

Trabajo Práctico Final – Buscador Rick & Morty

Introducción a la Programación – 2º Semestre 2024

Integrantes

- Nicolás Toledo
- Ulises Gómez
- Ignacio Funes
- Zoe Jaimez

Introducción

Este proyecto es una aplicación web que permite a los usuarios explorar y buscar personajes de la serie de televisión Rick & Morty. Los usuarios pueden ver tarjetas de personajes que muestran sus fotos, estado actual, última ubicación conocida y capítulo de primera aparición en la serie. Además, luego de crear una cuenta, los usuarios pueden guardar sus personajes favoritos y añadir comentarios personales sobre ellos.

La aplicación está compuesta de 6 capas, a saber:

1. Capa de transporte
2. Capa de servicio o “lógica de negocio”
3. Capa de persistencia de datos
4. Capa de utilidades
5. Capa de vista
6. Capa de presentación

Cada una de ellas cumple una función específica y bien definida, dejando una clara separación de responsabilidades y facilitando el mantenimiento y la escalabilidad del código si fuera necesario en un futuro.

1. Capa de transporte

Implementada en *transport.py*, es el punto de entrada de datos externos a nuestra aplicación. Su función principal es establecer la comunicación con la API de Rick & Morty para obtener la información de los personajes.

Contiene la función *getAllImages*, que recibe dos parámetros: *input* (opcional, para filtrar personajes por nombre) y *page* (para la paginación de resultados). La función construye la URL de la petición incluyendo estos parámetros y realiza la solicitud HTTP a la API.

Al recibir la respuesta, extrae dos elementos fundamentales:

- Los personajes (*characters*)
- El total de páginas (*total_pages*)

Además, verifica que cada personaje tenga una imagen asociada, descartando aquellos que no la tengan.

La capa contempla posibles errores a la hora de realizar la solicitud, retornando listas vacías en caso de fallo. Finalmente, devuelve la lista de personajes y el número total de páginas, que serán procesados por las siguientes capas de la aplicación.

```
transport.py

import requests
from ...config import config

def getAllImages(input, page):

    url = config.BASE_URL
    params = {'page': page}
    if input:
        params['name'] = input

    try:
        response = requests.get(url, params=params)
        response.raise_for_status()
        data = response.json()
        characters = data.get("results", [])
        total_pages = data.get("info", {}).get("pages", 1)
    except requests.exceptions.RequestException as e:
        print(f"Error al hacer la solicitud: {e}")
        return [], 1

    images = []

    for object in characters:
        try:
            if 'image' in object:
                images.append(object)
            else:
                print("[transport.py]: se encontró un objeto sin clave 'image', omitiendo...")
        except KeyError:
            pass

    return images, total_pages
```

2. Capa de servicio o "lógica de negocio"

Implementada en *services.py*, actúa como el "cerebro" de la aplicación, coordinando el flujo de datos entre las distintas capas y aplicando la "lógica de negocio" – las reglas y los procesos que definen cómo debe funcionar la aplicación.

Sus principales funciones son:

getAllImages:

- Recibe los parámetros de búsqueda y página, validándolos
- Obtiene los datos "crudos" desde la capa de transporte
- Transforma cada personaje en una tarjeta mediante el translator
- Retorna la lista de tarjetas y el total de páginas
- Si los hay, informa de errores en cada proceso de la función

```
services.py

def getAllImages(input=None, page=config.DEFAULT_PAGE):

    if page < 1:
        page = config.DEFAULT_PAGE

    if input is not None:
        input = input.strip()

    try:
        characters, total_pages = transport.getAllImages(input, page)
        imageCards = []

        for character in characters:
            try:
                characterCard = translator.fromRequestIntoCard(character)
                imageCards.append(characterCard)
            except KeyError as e:
                print(f"[services.py]: Error al procesar datos del personaje: {e}")
                continue

        return imageCards, total_pages

    except Exception as e:
        print(f"[services.py]: Error al obtener datos de la API: {e}")
        return [], 0
```

get_home_context:

