Cuervo Pilar; Fernández Angela; Garvich Pedro; Leiva Jesica; Ríos Nicolás.

Trabajo práctico de Introducción a la Programación

En el presente informe se describe cómo fue el desarrollo del trabajo práctico de la materia Introducción a la programación, donde el trabajo consiste en implementar una aplicación web usando Django. Esta aplicación permite buscar imágenes de los personajes de la serie Rick & Morty, usando su API homónima.

Algunos detalles que se tuvieron en cuenta para la realización del trabajo práctico son que la información que provenga de ésta API sea renderizada por el *framework* en distintas tarjetas que mostrarán la imagen del personaje, el estado, la última ubicación y el episodio inicial.

Adicionalmente se espera, entre otras cosas, que los estudiantes desarrollen 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.

En primer lugar, se mostrará lo realizado en las funciones que faltaban implementar de los módulos *views.py* - *services.py* y el template *home.html* que faltaba modificar . Éstas son las encargadas de hacer que las imágenes de la galería se muestren.

- **views.py:**

  home(request): muestra la página inicial, obtiene 2 listados que corresponden a las imágenes de la API y los favoritos del usuario, y los usa para dibujar el correspondiente template. Si el opcional de favoritos no está desarrollado, devuelve un listado vacío.

  ```
  def home(request):
      images = []
      images = images + services.getAllImages()
      favourite_list = services.getAllFavourites(request)
      return render(request, 'home.html', {'images': images, 'favourite_list': favourite_list})
  ```

- **services.py:**

  getAllImages(input=None): obtiene un listado de imágenes de la API. El parámetro *input*, si está presente, indica sobre qué imágenes/personajes debe filtrar/traer.

  ```
  def getAllImages(input=None)->list:
  ```

```
    # obtiene un listado de datos "crudos" desde la API, usando a transport.py.
    json_collection = []
    #for i in range(1,4):
    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 object in json_collection:
        images.append(translator.fromRequestIntoCard(object))
    return images
```

- **home.html:**

La card debe cambiar su *border color* dependiendo del estado del personaje. Si está vivo (*alive*), mostrará un borde verde; si está muerto (*dead*) mostrará rojo y si es desconocido (*unknown*), será naranja.

```
{% if img.status == 'Alive' %}
    <div class="card mb-3 ms-5 border-success" style="max-width: 540px;
background-color: #00ff0f; color: #000000;">
    {% elif img.status == 'Dead' %}
    <div class="card mb-3 ms-5 border-danger" style="max-width: 540px;
background-color: #ff0000; color: #000000;">
    {% else %}
    <div class="card mb-3 ms-5 border-warning" style="max-width: 540px;
background-color: #ffa200; color: #000000;">
    {%endif%}
```

Modificación del código:

-En las líneas 40-46 es para generar el recuadro con el color que corresponde por fuera de la carta.

-En el condicional de las líneas 56-62 img.status es para que el círculo de adentro de la carta tenga el color según corresponda, al igual que el borde, si el personaje se encuentra vivo mostrará color verde; si está muerto mostrará rojo y si se desconoce el estado del personaje será de color naranja.

En segundo lugar, se mostrará lo realizado para los opcionales realizados en el trabajo.

- **Buscador**

  Se completó la funcionalidad para que el buscador filtre adecuadamente las imágenes, según los siguientes criterios:

  - Si el usuario NO ingresa dato alguno y hace clic sobre el botón 'Buscar', debe mostrar las mismas imágenes que si hubiese hecho clic sobre el enlace Galería.
  - Si el usuario ingresa algún dato (ej. *Samantha*), al hacer clic en 'Buscar' se deben desplegar las imágenes filtradas relacionadas a dicho valor.

```
def getAllFavouritesByUser(request):

    favourite_list = services.getAllFavourites(request)# llama al servicio para obtener favoritos


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

Modificación del código:

-En views.py**:** la línea 24 se agregó la lista vacía, images_search.

-En la línea 26 el condicional.

- **Inicio de sesión**

Se completó la funcionalidad de inicio de sesión. Anteriormente no era posible salir de la sesión.

Para que el usuario Admin pueda iniciar o cerrar sesión se crearon dos funciones en views.py**:**

- La función **login_views(request)** verifica que las credenciales ingresadas en usuario y contraseña sean válidas. Si lo son, entonces mediante la función login de Django permite al usuario ingresar. Si las credenciales no son correctas entonces redirige a la página login y vuelve a pedir usuario y contraseña.

El còdigo utilizado es el siguiente:

```
def login_views(request):
    username=request.POST['username']
    password=request.POST['password']
    user=authenticate(request,username=username,password=password)
    if user is not None:
```

login(request,user)

Para que funcione correctamente se importa de Django en views.py lo siguiente:

from django.contrib.auth import authenticate, login

- La función **exit (request)** permite que el usuario cierre sesión utilizando la función logout(request). Luego se solicita que retorne la función login que es donde se inicia sesión, es decir que redirige a la pantalla de inicio de sesión.

  El còdigo utilizado es el siguiente:

