Informe

Introducción a la Programación

Integrantes

- Ulises Nahuel López - 47454907 - lulises476@gmail.com
- Felipe Martinez Tanoira - 43920367 - felipemtanoira@gmail.com
- Fabian Ariel Herrera - 44183741 - fabianariel02@gmail.com
- Lionel Edgardo Miller - 46935008 - leonmisionero29@gmail.com

En el trabajo práctico asignado se nos pidió modificar un programa que tenía por función buscar personajes de la serie llamada “Rick and morty”. En la misma, se nos presentaron las siguientes características en formato de tarjetas: nombre, estatus, última ubicación y episodio de su primera aparición.

El programa no estaba terminado, pero tenía grandes avances. Nuestra tarea era la de completar sus funcionalidades mediante la creación e implementación de código de diversos lenguajes.

Obligatorios

Views.py

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

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

Encontramos en el archivo services.py la función getAllImages() que nos traería la información de los personajes y se la asignamos a la variable *images* para que posteriormente sea enviada como información para ser procesada en el home.html.

Services.py

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

    # recorre cada dato crudo de la colección anterior, lo convierte en una Card y lo agrega a
    images.
    images = []
    for elem in json_collection:
        images.append(translator.fromRequestIntoCard(elem))

    return images
```

Encontramos en el archivo transport.py dentro de la carpeta ‘transport’ la función getAllImages() que ya se encargaba de generar la lista de personajes trayendo la información de la API y guardandola en formato lista.

Posteriormente, recorrimos esta lista ejecutando la función fromRequestIntoCard(), traída del archivo translator.py, para cada elemento de la lista para convertir la información de formato Json a Card.

Finalmente devolvemos esta nueva lista ya preparada para ser utilizada.

Home.html

```
{% else %} {% for img in images %}
<div class="col">
    <div class="card
                    {% if img.status == 'Alive' %} border-success
                    {% elif img.status == 'Dead' %} border-danger
                    {% else %} border-warning
                    {% endif %}
                    mb-3 ms-5" style="max-width: 540px;">
        <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>
```

En primer lugar implementamos un *if* para determinar el status del personaje y posteriormente, dependiendo del mismo, le asignamos un borde con un color con la clase border-(color).

Segundo, nos dimos cuenta que el if que agrega el botón y el status del usuario, no estaba funcionando adecuadamente, así que cambiamos los condicionales para que este funcione correctamente.

Opcionales

Buscador

Views.py

```python
def search(request):
    #search_msg guarda el valor de la información busacada en el buscador
    search_msg = request.POST.get('query', '')
    #images alverga la información de TODOS los personajes
    images=services.getAllImages()

    #este "if" se ejecuta cuando el usuario 'busca' algo en la barra del buscador, ejecutando la
    función 'buscar' del archivo de services.py y enviando el resultado al html de home bajo el
    nombre de imges.
    #De no encontrar resultado muestra el mensaje "La búsqueda no arrojó resultados..." o, en caso
    de no rellenar la barra, redirije a 'home'.

    if (search_msg != ''):
        images = services.buscador(search_msg)
        return render(request, 'home.html', { 'images': images})
    else:
        return redirect('home')
```

Esta función se ejecuta al buscar algo en la barra de búsqueda del home.

Guarda la información buscada bajo la variable *search_msg.*

La variable Images guarda la lista de los primeros 20 personajes de manera default.

Posteriormente, el "if" se ejecuta cuando el usuario 'busca' algo en la barra del buscador, ejecutando la función 'buscar' del archivo de services.py y enviando el resultado obtenido al html de home bajo el nombre de images.

De no encontrar resultado muestra el mensaje "La búsqueda no arrojó resultados..." o, en caso de no rellenar la barra de búsqueda, redirige a 'home'.

Services.py