- Prepara todo el contexto necesario para la página principal
- Obtiene las imágenes filtradas y el total de páginas, mediante getAllImages
- Calcula el rango de páginas a mostrar
- Obtiene la lista de favoritos del usuario mediante la función getAllFavourites
- Construye y devuelve el contexto completo, desde las imágenes a mostrar hasta la búsqueda realizada si la hay

```
services.py

def get_home_context(request, input_name, current_page):
    images, total_pages = getAllImages(input_name, current_page)
    page_range = range(1, min(total_pages, 10) + 1)

    favourite_list = getAllFavourites(request)
    favourite_names = [fav.name for fav in favourite_list]

    return {
        'images': images,
        'favourite_list': favourite_names,
        'current_page': current_page,
        'total_pages': total_pages,
        'page_range': page_range,
        'name': input_name,
    }
```

register:

- Maneja el registro de nuevos usuarios
- Valida que el nombre de usuario no exista
- Crea el nuevo usuario en la base de datos
- Envía email de confirmación con credenciales
- Devuelve mensajes de éxito o error

```
services.py

def register(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        email = request.POST['email']
        first_name = request.POST['first_name']
        last_name = request.POST['last_name']

        if User.objects.filter(username=username).exists():
            messages.error(request, 'El nombre de usuario ya está en uso.')
        else:
            user = User.objects.create_user(
                username=username,
                password=password,
                email=email,
                first_name=first_name,
                last_name=last_name
            )
            user.save()
            send_mail(
                'Registro Exitoso',
                f'Tus credenciales de acceso son:\nUsuario: {username}\nContraseña: {password}',
                'noreply@tuapp.com',
                [email],
                fail_silently=False,
            )
            return messages.success(request, 'Usuario registrado exitosamente. Revisa tu correo electrónico para tus credenciales.')
    messages.error(request, 'Error al registrar.')
```

2. 1. Gestión de Favoritos

La aplicación permite a los usuarios autenticados guardar y gestionar sus personajes favoritos, los cuales se almacenan en la base de datos de la misma. Este sistema se implementa a través de varias funciones en la capa de servicio:

getAllFavourites:

- Verifica que el usuario esté autenticado (si no lo está, devuelve una lista vacía)
- Obtiene los favoritos desde la base de datos
- Transforma cada favorito en una tarjeta de personaje
- Devuelve la lista completa de favoritos del usuario

```
services.py

def getAllFavourites(request):
    if not request.user.is_authenticated:
        return []
    else:
        user = get_user(request)

        favourite_list = repositories.getAllFavourites(user)
        mapped_favourites = []

        for favourite in favourite_list:
            card = translator.fromRepositoryIntoCard(favourite)
            mapped_favourites.append(card)
        return mapped_favourites
```

saveFavourite:

- Recibe los datos del personaje
- Crea una nueva tarjeta de personaje con estos datos
- Asocia el usuario actual como propietario del favorito
- Delega a la capa de persistencia el guardado en la base de datos

```
services.py

def saveFavourite(request):
    fav = translator.fromTemplateIntoCard(request)
    fav.user = get_user(request)
    return repositories.saveFavourite(fav)
```

deleteFavourite:

- Recibe el ID del favorito a eliminar
- Solicita a la capa de persistencia la eliminación en la base de datos
- Devuelve el resultado de la operación

```
services.py

def deleteFavourite(request):
    favId = request.POST.get('id')
    return repositories.deleteFavourite(favId)
```

3. Capa de persistencia de datos

Implementada en *repositories.py*, se encarga del almacenamiento y recuperación de datos en la base de datos SQLite, manejando específicamente las operaciones relacionadas con los favoritos de los usuarios. Se compone de las funciones:

saveFavourite:

