TRABAJO PRACTICO I.P.

INTEGRANTES: AGUSTIN EZEQUIEL PAGLINI Y JULIAN EMANUEL PAGLINI

PROFESORES: DANIEL BRESSKY, ESTEBAN FASSIO Y SANTIAGO VELAZQUEZ

COMISION: 03

INTRODUCCION

Este trabajo práctico tiene como objetivo desarrollar una aplicación web que permita a los usuarios explorar información sobre personajes de la serie *Rick & Morty*. La idea principal es ofrecer una herramienta donde las personas puedan buscar imágenes y datos de los personajes de una manera fácil.

La aplicación toma los datos directamente desde una fuente en línea, asegurando que la información sea actual y completa. Los resultados de las búsquedas se presentan en una página web diseñada para que sean claros y accesibles para cualquier usuario.

PROCEDIMIENTO

La API se encuentra en https://rickandmortyapi.com/.
Usaremos la ruta /character/ para obtener información sobre los personajes.

1. **ACCESO A LAS VISTAS:**

```
from Django. URLs import path
from. import views

urlpatterns = [
    path ('', views. Index, name='index'),
    path ('buscar/', views. buscar_personajes, name='buscar_personajes'),
]
```

2. **OBTENER DATOS DE LA API**

```
import requests
from django. shortcuts import render

API_URL = "https://rickandmortyapi.com/api/character/"

# Vista principal
def index(request):
```

```
    return render(request, 'personajes/index.html')


# Vista para buscar personajes
def buscar_personajes(request):
    query = request.GET.get('nombre', '')  # Obtener el parámetro de búsqueda
    resultados = []


    if query:
        response = requests.get(API_URL, params={'name': query})
        if response.status_code == 200:  # Si la API responde correctamente
            data = response.json()
            resultados = data.get('results', [])  # Obtener los resultados


    # Renderizar los resultados en la plantilla
    return render(request, 'personajes/resultados.html', {'resultados': resultados, 'query':
query})
```

3. **PÁGINA PRINCIPAL (INDEX.HTML) CONTIENE UN FORMULARIO PARA BUSCAR PERSONAJES.**

```
<!DOCTYPE html>
<html>
<head>
    <title>Buscador Rick & Morty</title>
</head>
<body>
    <h1>Busca personajes de Rick & Morty</h1>
    <form method="get" action="{% url 'buscar_personajes' %}">
        <input type="text" name="nombre" placeholder="Nombre del personaje">
        <button type="submit">Buscar</button>
    </form>
</body>
</html>
```

4. **RESULTADOS DE LA BÚSQUEDA (RESULTADOS.HTML) RENDERIZA LOS DATOS EN TARJETAS (*CARDS*).**

```
<!DOCTYPE html>
<html>
<head>
  <title>Resultados</title>
</head>
<body>
  <h1>Resultados de búsqueda para "{{ query }}"</h1>
  {% if resultados %}
    <div style="display: flex; flex-wrap: wrap;">
      {% for personaje in resultados %}
        <div style="border: 1px solid #ccc; padding: 10px; margin: 10px; width: 200px;">
          <img src="{{ personaje.image }}" alt="{{ personaje.name }}" style="width: 100%;">
          <h3>{{ personaje.name }}</h3>
          <p>Estado: {{ personaje.status }}</p>
          <p>Última ubicación: {{ personaje.location.name }}</p>
          <p>Primer episodio: {{ personaje.episode|first }}</p>
        </div>
      {% endfor %}
    </div>
  {% else %}
    <p>No se encontraron resultados.</p>
  {% endif %}
</body>
</html>
```

5. **DISEÑO DE IMÁGENES**

```
body {
  color: #566787;
```

```
    background: #f5f5f5;
    font-family: 'Roboto', sans-serif;
}

.table-responsive {
    margin: 30px 0;
}

.table-wrapper {
    min-width: 1000px;
    background: #fff;
    padding: 20px;
    box-shadow: 0 1px 1px rgba(0, 0, 0, .05);
}

.table-title {
    padding-bottom: 10px;
    margin: 0 0 10px;
    min-width: 100%;
}

.table-title h2 {
    margin: 8px 0 0;
    font-size: 22px;
}

.search-box {
    position: relative;
    float: right;
}

.search-box input {
```

```
    height: 34px;
    border-radius: 20px;
    padding-left: 35px;
    border-color: #ddd;
    box-shadow: none;
}

.search-box input:focus {
    border-color: #3FBAE4;
}

.search-box i {
    color: #a0a5b1;
    position: absolute;
    font-size: 19px;
    top: 8px;
    left: 10px;
}

table.table tr th,
table.table tr td {
    border-color: #e9e9e9;
}

table.table-striped tbody tr:nth-of-type(odd) {
    background-color: #fcfcfc;
}

table.table-striped.table-hover tbody tr:hover {
    background: #f5f5f5;
}
```

```
table.table th i {
    font-size: 13px;
    margin: 0 5px;
    cursor: pointer;
}

table.table td:last-child {
    width: 130px;
}

table.table td a {
    color: #a0a5b1;
    display: inline-block;
    margin: 0 5px;
}

table.table td a.view {
    color: #03A9F4;
}

table.table td a.edit {
    color: #FFC107;
}

table.table td a.delete {
    color: #E34724;
}

table.table td i {
    font-size: 19px;
}
```

```
.pagination {
    float: right;
    margin: 0 0 5px;
}

.pagination li a {
    border: none;
    font-size: 95%;
    width: 30px;
    height: 30px;
    color: #999;
    margin: 0 2px;
    line-height: 30px;
    border-radius: 30px !important;
    text-align: center;
    padding: 0;
}

.pagination li a:hover {
    color: #666;
}

.pagination li.active a {
    background: #03A9F4;
}

.pagination li.active a:hover {
    background: #0397d6;
}

.pagination li.disabled i {
    color: #ccc;
```

```
}

.pagination li i {
    font-size: 16px;
    padding-top: 6px
}

.hint-text {
    float: left;
    margin-top: 6px;
    font-size: 95%;
}
```

JUSTIFICACIÓN DE DECISIONES

1. **Uso de la API externa**: Elegimos consumir la API directamente para garantizar datos actualizados sin necesidad de almacenamiento local.
2. **Renderizado dinámico**: Se utiliza Django para crear *cards* de manera flexible y escalable.
3. **Gestión de errores**: Verificamos la respuesta de la API para manejar posibles errores.