Informe trabajo final
Introducción a la programación

Comisión: 13
Participantes: Elias Lazzaro y Chavez Michael

Introducción:
El objetivo de este trabajo es implementar funcionalidades esenciales en una aplicación web que permiten la gestión de usuarios y sus imágenes favoritas. Estas funcionalidades incluyen el registro, inicio de sesión, búsqueda de imágenes, paginación y manejo de favoritos. El sistema asegura que solo los usuarios autenticados puedan realizar acciones específicas, como guardar favoritos. Además, se han diseñado las páginas correspondientes para proporcionar una experiencia de usuario intuitiva y eficiente.

Cambios en Services.py

- **getApiInfo**
  - Se creó esta función para obtener los datos de todas las páginas de la api, ya que antes se daban de a 20 templates. Era necesaria a la hora de crear los índices porque no se tomaban todos los datos para hacer un cálculo de cuántas páginas debería haber.
    - Dificultades: tuvimos que leer los docs de la api para saber cómo extraer la información que necesitábamos.
    - Decisiones tomadas: Seguimos las instrucciones dadas por la api de como llegar a los datos que necesitábamos

- **saveFavourites**
  - Se le agregó la funcionalidad de asignarle a cada usuario sus respectivos personajes favoritos que el propio usuario haya agregado.
    - Dificultades: gestionar el flujo de usuarios no autenticados sin romper la lógica.
    - Decisiones tomadas: Hemos decidido que solo los usuarios autenticados puedan guardar favoritos.

- **getAllFavorites**
  - Se le agregó la funcionalidad de hacer una lista en la cual se agregan todos los favoritos de un usuario y se los transforma en una card, para luego mostrarlos en la parte de favoritos como un listado.

    - Dificultades: mapear los datos a un formato de tarjeta para mostrarlos correctamente en el template
    - Decisiones tomadas: retornar una lista vacía para usuario no autenticado , siguiendo un enfoque similar al home

- **deleteFavourites**
  - Se agregó la función de remover un favorito de la lista a través de su ID.

    - Dificultades:validar correctamente los ID para evitar errores en el servidor.
    - Decisiones tomadas: Mantener una API RESTful en la comunicación con el template.

Cambios en Transport.py

- **getAllImages**
  - se modifico esta función para que sea coherente con la forma de recorrer la api de “getApiInfo” y poder dar las imágenes que correspondan a cada índice.
    - Dificultades: No hemos tenidos ninguna dificultad en este punto
    - Decisiones tomadas:Hemos decidido agregar una nuevo código para la extracción de la página.

Cambios en Views.py

- **registro**
  - Se creó una función que permite crear un usuario ingresando los datos, para así acceder a las funciones de favoritos.
    - Dificultades: Tuvimos problemas con el SMTP ya que nos daba muchos errores al momento de enviar el correo de verificación los cuales no sabíamos solucionar.
    - Decisiones tomadas: Descartamos el envío de correo y solamente solicitamos los datos básicos para guardar en una base de datos.

- **Login_view**
  - Se permite a un usuario con cuenta ya creada ingresar.
    - Dificultades: No hubo mayores complicaciones
    - Decisiones tomadas: Manejo de la validación personalizada para asegurarse de que las contraseñas coinciden. Garantizar que el usuario sea activado automáticamente tras el registro.

- **calcular_paginacion**
  - Se creó esta función para poder mostrar los templates separados por índices,al crearlos se los redondea hacia arriba para incluir a todos las cards, estos están divididos de 10 en 10 por una cuestión estética, ya que al haber más índices en pantalla podría resultar molesto.
    - Dificultad: Hubo problemas al realizar la lógica de la función, al ser abstracta, se facilitó cuando se utilizaron números como ejemplo. Se tuvo que ir corrigiendo a medida que surgen errores

        - Decisiones tomadas: Se mostró por grupos para saber cuáles eran los índices a mostrar, y también se creó el inicio y fin “slice” para facilitar las búsquedas.

- **home**
    - Esta función obtiene dos listados: imágenes obtenidas desde la API mediante services.py , una lista de favoritos y los índices de la función calcular_paginación. Su objetivo principal es cargar el template home.html con la información necesaria para representar la galería..
        - Dificultad:Integrar los datos en el contexto del template fue sencillo debido a la estructura previa de services.py al igual de implementar la lista de favorito del usuario, la implementación de los índices nos llevo un tiempo poder hacerlo ya que no sabíamos por dónde empezar
        - Decisiones tomadas:Mantener una lista vacía para los favoritos si esta funcionalidad no estaba desarrollada, asegurando que el código no falle si no se implementa esta parte.

