COM-06

INFORME

TP de introduction a la programacion

ALUNMNOS:

- LUCIANO ABEL FERRANDO
- Ariel David palacio
- Caro victor santiago

Este trabajo trata sobre una app web que se basa en un buscador de personajes de la tan icónica serie llamada “Rick and Morty” en esta podemos encontrar los personajes de dicha serie junto con información adicional como su episodio inicial, ultima ubicación, su estado ya sea muerto, vivo o si es desconocido junto a una imagen del personaje en sí.

- 1) **Views.py:**

```
def home(request):
    images = getAllImages()
    favourite_list = []

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

En la imagen esta una función que se encarga de tomar dos listas una de “images” y otra “favourite_list” esta función se encarga de mostrar las imágenes en la app web dicho esto en este punto trabajaremos con la lista “images” lo que hicimos fue llamar a la función “getallimages” esta función está desarrollada en otro archivo este punto no resulto tener una gran dificultad a la hora de realizarlo.

- 2) **service.py:** en esta parte del codigo desarrollamos la funcion “getAllImages” la cual fue utilizada en el anterior punto, en primer lugar llame a la funcion getAllImages desde el archivo transport a la cual le asigne una variable en este caso “X” luego desarrollo dicha funcion con un “for” para recorrer json_collection el cual contiene los datos tipo json que paso a convertir a tipo card para que pueda ser utilizado en el anterior punto los guardo en “card” la cual despues agrego con append a images y luego retornar las imagenes en elformato requerido por el punto anterior.

```python
# capa de servicio/lógica de negocio #python manage.py runserver 3000

from ..persistence import repositories
from ..utilities import translator
from django.contrib.auth import get_user
from app.layers.transport.transport import getAllImages as x

def getAllImages(input=None):
        You, 1 second ago • Uncommitted changes
    # obtiene un listado de datos "crudos" desde la API, usando a transport.py.
    json_collection = x(input)
    # recorre cada dato crudo de la colección anterior, lo convierte en una Card y lo agrega a images.
    images = []
    for datos in json_collection:
        card = translator.fromRequestIntoCard(datos)
        images.append(card)
    return images
```

- 3)**Home.html:**

```
<div class="col-md-3">
    <div class="card mb-3 ms-5
    {% if img.status == 'Alive' %} border-success
    {% elif img.status == 'Dead' %} border-danger
    {% else %} border-warning
    {% endif %}"
```

En cada carta hay condicionales que harán que difiera el borde de la misma. Por ejemplo, se activará la condicional “if” o “elif”, de acuerdo al estado de cada personaje (img.status): si esta vivo (alive) su borde de tarjeta será verde (border-success), si esta muerto será rojo (border-danger) y si no se cumple con ninguna condicional, será amarillo (border-warning)

```
<p class="card-text">
    <strong>
                {% if img.status == 'Alive' %} 🟢 {{ img.status }}
                {% elif img.status == 'Dead' %} 🔴 {{ img.status }}
                {% else %} 🟠 {{ img.status }}
                {% endif %}
            </strong>
```

Para el circulo que también define el estado del personaje, se aplico la misma lógica

Opcionales:

- **Buscador:**

```python
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(search_msg)
        favourite_list=services.getAllImages(search_msg)
        return render(request, 'home.html', {
            'images': images,
            'favourite_list': favourite_list,
            'search_msg': search_msg
        })
    else:
        return redirect('home')
```

en primer lugar, la función search sirve para filtrar los personajes que desea buscar el usuario la variable "query" obtiene el texto de búsqueda luego con un condicional verifica si el usuario tipeo un texto o no en el caso de que haya buscado un personaje con la función "getallimages" busca las imágenes con el parámetro que el usuario haya puesto en el caso de que no haya buscado nada se re direcciona a el home.

- **Loading Spinner para la carga de imágenes:**

```html
<div id="loader"
    style="display: none; position: fixed; top: 0; left: 0; width: 100%; height: 100%;
        background: #11182ef0; z-index: 9999; justify-content: center; align-items: center;">
    <div class="spinner"></div>
</div>

<script>
    document.addEventListener("DOMContentLoaded", function () {
        const loader = document.getElementById("loader");
        const links = document.querySelectorAll("a");

        loader.style.display = "none";

        if (links.length > 0) {
            links.forEach(link => {
                link.addEventListener("click", function (event) {
                    loader.style.display = "flex";
                });
            });
        }
    });