```
def exit(request):
    logout(request)
    return redirect('login')
```

Para que funcione correctamente se importa de Django en views.py:

from django.contrib.auth import logout
from dajngo.shortcuts import render, redirect

Para que las funciones agregadas en views.py funcione correctamente se agrega en url.py:

from django.contrib.auth import authenticate, login, logout
from django.shortcuts import render, redirect
from .views import login_views

Observación: inicialmente habían comandos en la función que luego se quitaron para su correcto funcionamiento, lo mencionamos a continuación:

- De login_views se eliminó la rama else pues debería devolver un mensaje de error en caso de loguearse con las credenciales incorrectas pero no fue posible hacerlo funcionar.
- La rama: **if request.method='post'** porque lo que se recibe es un formulario y ya se está trabajando con el método POST por lo que no es necesario chequear que lo sea.
- **return render (request, 'login.html')** pues la función renderiza la página automáticamente, este comando no es necesario.

- **<u>Favoritos</u>**
  - En services.py se completó la función saveFavourite requiriendo a la función fromRequestIntoCard de translator y asociando la solicitud al usuario.

```
def saveFavourite(request):
    fav = translator.fromTemplateIntoCard(request) # transformamos un request del template en una Card.
    fav.user = request.user # le asignamos el usuario correspondiente.
    repositories.saveFavourite(fav)
    return fav  # lo guardamos en la base.
```

Se completó la función getAllFavourites que permite enlistar todas las tarjetas favoritas del usuario.

```
def getAllFavourites(request):
  if not request.user.is_authenticated:
    return []
  else:
    user = get_user(request)
    favourite_list = repositories.getAllFavourites(request.user) # buscamos desde el repositories.py TODOS los favoritos del usuario (variable 'user').
    mapped_favourites = []
    for favourite in favourite_list:
      mapped_favourites.append(translator.fromRepositoryIntoCard(favourite))  # transformamos cada favorito en una Card, y lo almacenamos en card.
    return mapped_favourites
```

- En views.py se definió la función getAllFavouritesByUser que constituye la lista de favoritos por usuario recurriendo a la función getAllFavourites.

```
@login_required
def getAllFavouritesByUser(request):
    favourite_list = services.getAllFavourites(request) # llama al servicio para obtener favoritos
  return render(request, 'favourites.html', { 'favourite_list': favourite_list })
```

También se definió saveFavourite solicitando al servicio del mismo nombre, al igual que con el servicio deleteFavourite.

```
def saveFavourite(request):
  services.saveFavourite(request)
  return redirect(home) #llama al servicio para guardar el favorito

def deleteFavourite(request):
```

```
    services.deleteFavourite(request) #llama al servicio para eliminar el favorito
        return redirect(getAllFavouritesByUser)
```

Llama a la función deleteFavourite de services.py:

```
def deleteFavourite(request):
    favId = request.POST.get('id')
    return repositories.deleteFavourite(favId) # borramos un favorito por su ID.
```

En todos los casos funciona como conexión entre la solicitud del usuario en formato HTTP con los servicios solicitados.

En el caso de los servicios existió el problema de querer traer la información mediante el método que la trae desde la API directamente, en lugar de usar otro que la trae desde el template.

- **Interfaz de usuario**

  - **Descargas**

Se descargó la tipografía get_schwifty utilizada en la serie y se la guardó en **/TP-IP2024/static/Fuentes** bajo el nombre mortyfont-ttf.

Además, se sumaron 3 imágenes guardadas en **/TP-IP2024/static/Imagenes,** para utilizarlas una como fondo de pantalla, y las otras dos para el cambio de página.

  - **Tipografía**

Se declaró la tipografía descargada en el archivo .css y se agregó al estilo de CSS para body. Esto permitió de manera automática modificar la mayor parte de la tipografía puesto que estaba asociada a esta etiqueta.

```
@font-face {
    font-family: 'get_schwifty';
    src: url('Fuentes/mortyfont.ttf') format('truetype');
    font-weight: normal;
    font-style: normal;
}
```

Además, se armaron estilos css para los títulos etiquetados “h1”, “h2” y para el footer utilizando dicha tipografía y modificando en cada caso tamaño de letra y colores (intentando adaptarlos a la paleta de la serie).

  - **Body**

Además del cambio de la tipografía se agregó una imagen de fondo y se la ajustó de tal manera que estuviera centrada, ocupará todo el espacio y al desplazarse por la página esta se mantuviera anclada o estática.

Se aplicó un color a la tipografía.

