Rick&Morty Web

El trabajo consiste en implementar una aplicación web usando **Django** que permita buscar imágenes de los personajes de la serie **Rick & Morty**, usando su API homónima. La información que provenga de esta API será renderizada por el *framework* en distintas *cards* que mostrarán -como mínimo- la imagen del personaje, el estado, la última ubicación y el episodio inicial. Adicionalmente se desarrolló la lógica necesaria para hacer funcionar el buscador central y un módulo de autenticación básica (usuario/contraseña) para almacenar uno o más resultados como **favoritos**, que luego podrán ser consultados por el usuario al loguearse. En este último, la app deberá tener la lógica suficiente para verificar cuándo una imagen fue marcada en favoritos.

- **Galería**

Al iniciar el proyecto se mostrará al usuario una pantalla con un mensaje de bienvenida. En el margen superior izquierdo se observa un componente nav con las opciones de inicio, galería e iniciar sesión. Para brindar funcionalidad a la pantalla de galería, desde la capa de servicio se llama a la función getAllImages, que retorna una lista de personajes que obtendrá consumiendo la API. Se itera con un ciclo for la lista con los datos, convirtiendo cada uno en una card y agregandola a una nueva lista.

```python
def getAllImages(input=None):  3 usages  unknown
    # obtiene un listado de datos "crudos" desde la API, usando a transport.py.
    json_collection = []
    json_collection = getAllImages1(input)
    # recorre cada dato crudo de la colección anterior, lo convierte en una Card y lo agrega a images.
    images = []
    for i in range(0, len(json_collection)):
        character = Card(
            json_collection[i]["image"],
            json_collection[i]["name"],
            json_collection[i]["status"],
            json_collection[i]["location"]["name"],
            json_collection[i]["origin"]["name"],
        )
        images.append(character)
    return images
```

Desde views.py se llamará a esta función con el objetivo de renderizar las cards correspondientes.

```python
def home(request):  1 usage  unknown +1
    favourite_list = services.getAllFavourites(request)
    images = getAllImages()
    return render(request, template_name: 'home.html', context: { 'images': images, 'favourite_list': favourite_list })
```

En el archivo home.html se itera sobre la lista de personajes, generando la card de cada uno de ellos. En la iteración se verificará el estado de cada uno de ellos(alive, dead, unknown) para asignar el color de los bordes. La misma verificación se realiza para indicar el estado del personaje en el componente dentro de la card.

```html
<div class="row row-cols-1 row-cols-md-3 g-4">
    {% if images|length == 0 %}
    <h2 class="text-center">La búsqueda no arrojó resultados...</h2>
    {% else %} {% for img in images %}
    <div class="col">
        {%if img.status == 'Alive'%}
        <div class="card mb-3 ms-5" style="max-width: 540px; border-color: green;">
            {% elif img.status == 'Dead' %}
            <div class="card mb-3 ms-5" style="max-width: 540px; border-color: red;">
                {% else %}
                <div class="card mb-3 ms-5" style="max-width: 540px; border-color: orange;">
                    {% endif %}
            <div class="row g-0">
                <div class="col-md-4">
                    <img src="{{ img.url }}" class="card-img-top" alt="imagen">
                </div>
```

```html
                <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>
```

- **Buscador**

El usuario ingresara caracteres en el input de búsqueda. En caso de que no ingrese ningún caracter y presione el botón de todas formas se renderizará la pantalla home. En caso contrario, se renderiza las cards con los personajes que matcheen con lo ingresado por el usuario.

```
def search(request):  1 usage  unknown +1
    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 != ''):
        return render(request, template_name: 'home.html', context: { 'images': getAllImages(search_msg) })
    else:
        return redirect('home')
```

- **Inicio de sesión**

Para realizar el inicio de sesión, se modificó login.html para que ejecute la acción de login.

```
<div class="login-form" style="...">
    <form action="{% url 'login' %}" method="POST" style="...">
        {% csrf_token %}
        <h2 class="text-center">Inicio de sesión</h2>
```

Modificamos header.html para que ejecute logout.

```
<li class="nav-item">
    {% if request.user.is_authenticated %}
    <a class="nav-link" href="{% url 'logout' %}">Salir</a> {% else %}
    <a class="nav-link" href="{% url 'login' %}">Iniciar sesión</a> {% endif %}
</li>
```

Y finalmente modificamos views.py para que al ejecutarse la función exit se renderice la pantalla de inicio.

```
@login_required  1 usage  Mariano Avancini +1
def exit(request):
    return render(request, template_name: 'index.html')
```

- **Favoritos**

Una vez desarrollada la función de inicio de sesión continuamos con el feature para agregar favoritos.

- **Capa de servicios**

En la función saveFavourite llamamos a fromTemplateToCard para que transforme el request en una card. Luego le asignamos el usuario correspondiente.

```
# añadir favoritos (usado desde el template 'home.html')
def saveFavourite(request):  1 usage  Mariano Avancini +1
    fav = fromTemplateIntoCard(request) # transformamos un request del template en una Card.
    fav.user = get_user(request) # le asignamos el usuario correspondiente.
    return repositories.saveFavourite(fav) # lo guardamos en la base.
```

Por otro lado, en la función getAllFavourites llamamos a getAllFavourites de la capa de repository pasandole el usuario para que retorne la información que será iterada y convertida en cards utilizando la función fromRepositoryIntoCard.

```python
# usados desde el template 'favourites.html'
def getAllFavourites(request):  2 usages  unknown +1
    if not request.user.is_authenticated:
        return []
    else:
        user = get_user(request)
        favourite_list = repositories.getAllFavourites(user) # buscamos desde el repositories.py TODOS los favoritos del usuario (variable 'user').
        mapped_favourites = []

        for favourite in favourite_list:
            card = translator.fromRepositoryIntoCard(favourite) # transformamos cada favorito en una Card, y lo almacenamos en card.
            mapped_favourites.append(card)

        return mapped_favourites
```

- **Capa de vista**

En la función home asignamos a la variable fovourite_list la función de la capa de servicios getAllFavourites.

```python
@login_required  1 usage  unknown +1
def getAllFavouritesByUser(request):
    favourite_list = services.getAllFavourites(request)
    return render(request, template_name: 'favourites.html', context: { 'favourite_list': favourite_list })
```

En la función getAllFavouritesByUser llamamos a getAllFavourites de la capa de servicios para que retorne la lista de favoritos del usuario y la renderice.

```python
@login_required  1 usage  unknown +1
def getAllFavouritesByUser(request):
    favourite_list = services.getAllFavourites(request)
    return render(request, template_name: 'favourites.html', context: { 'favourite_list': favourite_list })
```

Para añadir favoritos modificamos la función saveFavourite de views.py para que llame a la función saveFavourite de la capa de servicios pasandole el request y finalmente redireccionando al usuario a la pantalla home actualizada.

```python
@login_required  1 usage  Mariano Avancini +1
def saveFavourite(request):
    services.saveFavourite(request)
    return redirect('home')
```

Para eliminar favoritos, modificamos views.py para que ejecute la función de la capa de servicio deleteFavourite pasándole por parámetros en request y finalmente redireccionando al usuario a la misma pantalla pero actualizada.

```python
@login_required  1 usage  Mariano Avancini +1
def deleteFavourite(request):
    services.deleteFavourite(request)
    return redirect('favoritos')
```