    window.addEventListener("pageshow", function () {
        const loader = document.getElementById("loader");
        loader.style.display = "none";
    });
</script>
```

En el archivo Header.html se aplicó el código (id=”loader”) que servirá para indicar la pantalla de carga en el programa. Dentro del mismo se personalizó con el código “style”, agregándole un fondo semitransparente (background), que esté oculto por defecto (display:none), su contenido esté centrado (justify-content: center; align-items: center) y que ocupe toda la pantalla, ya que se superpone con la página actual (position: fixed; top: 0; left: 0; widht: 100%; height: 100%).

En el “loader” también habrá un estilo(class) llamado “spinner” que será la animación de carga .

Mas adelante se implementó javascript, para el correcto funcionamiento del “loader”:

Se busca en el documento, un método de registro llamado “addEventListener” y se activará cuando el “DOMContentLoaded” este completamente cargado, verificándose con “getElementById” y “querySelectorAll”, si “loader” y los elementos de la página(a) existen antes de tener que acceder a ellos. Una vez que esté cargada la página se asegura que “loader” este oculto volviendo a poner el código “none” (loader.style.display = “none”) ya que sin esto, puede activarse por una navegación previa.

Se agregó la condición de si hay enlaces en la página, “length”, los va a enumerar (if (links.lenght >0)), “for each” los va a ir recorriendo y a cada uno les pondrá un “addEventListener” que se activara cuando el usuario haga clic en el enlace y ejecutara el evento de cambiar el estado de “none” a “flex”, en el “loader” haciéndolo visible (loader.style.display=”flex”)

Dificultades:

Al volver atrás en el historial de la pagina se mostraba el loader, sin poder acceder a otros enlaces de ninguna forma que no sea reiniciando la página.

Solución:

Se tuvo que recurrir, a que en la ventana del navegador hubiera un “addEventListener”, que detectará un “pageshow”, que es el momento en donde una página web se muestra en la ventana. Se ejecutará un “getElementsById” para buscar al “loader” y cambiarle el estado a “none”

```css
.spinner {
    width: 50px;
    height: 50px;
    border: 5px solid #ccc;
    border-top-color: #007bff;
    border-radius: 50%;
    animation: spin 1s linear infinite;
}

@keyframes spin {
    from {
        transform: rotate(0deg);
    }
    to {
        transform: rotate(360deg);
    }
}
```

En styles.css agregé un estilo “spinner”, que es la animación en el centro para indicar que está cargando y lo fui editando: width= ancho, height=altura, border= grosor del borde, border-top-color= color de la parte superior del borde, border-radius: redondea las esquinas, animation= aplica una animación (en este caso spin,dura 1 segundo, constante e infinita)

Se utiliza “@keyframes” para definir la animación de spin: al iniciar la animación (from) el spinner se encuentra en su posición inicial (rotate(0deg)) y al finalizarla (to) el spinner habrá rotado 360 grados (rotate(360deg))

- **Renovar interfaz gráfica:**

```html
<body
    style="font-family: 'roboto', sans-serif; background-color: #1b202e ;">
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark shadow-sm ">
```

En el archivo Header.html, se aplicó un fondo azul oscuro y se modificó el navbar para que sea negro con sombra (navbar-dark bg-dark shadow-sm)

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

<div class="d-flex justify-content-center mt-4" style="margin-bottom: 1%">
```

En el archivo Home.html, se modificó el título “buscador Rick y Morty” cambiándole al color y bajándolo un poco en el archivo index.html también se aplicó este cambio), ya que chocaba con el navbar (text-primary mt-4). También se centró el paginador (justify-content-center mt-4)

```
<div class="d-flex justify-content-center mb-4" style="margin-bottom: 1%">
    <!-- Buscador del sitio -->
    <form class="d-flex w-50" action="{% url 'buscar' %}" method="POST">
        {% csrf_token %}
        <input class="form-control me-2" type="search" name="query" placeholder="Escribí una palabra" aria-label="Search">
        <button class="btn btn-primary" type="submit">Buscar</button>
    </form>
</div>
```

Se ensanchó la barra del buscador (w-50) y se cambió el color del botón de “buscar”, al color primario (btn-primary)

```
<div class="row g-4">
    {% if images|length == 0 %}
    <h2 class="text-center">La búsqueda no arrojó resultados...</h2>
    {% else %} {% for img in images %}
    <div class="col-md-3">
        <div class="card mb-3 ms-5
        {% if img.status == 'Alive' %} border-success
        {% elif img.status == 'Dead' %} border-danger
        {% else %} border-warning
        {% endif %}"
        style="background-color: #2e3f51; color: #cfd8dc; max-width: 250px;">
            <div class="row g-0">
                <div class="col-md-0">
                    <img src="{{ img.url }}" class="card-img-top" alt="imagen">
                </div>

                <div class="col-md-40">
                    <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>
```

Se ajustó la posición de las tarjetas, el tamaño y su información (g-4, g-0, col-md-0, col-md-40 max-widht: 250px), se cambió el fondo y el color de las letras.

```
pagination li a {
    border: none;
    font-size: 95%;
    width:50px;
    height: 30px;
    color: #00aaff;
    margin: 0 2px;
    line-height: 30px;
    border-radius: 5px !important;
    text-align: center;
    padding: 0;
}
```

En styles.css se bajó el nivel de redondeo de los bordes, se modificó el ancho y se cambió el color de los números (weight:30px, border-radius:5px)

En este archivo aparte de esto, cambié el color y la sombra de table-wrapper y el color de table-title