```python
# ejecuta la función de 'getAllImages' del archivo transport con el input de la barra de busqueda y
los devuelve en una lista
def buscador(input):
    json_collection = transport.getAllImages(input)
    images = []
    for elem in json_collection:
        images.append(translator.fromRequestIntoCard(elem))
    return images
```

En este archivo creamos la función “buscador(input)” para traer la función getAllImages del archivo transport y ejecutarla con el input del buscador. Esta función lo que hace es buscar personajes por su nombre y devolvernos una lista con los primeros 20 que contengan el valor del input.

Nuevamente, recorrimos esta lista para transformar cada personaje de formato Json a Card, lo volvimos a agregar y a enviar en una lista.

Inicio de Sesión

Views.py

```python
from django.shortcuts import redirect, render
```

```python
from django.contrib.auth import logout
```

```python
@login_required
def exit(request):
    logout(request)
    return redirect("login")
```

Se completó la función “exit”, utilizando las funciones “logout” y “redirect” ya incluidas desde un principio, de manera que cierre sesión al ser invocada presionando el botón de Salir. Asimismo dirigiéndonos a la página “login”.

Loading Spinner

1. Agregado del Spinner en el HTML

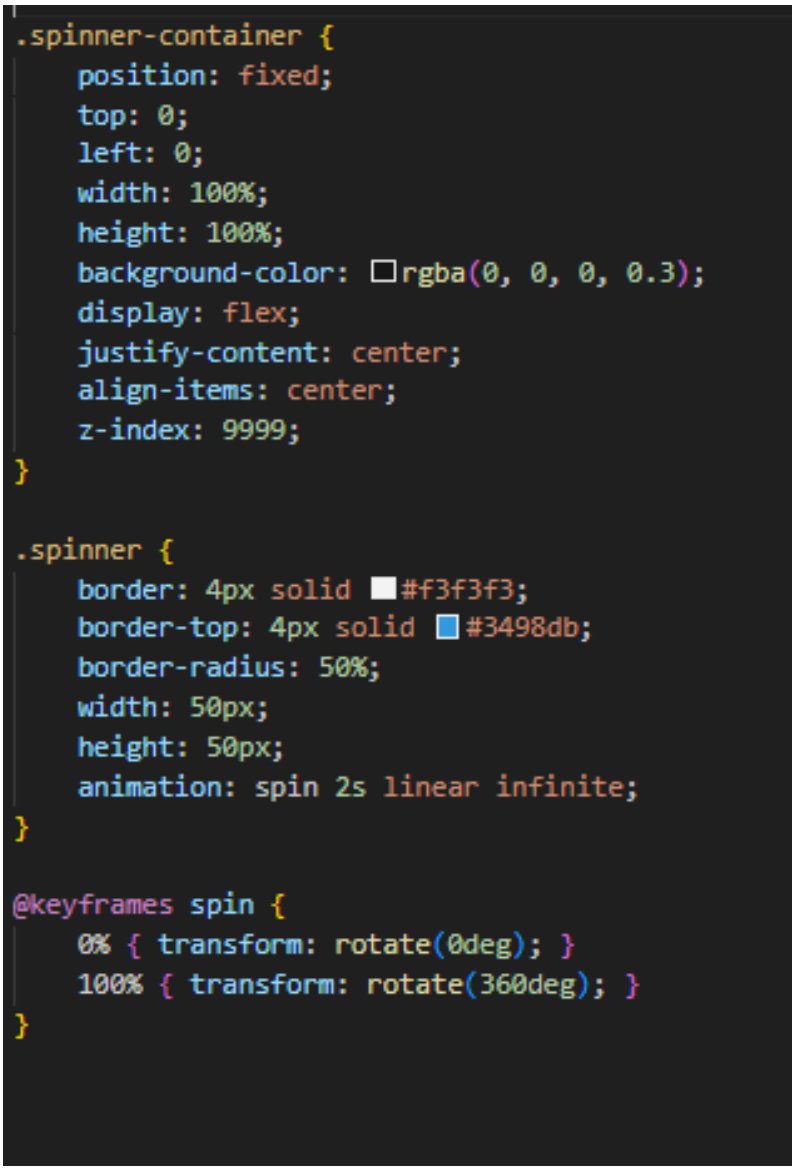
- Añadimos un contenedor para el spinner en el archivo Header.html y para el buscador le añadimos un ID “searchForm”