- **search**
    - Esta función es la que permite realizar búsquedas de personajes por nombres y divide los resultados en índices.
        - Dificultad: Teníamos problemas a la hora de buscar más de 20 personajes por la forma en la que está hecha la api. Al solucionarlo surgieron problemas con los índices, al navegar estos no guardaban las búsquedas.
        - Decisiones tomadas: A partir del problema de la cantidad de resultados tuvimos que cambiar la forma en la que se recopilan. En lo que respecta al problema de los índices, tuvimos que agregar variables en la función que los crea para así mostrar adecuadamente los personajes que correspondan a la búsqueda

- **getAllFavouritesByUser, saveFavourite, deleteFavourite**
    - A estas funciones les dio uso llamando a las mismas funciones que ya están desarrolladas en services.py, pero con la condición de que se debe estar logueado para por usarlas
        - Dificultad: No hubo muchos problemas a la hora de completar las funciones..
        - Decisiones tomadas:utilizamos la services.py para poder completar las funciones y pasarles los datos correspondiente a cada función que de services.py llamandola con su correspondiente función.

- **exit**
    - Se le agregó la función de un logout.
        - Dificultad: No hubo complicaciones
        - Decisiones tomadas: Utilizamos funciones ya brindadas para cumplir esta tarea

Cambios en urls.py

- **registro y login**
  - Se añadieron direcciones para ir al registro y al inicio de sesión.
    - dificultades: En este punto hemos tenido dificultades de errores que no entendiamos por que eran.
    - Decisiones tomadas: Hemos decididos para arreglar los errores que teníamos, decidimos cambiar cierta parte de un path que es “views.index_page” por “views.login_view” y también agregamos un path nuevo para registro.

Cambios en home.html

- **Templates**
  - Se les agregó un borde redondeado que indica el estado del personaje, un hover que indica cuando el mouse pasa por encima y una sombra alrededor. Al botón de añadir a favorito se le agregó un cambio de color ligero al seleccionarlo y un efecto de elevacion.
    - Dificultades: En este punto no hemos tenido problema
    - Deciciones tomadas: hemos aplicado un css para que los template tenga una pequeña animación y decoración , también hemos agregado bordes de colores verdes(vivo), naranja(desconocido) y rojo(muerto) que nos indica el estado del personaje.
- **Índices**
  - Se crearon índices para navegar entre páginas y también a la hora de hacer búsquedas, incluye flechas para moverse entre grupos de páginas.

- **Pantalla de carga**
  - Se creó una animación de spinner de carga con un background transparente mientras se espera los resultados de la búsqueda y al cambiar de página.
    - Dificultades: las dificultades que tuvimos en este punto fue que el spinner de carga pudiera girar e implementar el fondo y cuando cambiamos de índices apareciera.
    - Decisiones tomadas: hemos tomado la decisión de usar javascript para crear un script mostrando el spinner mientra que el usuario espera a que carguen las imágenes carguen para darle hacerlo girar al spinner hemos decidido implementar css para poder hacer girar el spinner.

Cambios en header.html

- **Registro**
  - Se agregó la opción para acceder al registro.

        - Dificultades:no hemos tenido ninguna dificultad a la hora de implementar el registro en el header
        - Decisiones:

- **Estilo**
    - Se añadió una barra azul para que los elementos no aparezcan flotando en la nada.
        - Dificultades: No hemos ninguna dificultad a la hora de implementar los estilo en el header
        - Decisiones tomadas: hemos decidido implementar Bootstrap para la barra en el header.

Se creo forms.py y register.html

- Son necesarios para poder lograr el que el usuario registre una cuenta nueva.
    - Dificultades:A la hora de implementar el archivo register.html no hemos tenido problema cuando hemos creado forms.py e implementar las funciones del código en el archivo de views.py para la función registro hemos tenido problemas de implementación.
    - Decisiones tomadas: para poder resolver los errores que teníamos con forms a la hora de implementarlo hemos decicido en llamarlo con un froms.

Conclusión:

Este proyecto nos ayudó a comprender mejor cómo se trabaja por fuera de las prácticas habituales de las clases en las que solo se utilizan códigos aislados, y empezar a trabajar con repositorios en github para facilitar el envío de versiones del proyecto. Tuvimos que dedicar tiempo simplemente al entendimiento del código ya brindado y también la corrección de nuestros errores que surgían durante la implementación del nuevo código.