- Recibe una tarjeta de personaje
- Crea un nuevo registro en la tabla Favourite
- Almacena todos los datos del personaje (imagen, nombre, estado, etc)
- Asocia el registro con el usuario correspondiente
- Devuelve el favorito creado o, en caso de error, nada

```

persistence/repositories.py

def saveFavourite(image):
    try:
        fav = Favourite.objects.create(url=image.url, name=image.name, status=image.status,
        last_location=image.last_location, first_seen=image.first_seen, user=image.user)
        return fav
    except Exception as e:
        print(f"Error al guardar el favorito: {e}")
        return None
```

getAllFavourites:

- Recibe un usuario como parámetro
- Consulta la base de datos filtrando por el usuario específico
- Obtiene todos los datos de cada favorito que tiene dicho usuario
- Devuelve la lista de favoritos

```

persistence/repositories.py

def getAllFavourites(user):
    favouriteList = Favourite.objects.filter(user=user).values('id', 'url', 'name', 'status', 'last_location', 'first_seen')
    return list(favouriteList)
```

deleteFavourite:

- Recibe el ID del favorito a eliminar
- Busca el registro correspondiente en la base de datos
- Elimina el favorito si existe
- Devuelve si la operación fue exitosa o no

```

persistence/repositories.py

def deleteFavourite(id):
    try:
        favourite = Favourite.objects.get(id=id)
        favourite.delete()
        return True
    except Favourite.DoesNotExist:
        print(f"El favorito con ID {id} no existe.")
        return False
    except Exception as e:
        print(f"Error al eliminar el favorito: {e}")
        return False
```

4. Capa de utilidades

Ubicada en la carpeta *utilities*, contiene herramientas y estructuras fundamentales para el funcionamiento de la aplicación, dividida en dos componentes principales:

4.1. card.py:

- Define la estructura base de una tarjeta de personaje
- Contiene todos los campos necesarios: URL de imagen, nombre, estado, última ubicación y primer episodio
- Implementa métodos de comparación para verificar si dos tarjetas son iguales
- Permite asociar un usuario y un ID para el manejo de favoritos

```
utilities/card.py

class Card:
    def __init__(self, url, name, status, last_location, first_seen, user=None, id=None):
        self.url = url
        self.name = name
        self.status = status
        self.last_location = last_location
        self.first_seen = first_seen

        self.user = user
        self.id = id

    def __str__(self):
        return f'IMG URL: {self.url}, name: {self.name}, status: {self.status}, última ubicación: {self.last_location}, primera vez visto: {self.first_seen}, Usuario: {self.user}, Id: {self.id}'

    def __eq__(self, other):
        if not isinstance(other, Card):
            return False
        return (self.url, self.name, self.status) == \
            (other.url, other.name, other.status)

    def __hash__(self):
        return hash((self.url, self.name, self.status))
```

4.2. translator.py:

Actúa como puente entre las diferentes representaciones de datos en la aplicación, implementando tres funciones de conversión:

- fromRequestIntoCard: convierte los datos de la API en una tarjeta de personaje

```
utilities/translator.py

def fromRequestIntoCard(object):
    card = Card(
        url=object['image'],
        name=object['name'],
        status=object['status'],
        last_location = object['location']['name'],
        first_seen = object['origin']['name']
    )
    return card
```

- fromTemplateIntoCard: transforma los datos de un favorito en una tarjeta de personaje, antes de guardarla en la base de datos

```
utilities/translator.py

def fromTemplateIntoCard(templ):
    card = Card(
        url=templ.POST.get("url"),
        name=templ.POST.get("name"),
        status=templ.POST.get("status"),
        last_location=templ.POST.get("last_location"),
        first_seen=templ.POST.get("first_seen")
    )
    return card
```

- fromRepositoryIntoCard: convierte los datos de la base de datos en una tarjeta de personaje

```
utilities/translator.py

def fromRepositoryIntoCard(repo_dict):
    card = Card(
        id=repo_dict['id'],

        url=repo_dict['url'],
        name=repo_dict['name'],
        status=repo_dict['status'],
        last_location=repo_dict['last_location'],
        first_seen=repo_dict['first_seen'],
    )
    return card
```

De esta manera, se garantiza la consistencia en el manejo de datos a través de toda la aplicación, proporcionando una estructura común y formas de convertir los datos entre diferentes formatos.

