i == current_page %}active{% endif %}">
                <a class="page-link" href="?query={{ query }}&page={{ i }}">{{ i }}</a>
            </li>
            {% endfor %}
        </ul>
    </nav>
</div>
```

2. *Ponerles color a los bordes según su estado*

home.html

Cambios realizados:

Modificación del estilo de la tarjeta para incluir un borde de color basado en el estado del personaje.

```
<div class="col-md-4 mb-3">
    <div class="card h-100" style="border: 2px solid {% if img.status == 'Alive' %} green {% elif
<img.status == 'Dead' %} red {% else %} orange {% endif %};">
        <img src="{{ img.url }}" class="card-img-top" alt="{{ img.name }}">
        <div class="card-body">
            <h5 class="card-title">{{ img.name }}</h5>
            <p class="card-text">
                <strong>
                    {% if img.status == 'Alive' %} 🟢 {{ img.status }}
                    {% elif img.status == 'Dead' %} 🔴 {{ img.status }}
                    {% else %} 🟠 {{ img.status }}
                    {% endif %}
                </strong>
            </p>
```

COMPLICACIONES A LA HORA DE IMPLEMENTAR LAS FUNCIONES:

se clonaba un círculo extra del color del estado, se clonaba una palabra del estado pero con el color del estado correspondiente

3. Buscador

home.html

Cambios realizados:

Inclusión del formulario de búsqueda en la parte superior de la galería.

```
<div class="d-flex justify-content-center mb-3">
    <form class="form-inline" action="{% url 'buscar' %}" method="POST">
        {% csrf_token %}
        <input class="form-control mr-2" type="search" name="query" placeholder="Escribí una
palabra" aria-label="Search" value="{{ query }}">
        <button class="btn btn-outline-success" type="submit">Buscar</button>
    </form>
</div>
```

COMPLICACIONES A LA HORA DE IMPLEMENTAR LAS FUNCIONES:

la página no dejaba utilizar el buscador y a la hora de buscar algo aparecía un error en pantalla.

views.py

Cambios realizados:

Adición de una vista para manejar la búsqueda y redirigir a la página de resultados.

```
def search(request):
    query = request.POST.get('query', '')
    return redirect(f'/home?query={query}&page=1')
```

4. Añadir comentarios a favoritos

favourites.html

Cambios realizados:

Inclusión de un formulario para añadir y editar comentarios en los favoritos.

```
  form method="post" action="{% url 'editar-comentario' %}">
    {% csrf_token %}
    <input type="hidden" name="id" value="{{ favourite.id }}">
    <input type="text" name="comment" value="{{ favourite.comment }}" placeholder="Añadir un
comentario">
    <button type="submit" class="btn btn-primary btn-sm">💬 Guardar comentario</button>
</form>
{% if favourite.comment %
  div class="mt-2"
    <p>{{ favourite.comment }}</p>
    <form method="post" action="{% url 'borrar-comentario' %}">
        {% csrf_token %}
        <input type="hidden" name="id" value="{{ favourite.id }}">
        <button type="submit" class="btn btn-danger btn-sm">Eliminar comentario< button>
    </form>
</div>
{% endif %}
```

views.py

Cambios realizados:

Adición de vistas para manejar la edición y eliminación de comentarios.

```
@login_required
def editComment(request):
    fav_id = request.POST.get('id')
    comment = request.POST.get('comment')
    services.editComment(fav_id, comment)
    return redirect('favoritos')

@login_required
def deleteComment(request):
    fav_id = request.POST.get('id')
    services.deleteComment(fav_id)
    return redirect('favoritos')
```

repositories.py

Cambios realizados:

Adición de funciones para actualizar y eliminar comentarios en la base de datos.

```
def editComment(fav_id, comment):
    try:
        favourite = Favourite.objects.get(id=fav_id)
        favourite.comment = comment
        favourite.save()
        return True
    except Favourite.DoesNotExist:
        print(f"El favorito con ID {fav_id} no existe.")
        return False
    except Exception as e:
        print(f"Error al actualizar el comentario: {e}")
        return False

def deleteComment(fav_id):
    try:
        favourite = Favourite.objects.get(id=fav_id)
        favourite.comment = ""
        favourite.save()
        return True
    except Favourite.DoesNotExist:
        print(f"El favorito con ID {fav_id} no existe.")
        return False
    except Exception as e:
        print(f"Error al eliminar el comentario: {e}")
        return False
```

5. Añadir a favoritos

home.html

Cambios realizados:

Inclusión de un formulario para añadir personajes a la lista de favoritos.