```html
<div id="spinner" style="display:none; justify-content: center; align-items: center; position: fixed; top: 0; left: 0; width: 100%; height:
    <div class="spinner-border text-light" role="status">
        <span class="visually-hidden">cargando...</span>
    </div>
</div>
```

```html
    <!-- Buscador del sitio -->
    <form  class="d-flex" action="{% url 'buscar' %}" method="POST" id="searchForm">
        {% csrf_token %}
```

2. Estilos CSS para el Spinner

Añadimos estilos en `styles.css` para centrar el spinner y darle diseño.



```css
.spinner-container {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, 0.3);
    display: flex;
    justify-content: center;
    align-items: center;
    z-index: 9999;
}

.spinner {
    border: 4px solid #f3f3f3;
    border-top: 4px solid #3498db;
    border-radius: 50%;
    width: 50px;
    height: 50px;
    animation: spin 2s linear infinite;
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}
```

3. JavaScript para Mostrar/Ocultar el Spinner

en el archivo Spinner.js, dentro de los archivos estàticos se creó toda la lógica para que el spinner se muestre y se oculte. Optamos por Javascript ya que nos parecía mejor para el manejo del DOM.

```
//SPINNER PARA LAS SECCIONES

document.addEventListener('DOMContentLoaded', function () {
    // Elementos del DOM necesarios
    const spinner = document.getElementById('spinner');
    const links = document.querySelectorAll('a.nav-link');

    // Verifica si el spinner existe
    if (!spinner) {
        console.error('Spinner element not found.');
        return;
    }

    // Agrega el evento 'click' para los enlaces de navegación
    links.forEach(link => {
        link.addEventListener('click', function (event) {
            // Evita comportamiento por defecto si el link es inválido
            if (!link.href || link.href === '#') {
                return;
            }

            // Muestra el spinner antes de redirigir
            spinner.style.display = 'flex';
        });
    });

    // Asegura que el spinner desaparezca al cargar la nueva página
    window.addEventListener('load', function () {
        spinner.style.display = 'none';
    });
});
```

Esta es la lógica para el loading spinner para el buscador y arriba para la parte de las secciones.

```
//SPINNER PARA EL BUSCADOR

document.addEventListener('DOMContentLoaded', function () {
    // Mostrar el spinner cuando se envía el formulario de búsqueda
    const searchForm = document.getElementById('searchForm');
    if (searchForm) {
        searchForm.addEventListener('submit', function (e) {
            // Mostrar el spinner al hacer submit
            document.getElementById('spinner').style.display = 'flex';
        });
    }

    function hideSpinner() {
        document.getElementById('spinner').style.display = 'none';
    }
});
```

Interfaz Gráfica

Styles.css (cambios generales)

Los cambios realizados en los estilos de la página fueron muchos. A continuación destacaremos los más importantes.

```css
body {
    color: #f1f1f1;
    background: #1e2021;
    font-family: 'Montserrat', sans-serif;
}
```

En “body” se cambió el color de la fuente a un tono blanco, el fondo a un tono oscuro y se modificó la fuente.

```css
.background {
    background-image: url('https://wallpapercat.com/w/full/1/f/5/
    46701-3840x2160-desktop-4k-rick-and-morty-background-image.jpg');
    background-size: cover; /* El bg se ajusta a la pantalla */
    background-position: center;
    background-repeat: no-repeat; /* No se repite en caso de que la página sea larga */
    height: 92vh;
    color: #f1f1f1;
    display: flex;
    flex-direction: column; /* Conseguimos que no cambie las ubicaciones de los contenidos como el login */
    justify-content: center;
  }
```

En “login.html” se agregó una imágen de fondo, en la cual ajustamos su resolución, y la posicionamos correctamente.

header.html