```
body {
    background: #04021f;
    background-image: url('Imagenes/Fondo.jpg');
    background-size: cover;
    background-position: center;
    background-attachment: fixed;
    font-family: 'get_schwifty', 'Roboto';
    color: #33ff9f;
}

h1.text-center {
    font-family: 'get_schwifty', sans-serif;
    font-size: 60px;
    color: #00f0ff;
    -webkit-text-stroke: 2px #97ce4c;
    text-align: center;
}
h2.text-center {
    font-family: 'get_schwifty', sans-serif;
    font-size: 30px;
    color: #00f0ff;
    -webkit-text-stroke: 2px #97ce4c;
    text-align: center;
}

h10.text-footer {
    font-family: 'get_schwifty', sans-serif;
    font-size: 20px;
    color: #f700ff;
}
```

- **home.html**

En las etiquetas de las cartas se modificó el atributo style dotándolas de un color de fondo según el estadode personaje al que correspondan y un color alternativo de la tipografía.

```
        <div class="card mb-3 ms-5 border-success" style="max-width: 540px; background-color: #00ff0f; color: #000000;"> <!-- Acá está el cambio de color-->
{% elif img.status == 'Dead' %}
        <div class="card mb-3 ms-5 border-danger" style="max-width: 540px; background-color: #ff0000; color: #000000;"> <!-- Acá está el cambio de color-->
{% else %}
        <div class="card mb-3 ms-5 border-warning" style="max-width: 540px; background-color: #ffa200; color: #000000;"> <!-- Acá está el cambio de color-->
```

Por otro lado, se sustituyeron las flechas del paginador por imágenes de un artefacto característico de la serie. Además, se eliminó el color de background del estilo creado para la etiqueta “pagination li a” para favorecer la visualización de dicha imagen.

```
<img src="../static/Imagenes/Pistola_Atras.webp" alt="Flecha1" style="width: 30px; height: 30px;">
<img src="../static/Imagenes/Pistola_Portal.webp"alt="Flecha" style="width: 30px; height: 30px;">

.pagination li a {
   border: none;
   background: 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;
}
```

Se modificó en sus propias etiquetas el color de fondo y el color de las tipografías de la caja del buscador y del botón buscar.

```
<input class="form-control me-2" style="background-color: #00073e; color: #00e8ff;" type="search" name="query" placeholder="Escribí una palabra" aria-label="Search">
<button class="btn btn-outline-success" style="background-color: #00073e; color: #00e8ff;" type="submit">Buscar</button>
```

- **login.html**

Se repitió el último procedimiento para las cajas del nombre de usuario y la contraseña.

```
<input style="background-color: #00073e; color: #00e8ff;" type="text" name="username" id="username" class="form-control" placeholder="Usuario" required="required">
<input style="background-color: #00073e; color: #00e8ff;" type="password" name="password" id="password" class="form-control" placeholder="Contraseña" required="required">
```

- **footer.html**

Se aplicó la nueva etiqueta h10 a los dos textos. Se modificó el color de la caja contenedora del footer.

```
<div class="text-center p-3" style="display: flex;background-color: rgb(15 255 0 / 25%);justify-content: space-between;">
    <h10 class="text-footer">Introducción a la Programación | UNGS</h10>
    <h10 class="text-footer">{{VERSION}}</h10>
```

- **Estilos de tablas**

Se logró modificar el color de la tipografía general de la tabla modificando el estilo de *table-title*, *table-title* h2 y *table-wrapper*. Modificando *table-wrapper* se cambió el color de fondo de la tabla. También se modificó el estilo de *table.table-striped tbody tr:nth-of-type(odd)* para que adopte un color alternativo negro. Se modificó el estilo de table.table-striped.table-hover tbody tr:hover para cambiar el color que adoptan las filas a posar el cursor sobre ellas.

```
.table-title {
    padding-bottom: 10px;
    margin: 0 0 10px;
    color: #00e8ff;
    min-width: 100%;
}
.table-title h2 {
    margin: 8px 0 0;
    color: #00ff2e;
```

```
    font-size: 22px;
}
.table-wrapper {
    min-width: 1000px;
    background: #00073e;
    color: #00e8ff;
    padding: 20px;
    box-shadow: 0 1px 1px rgba(0, 0, 0, .05);
}
table.table-striped tbody tr:nth-of-type(odd) {
    background-color: #000202;
    color: #00ff00;
}
table.table-striped.table-hover tbody tr:hover {
    background: #eea200;
}
```

- **Loading spinner**

Se descargó un gif para ser utilizado en el trabajo de la página https://loading.io/ y se la guardó en **/TP-IP2024/static/Imagenes** bajo el nombre spinner gif.

En principio se agregó en el encabezado del archivo home.html ***{% load static %}*** el cual sirve para gestionar archivos estáticos.

Se agregó cerca del final del mismo archivo el siguiente código:

```
<div id="spinner-overlay" style="display: none; position: fixed; top: 0; left: 0; width: 100%; height: 100%; background: rgba(0,0,0,0.5); z-index: 9999; justify-content: center; align-items: center;">
    <img src="{% static 'Imagenes/spinner.gif' %}" alt="Cargando..." style="width:100px; height: 100px;">
</div>
```

Este código crea una superposición opaca que cubre el centro de la pantalla y muestra el gif mientras se procesan acciones como buscar. En caso de que no se cargue el gif, se añadió un texto alternativo.

Luego, se creó un código de JavaScript para el spinner.