Códigos:

- **services.py (getApiInfo)**

```
def getApiInfo(): #permite acceder a la informacion de la api
    response = requests.get(config.DEFAULT_REST_API_URL).json()
    return response['info']
```

- **services.py (saveFavourites)**

```
def saveFavourite(request):
    fav = translator.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.
```

- **services.py (getAllFavorites)**

```
def getAllFavourites(request):
    if not request.user.is_authenticated:
        return "usuario no resgistrado"
    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
```

- **services.py (deleteFavourites)**

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

- **views(exit)**

```
@login_required
def exit(request):
    logout(request)
    return redirect('home')
```

- **transport.py(getAllImages)**

```
def getAllImages(input=None):
    if input is None:
        json_response = requests.get(config.DEFAULT_REST_API_URL).json()
    elif input.startswith('?page='):
        # Si el input es una página, usar la URL directamente con el número de página
        page_number = input.replace('?page=', '')
        json_response = requests.get(f'https://rickandmortyapi.com/api/character?page={page_number}').json()
    else:
        json_response = requests.get(config.DEFAULT_REST_API_SEARCH + input).json()

    json_collection = []

    # si la búsqueda no arroja resultados, entonces retornamos una lista vacía de elementos.
    if 'error' in json_response:
        print("[transport.py]: la búsqueda no arrojó resultados.")
        return json_collection

    for object in json_response['results']:
        try:
            if 'image' in object:  # verificar si la clave 'image' está presente en el objeto (sin 'image' NO nos sirve, ya que no mostrará las imágenes).
                json_collection.append(object)
```

- **views.py (registro)**

```python
def registro(request):                  #recibe como parametro un request que contiene toda la informacion sobra la solicitud
    if request.method == 'POST':        #verifica si el metodo de la silicitud es POST
        form = RegistroForm(request.POST)#se crea una
        if form.is_valid():             #Revisa que los datos ingresados sean validos
            username = form.cleaned_data.get('username')#
            if User.objects.filter(username=username).exists(): #Revisa que el usuario no exista
                messages.error(request, "El nombre de usuario ya está en uso.")#tira un mensaje de error si el nombre de usuario ya esta registrado
            else:
                user = form.save(commit=False)  #Crea el usuario
                user.set_password(form.cleaned_data.get('password')) #Guarda la contraseña de cada usuario
                user.is_active = True  # Activar el usuario inmediatamente
                user.save()#guarda los nuevos usuario en la base de datos

                messages.success(request, "Registro exitoso. Ahora puedes iniciar sesión.")#muestra un mensaje en la parte de login si el registro del usuario fue exitoso
                return redirect('login')
    else:
        form = RegistroForm()
    return render(request, 'registro.html', {'form': form})
```

- **views.py(Login_view)**

```python
def login_view(request):
    if request.method == 'POST':#verifica si el metodo de la solicitud es POST
                                #(es importante por que los datos del formulario se envian mediante una solicitus POST)
        username = request.POST.get('username')  # Guarda el usuario
        password = request.POST.get('password')  # Guarda la contraseña

        # Autenticación del usuario
        user = authenticate(request, username=username, password=password)#verifica las credenciales del usuario
        if user is not None:
            login(request, user)  # Inicia la sesión
            messages.success(request, 'Sesión iniciada correctamente.')
            return redirect('home')  # Redirige a la página principal
```

- **views.py(calcular_paginacion)**

```python
def calcular_paginacion(total_items, pagina_actual, items_por_pagina=20, paginas_por_grupo=10):

    total_paginas = (total_items + items_por_pagina - 1) // items_por_pagina #calcual cuantos indices se necesitaran, se suma los items de pagina menos 1 para redondear hacia arriba
    #sirve para no pasarse de los limites de los indices
    if pagina_actual < 1:
        pagina_actual = 1
    elif pagina_actual > total_paginas:
        pagina_actual = total_paginas

    grupo_actual = (pagina_actual - 1) // paginas_por_grupo #divide los indices por grupos
    inicio = grupo_actual * paginas_por_grupo + 1 #calcula cual seria el primer indice de cada grupo
    fin = min(inicio + paginas_por_grupo - 1, total_paginas) #calcula cual seria el ultimo indice del grupo
    paginas_mostrar = range(inicio, fin + 1) if total_paginas > 0 else [] #son la lista de paginas que se muestran en los indices
    inicio_slice = (pagina_actual - 1) * items_por_pagina #indica cual es el primer item a mostrar segun el indice
    fin_slice = min(inicio_slice + items_por_pagina, total_items) #indica cual es el ultimo item a mostrar segun el indice

    return {
        'paginas_mostrar': paginas_mostrar,
        'pagina_actual': pagina_actual,
        'total_paginas': total_paginas,
        'grupo_inicio': inicio,
        'grupo_fin': fin,
        'inicio_slice': inicio_slice,
        'fin_slice': fin_slice
    }
```