```
{% if request.user.is_authenticated %}
<div class="card-footer">
    <form method="post" action="{% url 'agregar-favorito' %}">
        {% csrf_token %}
        <input type="hidden" name="name" value="{{ img.name }}">
        <input type="hidden" name="url" value="{{ img.url }}">
        <input type="hidden" name="status" value="{{ img.status }}">
        <input type="hidden" name="last_location" value="{{ img.last_location }}">
        <input type="hidden" name="first_seen" value="{{ img.first_seen }}">
        {% if img in favourite_list %}
        <button type="submit" class="btn btn-primary btn-block" disabled>✔️ Ya está en
favoritos</button>
        {% else %}
        <button type="submit" class="btn btn-primary btn-block">❤️ Añadir a favoritos</button>
        {% endif %}
    </form>
</div>
{% endif %}
```

COMPLICACIONES A LA HORA DE IMPLEMENTAR LAS FUNCIONES:

La página no indicaba si se había añadido un personajes en favoritos, tampoco dejaba eliminar a los personajes ya incluidos a la lista de favoritos.

6. Formulario de inicio de sesión

Cambios realizados:

Inclusión de un formulario de inicio de sesión utilizando Bootstrap.

Inclusión de mensajes de error.

```
{% extends 'header.html' %} {% block content %}
<div class="container mt-5">
    <div class="row justify-content-center">
        <div class="col-md-6">
            <div class="card">
                <div class="card-header bg-dark text-white">
                    <h3 class="text-center">Inicio de sesión</h3>
                </div>
                <div class="card-body">
                    <form action="{% url 'login' %}" method="POST">
                        {% csrf_token %}
                        <div class="form-group">
                            <label for="username">Usuario</label>
                            <input type="text" name="username" id="username" class="form-control"
placeholder="Usuario" required>
                        </div>
                        <div class="form-group">
                            <label for="password">Contraseña</label>
                            <input type="password" name="password" id="password" class="form-control"
placeholder="Contraseña" required>
                        </div>
                        <button type="submit" class="btn btn-dark btn-block mt-3">Ingresar</button>
                        {% if messages %}
                        <div class="alert alert-danger mt-3" role="alert">
                            {% for message in messages %}
                                <p>{{ message }}</p>
                            {% endfor %}
                        </div>
                        {% endif %}
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

COMPLICACIONES A LA HORA DE IMPLEMENTAR LAS FUNCIONES:

se producía un error aunque se ponga el admin, la pagina no dejaba ingresar con la cuenta de admin, la pagina se congelaba a la hora de inciar sesion.

7. Renovación de la interfaz gráfica utilizando Bootstrap

Objetivo: Modernizar la interfaz gráfica utilizando Bootstrap.

header.html

Cambios realizados:

Inclusión de los estilos de Bootstrap:

```
<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
```

Inclusión de los estilos personalizados (styles.css):

```
<link rel="stylesheet" type="text/css" href="{% static 'styles.css' %}">
```

Inclusión de los iconos de Material Icons:

```
<link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
```

Inclusión de los scripts de Bootstrap y dependencias:

```
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
```

Home.html

Cambios realizados:

Uso de clases de Bootstrap para el diseño y estilo:

```
<main class="container mt-4">

    <h1 class="text-center mb-4">Buscador Rick & Morty</h1>
```

Integración de un formulario de búsqueda con diseño responsive:

```
<div class="d-flex justify-content-center mb-3">
    <form class="form-inline" action="{% url 'buscar' %}" method="POST">
        {% csrf_token %}
        <input class="form-control mr-2" type="search" name="query" placeholder="Escribí una
palabra" aria-label="Search" value="{{ query }}">
        <button class="btn btn-outline-success" type="submit">Buscar</button>
    </form>
</div>
```

Paginación utilizando Bootstrap:

```
<div class="d-flex justify-content-center mt-4">
    <nav aria-label="Page navigation">
        <ul class="pagination">
            {% for i in pages %}
            <li class="page-item {% if i == current_page %}active{% endif %}">
                <a class="page-link" href="?query={{ query }}&page={{ i }}">{{ i }}</a>
            </li>
            {% endfor %}
        </ul>
    </nav>
</div>
```

COMPLICACIONES A LA HORA DE IMPLEMENTAR LAS FUNCIONES:

-la página presentaba un mensaje de error, y la interfaz no se modificaba

8. Dar de alta a nuevos usuarios

Paso 1: Crear el formulario de registro

Código añadido:

```
python
from django import forms
from django.contrib.auth.models import User
from django.core.exceptions import ValidationError

class UserRegisterForm(forms.ModelForm):
    password = forms.CharField(widget=forms.PasswordInput)
    confirm_password = forms.CharField(widget=forms.PasswordInput)

    class Meta:
        model = User
        fields = ['first_name', 'last_name', 'username', 'email', 'password']

    def clean(self):
        cleaned_data = super().clean()
        password = cleaned_data.get('password')
        confirm_password = cleaned_data.get('confirm_password')

        if password != confirm_password:
            raise ValidationError("Las contraseñas no coinciden.")

        return cleaned_data

    def clean_username(self):
        username = self.cleaned_data.get('username')
        if User.objects.filter(username=username).exists():
            raise ValidationError("El nombre de usuario ya está en uso.")
        return username

    def clean_email(self):
        email = self.cleaned_data.get('email')
        if User.objects.filter(email=email).exists():
            raise ValidationError("El correo electrónico ya está anclado a otro usuario.")
        return email
```