```
<script>
```

```
        document.addEventListener("DOMContentLoaded", function() {
        const spinnerOverlay = document.getElementById("spinner-overlay");

        const searchForm = document.querySelector("form[action='{% url 'buscar' %}']");
        if (searchForm) {
            searchForm.addEventListener("submit", function() {
                spinnerOverlay.style.display = "flex";
            });
        }
        window.addEventListener("load", hideSpinner);
        window.addEventListener("pageshow", hideSpinner);

        function hideSpinner() {
            if (spinnerOverlay) {spinnerOverlay.style.display = "none";
            }
        }
    });
    </script>
```

En la primera parte, en el código se utilizó *event listeners* para manejar el estado del spinner. Este spinner se muestra al enviar un formulario y se oculta cuando la página termina de cargar.

Además, tenemos a ***DOMContentLoaded*** que es un evento que se dispara cuando el *DOM (estructura de la página HTML)* se carga completamente, pero antes de que las imágenes y otros elementos terminen de cargarse y, ***function()*** es una función anónima que contiene el resto del código.

Por un lado, se tiene a ***getElementById("spinner-overlay")***, que selecciona el elemento HTML con el ***id="spinner-overlay"*** y *const*, que declara una constante llamada *spinnerOverlay* para almacenar una referencia al spinner.

Por otro lado, se tiene a ***querySelector*** que busca el formulario que tiene el atributo action igual a ***{% url 'buscar' %}***. Esto indica que este código es específico para un formulario de búsqueda y ***const searchForm*** guarda la referencia del formulario seleccionado en la constante *searchForm*.

A este último se le agrega un condicional ***if (searchForm)*** que verifica que el formulario existe antes de intentar añadir algún evento. Dentro del condicional encontramos a ***addEventListener("submit", function() {...})*** que asocia algún evento que se dispara al enviar el formulario (submit). Luego, ***spinnerOverlay.style.display = "flex"****;* cambia el estilo display del spinner de none (oculto) a flex, haciéndolo visible.

También se tiene a ***window.addEventListener*** que añade eventos al objeto *window* que afectan a la ventana del navegador. Un evento es *"load"* que se dispara cuando toda la página, incluidos los recursos externos (imágenes, CSS, etc.), ha terminado de cargarse. Otro evento es *"pageshow"*, similar a load, pero también se activa cuando el usuario navega de regreso a una página cargada anteriormente usando la función ***hideSpinner*** para ocultar el spinner.

- **Registro de nuevos usuarios**

En views.py se llama a la función *UserCreationForm* del módulo *django.contrib.auth.forms* y se define la función "register" de la siguiente manera:

```
def register(request):
    form = UserCreationForm(request.POST)
    if request.method == 'POST':
        if form.is_valid():
            form.save()
            user_name=request.POST['username']
            user_pwd=request.POST['password1']
            user=authenticate(request,username=user_name,password=user_pwd)
            login(request,user)
            return redirect('home')
        else:
            messages.error(request,"No se pudo crear usuario, intentelo de nuevo")
            return render(request, 'registration/register.html', {'form': form})
    return render(request, 'registration/register.html', {'form': form})
```

Luego se crea el template "register.html", el cual contendrá los campos necesarios para registrar al usuario. Finalmente en *app*/*urls.py* se agrega la siguiente línea de código:

```
path('register/', views.register, name='register')
```

- **Requisitos de la contraseña**

En register.html hemos añadido un código especificando los requisitos para la contraseña. Si estos requisitos no se cumplen, el usuario no se podrá registrar en la página. Para ello resaltamos el texto con las condiciones en un cuadro semitransparente.

```
<p style="color: white; text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.8); background-color: rgba(0, 0, 0, 0.6); padding: 15px; border-radius: 10px; font-weight: bold; max-width: 400px; margin: 0 auto; text-align: left; font-size: 12px;">
    <small>
        La contraseña debe cumplir con las siguientes condiciones:<br>
        - Tener al menos 8 caracteres<br>
        - Incluir una mayúscula<br>
        - Incluir un número<br>
        - No ser igual ni demasiado parecida al nombre de usuario
    </small>
</p>
```