```html
<nav class="navbar navbar-expand-lg bg-body-tertiary">
    <div class="container-fluid">
        <a class="navbar-brand" href="#">Proyecto TP</a>
    </div> <!-- Brand Separated -->
    <div class="container-fluid justify-content-end"> <!-- las secciones se mueven a la derecha -->
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent"
        aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
        <div class="collapse navbar-collapse flex-grow-0" id="navbarSupportedContent"> <!-- flex grow 0 Added -->
            <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>
```

Las secciones fueron separadas de “Brand” (el título de ProyectoTP) para ser reposicionadas de una manera más fácil. Además, se agregó “flex grow 0” a las secciones para que se posicionen de la manera esperada.

footer.html

```
<footer class="bg-body-tertiary text-center text-lg-start fixed-bottom">
    <div class="text-center p-3" style="display: flex;background-color: rgb(56 56 56 / 78%);justify-content:
    space-between;">      You, yesterday • Styles + modo oscuro aplicados
        <strong>Introducción a la Programación | UNGS</strong>
        <strong>{{VERSION}}</strong>
    </div>
</footer>
```

Se modificó el color de su fondo a uno más oscuro.

index.html

```
<div id="rymImage">
    <img src="https://www.pngplay.com/wp-content/
    uploads/12/
    Rick-And-Morty-Wallpaper-PNG-Photo-Image.png"
    alt="Rick y morty portal">
</div>
```

```
#rymImage {
    text-align: center;
}

#rymImage img {
    height: 70vh;
}
```

Se agregó una imágen y se la modificó en “styles.css”

home.html

```
<div class="d-flex justify-content-center" style="margin-top: 5%">
    <!-- Buscador del sitio -->
    <form class="d-flex" action="{% url 'buscar' %}" method="POST">
        {% csrf_token %}
        <input class="form-control me-2" type="search" name="query" placeholder="Busca tu personaje"
        aria-label="Search">
        <button class="btn btn-outline-success" type="submit">Buscar</button>
    </form>
</div>

<div class="d-flex justify-content-end" style="margin-bottom: -4%; margin-top: 6%; margin-right: 2rem;">
    <!-- Selector de página -->
    <nav aria-label="...">
        <ul class="pagination">
            <li class="page-item">
                <a class="page-link"><svg xmlns="http://www.w3.org/2000/svg" width="28" height="28"
                fill="currentColor" class="bi bi-arrow-left-circle-fill" viewBox="0 0 15.5 17.5">
                    <path d="M8 0a8 8 0 1 0 0 16A8 8 0 0 0 8 0m3.5 7.5a.5.5 0 0 1 0 1H5.707l2.147 2.146a.5.5 0
                    0 1-.708.708l-3-3a.5.5 0 0 1 0-.708l3-3a.5.5 0 1 1 .708.708L5.707 7.5z"/>
                </svg></a>
            </li>
            <li class="page-item active" aria-current="page">
                <a class="page-link" href="#">1</a>
```

Se colocó la paginación (html) por debajo del buscador para evitar posicionamientos raros.

```
div.card-body, div.col-md-4 {
    background-color: #232123;
}
```

El color de la tarjeta y su imagen se cambiaron a un tono oscuro en “styles.css”

```
.col:hover {
    transform: translateY(-5px) scale(1.02); /* Menor movimiento y escalado */
    transition: .3s;
    background-color: #292929;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.15); /* Sombra más ligera */
}

.col {
    transition: transform 0.2s ease, box-shadow 0.2s ease;
}
```

Se agregó una implementación *hover* en “styles.css” a las tarjetas de los personajes para destacar mejor al seleccionado.

favourites.html

```css
/* Título */
.col-sm-8 {
    color: #008dc4;
    width: 100%;
    text-align: center;
    animation: cambioColor 3s infinite;
}

@keyframes cambioColor {
    0% { color: #ff6600; }
    50% { color: #ffcc00; }
    100% { color: #ff6600; }
}
```

```css
/* Tabla favoritos */

.table-wrapper {
    background-color: #323232;
    border-radius: 12px;
}

#tabla {
    border: 2px solid #888;
    color: #f1f1f1;
}
```

Por último, se agregó un color de título que varía, y unos tonos oscuros a la tabla de favoritos.