Explicación:

Formulario: Se creó un formulario usando Django Forms para capturar los datos del usuario: nombre, apellido, nombre de usuario, correo electrónico, contraseña y confirmación de contraseña.
Validaciones: Se incluyeron validaciones para asegurar que las contraseñas coincidan, que el nombre de usuario no esté en uso, y que el correo electrónico no esté registrado con otro usuario.

Paso 2: Modificar la vista de registro

Código modificado:

```
python

from django.utils.encoding import smart_str

def register(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST)
        if form.is_valid():
            user = form.save(commit=False)
            user.set_password(form.cleaned_data['password'])
            user.save()

            # Enviar correo electrónico con codificación UTF-8
            subject = smart_str('Registro exitoso', encoding='utf-8')
            message = smart_str(f'Tus credenciales de acceso son:\n\nUsuario:
{user.username}\nContraseña: {form.cleaned_data["password"]}', encoding='utf-8')

            send_mail(
                subject,
                message,
                'tu_correo@gmail.com',
                [user.email],
                fail_silently=False,
            )

            messages.success(request, 'Tu cuenta ha sido creada exitosamente. Revisa tu correo
electrónico para más detalles.')
            return render(request, 'registration/register.html', {'form': form, 'show_message': True})
    else:
        form = UserRegisterForm()
    return render(request, 'registration/register.html', {'form': form})
```

Explicación:

Importación de smart_str: Se importó smart_str de django.utils.encoding para manejar la codificación UTF-8.
Uso de smart_str: Se usó smart_str para codificar tanto el asunto (subject) como el mensaje (message) del correo electrónico en UTF-8.
Envío de correo: Se envió un correo electrónico con el asunto y mensaje codificados correctamente en UTF-8 para evitar errores de codificación.

Paso 3: Crear la plantilla para el formulario de registro

Código añadido:

```
html
{% extends 'header.html' %}
{% block content %}
<div class="container mt-5">
    <div class="row justify-content-center">
        <div class="col-md-6">
            <div class="card">
                <div class="card-header bg-dark text-white">
                    <h3 class="text-center">Registro de usuario</h3>
                </div>
                <div class="card-body">
                    {% if show_message %}
                    <div class="alert alert-success text-center" role="alert">
                        ¡Usuario creado con éxito! Serás redirigido a la página de inicio de sesión en breve.
                    </div>
                    <script>
                        setTimeout(function() {
                            window.location.href = "{% url 'login' %}";
                        }, 2000); // 2000 ms = 2 segundos
                    </script>
                    {% else %}
                    <form method="post">
                        {% csrf_token %}
                        {{ form.as_p }}
                        <button type="submit" class="btn btn-dark btn-block mt-3">Registrarse</button>
                    </form>
                    {% endif %}
                </div>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

Explicación:

Plantilla HTML: Se creó una plantilla HTML para el formulario de registro.
Mensaje de éxito: Si el registro es exitoso, se muestra un mensaje de éxito y se redirige al usuario a la página de inicio de sesión después de dos segundos usando JavaScript.

Paso 4: Configuración del envío de correos electrónicos en settings.py

Código añadido:

```
python
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = 'tu_correo@gmail.com'
EMAIL_HOST_PASSWORD = 'tu_contraseña_de_correo'
```

Explicación:

Configuración de SMTP: Se configuró el envío de correos electrónicos usando la cuenta de Gmail y el backend SMTP de Django.

Paso 5: Agregar la URL para la vista de registro

Código añadido:

```
python
from django.urls import path
from . import views

urlpatterns = [
    path('register/', views.register, name='register'),
    path('login/', views.login_user, name='login'),
    path('', views.index_page, name='index-page'),
    # otras URLs...
]
```

Explicación:

Ruta de URL: Se añadió la ruta URL para la vista de registro, permitiendo que los usuarios accedan al formulario de registro a través de /register/.

COMPLICACIONES A LA HORA DE IMPLEMENTAR LAS FUNCIONES:

una vez que se creaba un usuario aparecia un error, sin embargo la cuenta era creada exitosamente, y otro inconveniente era que al principio se podían crear varios usuarios utilizando el mismo correo electrónico