- **views.py (home)**

```python
def home(request):
    #lleva a la pagina 1
    try:
        pagina = int(request.GET.get('page', 1))
    except ValueError:
        pagina = 1
    # obtiene información de la API
    api_info = services.getApiInfo()
    total_paginas = api_info['pages']

    paginacion = calcular_paginacion(total_paginas * 20, pagina) #crea los indices

    list_images = services.getAllImages(f"?page={pagina}")  # obtiene imagenes de la API para la página actual
    favourite_list = services.getAllFavourites(request)

    contexto = {
        'images': list_images,
        'favourite_list': favourite_list,
        'paginas': paginacion['paginas_mostrar'],
        'pagina_actual': paginacion['pagina_actual'],
        'total_paginas': paginacion['total_paginas'],
        'grupo_inicio': paginacion['grupo_inicio'],
        'grupo_fin': paginacion['grupo_fin']
    }

    return render(request, 'home.html',contexto)
```

- **views.py(search)**

```python
def search(request):
    search_msg = request.POST.get('query') or request.GET.get('query', '')
    try:
        pagina = int(request.GET.get('page', 1))
    except ValueError:
        pagina = 1

    if not search_msg:
        return redirect('home')

    all_matching_images = []
    current_page = 1
    #recorre toda la api buscando coincidencias de nombre y las agrega a una lista
    while True:
        input = f"?page={current_page}&name={search_msg}"
        try:
            images = services.getAllImages(input)
        except Exception as e:
            print(f"Error al obtener imágenes: {e}")
            break

        if not images:
            break

        all_matching_images.extend(images) #se utiliza extend ya que es similar a append pero con varios elementos
        current_page += 1

    total_items = len(all_matching_images)

    paginacion = calcular_paginacion(total_items, pagina) #se crean indices para las busquedas

    images_page = all_matching_images[paginacion['inicio_slice']:paginacion['fin_slice']] #crea una sublista con inicio y fin en las imagenes correspondientes al indice

    contexto = {
        'images': images_page,
        'paginas': paginacion['paginas_mostrar'],
        'pagina_actual': paginacion['pagina_actual'],
        'total_paginas': paginacion['total_paginas'],
        'grupo_inicio': paginacion['grupo_inicio'],
        'grupo_fin': paginacion['grupo_fin'],
        'is_search': True,
        'search_term': search_msg,
        'query': search_msg
    }

    return render(request, 'home.html', contexto)
```

- **home.html(spinner)**

```html
<!-- Texto de carga -->
<div id="loading-spinner" style="visibility: hidden; position: fixed; top: 0; left: 0; width: 100%; height: 100%; background-color: rgba(255, 255, 255, 0.8); z-index: 9999
    <div class="spinner"></div>
</div>

<script>
    document.addEventListener('DOMContentLoaded', function () {
        const spinner = document.getElementById('loading-spinner'); // Referencia al spinner
        const form = document.getElementById('search-form'); // Referencia al formulario
        const pagination = document.getElementById('pagination'); // Referencia al paginador

        // Mostrar el spinner al enviar el formulario
        if (form) {
            form.addEventListener('submit', function () {
                spinner.style.visibility = 'visible';
            });
        }

        // Mostrar el spinner al hacer clic en un enlace del paginador
        if (pagination) {
            pagination.addEventListener('click', function (event) {
                const target = event.target;

                // Verifica si el clic fue en un enlace (<a>)
                if (target.tagName === 'A') {
                    spinner.style.visibility = 'visible'; // Muestra el spinner
                }
            });
        }
    });
</script>
```

- **styles.css(spinner)**

```css
.spinner {
    border: 4px solid rgba(255, 255, 255, 0.3); /* Fondo del círculo */
    border-top: 4px solid #007bff; /* Color del borde superior */
    border-radius: 50%; /* Hacerlo circular */
    width: 5rem; /* Tamaño del spinner */
    height: 5rem; /* Tamaño del spinner */
    animation: spin 1.5s linear infinite; /* Animación de giro */
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%); /* Centrado absoluto */
}

/* Definición de la animación de giro */
@keyframes spin {
    0% {
        transform: translate(-50%, -50%) rotate(0deg);
    }
    100% {
        transform: translate(-50%, -50%) rotate(360deg);
    }
}
```