5. Capa de vistas

Esta capa se encuentra en *views.py* y actúa como intermediaria entre las peticiones del usuario y la capa de servicios de la aplicación. Se encarga de recibir las solicitudes HTTP, procesarlas recurriendo a la capa de servicios si así se requiere y mostrar las diferentes páginas de la aplicación con los datos necesarios.

Sus funciones principales son:

index_page:

- Renderiza la página de inicio de la aplicación
- Muestra el template `index.html`

```
views.py

def index_page(request):
    return render(request, 'index.html')
```

home:

- Obtiene los parámetros de búsqueda y página actual
- Obtiene el “contexto” de la página (imágenes, favoritos, etc)
- Si no ocurre un error, renderiza el archivo `home.html` con los datos obtenidos

```
views.py

def home(request):
    input_name = request.GET.get('name', '')
    current_page = int(request.GET.get('page', 1))

    try:
        context = services.get_home_context(request, input_name, current_page)
        return render(request, 'home.html', context)
    except Exception as e:
        messages.error(request, f"Error cargando datos de la página: {e}")
        return redirect('index')
```

register_view:

- Recibe los datos del formulario de registro
- Delega el procesamiento a la capa de servicios
- Muestra la página de registro con el resultado de la operación

```
views.py

def register_view(request):
    services.register(request)
    return render(request, 'registration/register.html')
```

5.1. Funciones con autenticación de usuario

Algunas funcionalidades sólo están disponibles a usuarios previamente registrados, requiriendo que hayan iniciado sesión:

getAllFavouritesByUser:

- Obtiene la lista de favoritos del usuario
- Renderiza la página favourites.html con los favoritos correspondientes

```
views.py

@login_required
def getAllFavouritesByUser(request):
    favourite_list = services.getAllFavourites(request)
    return render(request, 'favourites.html', {'favourite_list': favourite_list})
```

saveFavourite:

- Procesa la solicitud de guardar un nuevo favorito
- Redirecciona a la página principal

```
views.py

@login_required
def saveFavourite(request):
    services.saveFavourite(request)
    return redirect('home')
```

deleteFavourite:

- Elimina el favorito seleccionado
- Redirecciona a la página de favoritos

```
views.py

@login_required
def deleteFavourite(request):
    services.deleteFavourite(request)
    return redirect('favoritos')
```

exit:

- Cierra la sesión activa del usuario en la aplicación
- Redirecciona a la página principal

```
views.py

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

6. Capa de presentación

Por último, en la carpeta *templates* se encuentran todas las páginas o “plantillas” que definen la interfaz visual de la aplicación, en forma de archivos HTML, a través de las cuales la información procesada por el resto de las capas se presenta en el navegador del usuario.

Esta capa utiliza Bootstrap para el diseño y estilizado, garantizando una experiencia de usuario consistente y un diseño responsivo en las variadas resoluciones.

Se compone de:

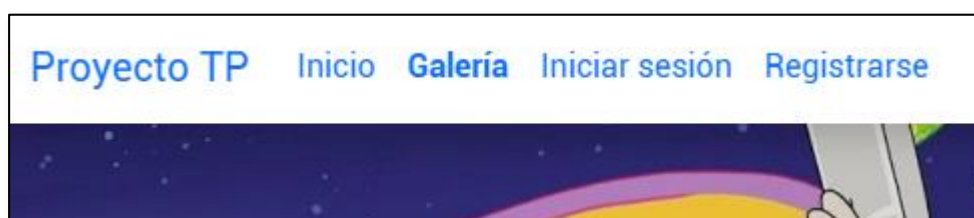
index.html:

- Actúa como página de bienvenida
- Muestra el logo de Rick & Morty
- Presenta el nombre del usuario si está autenticado
- Proporciona enlace directo a la galería de personajes



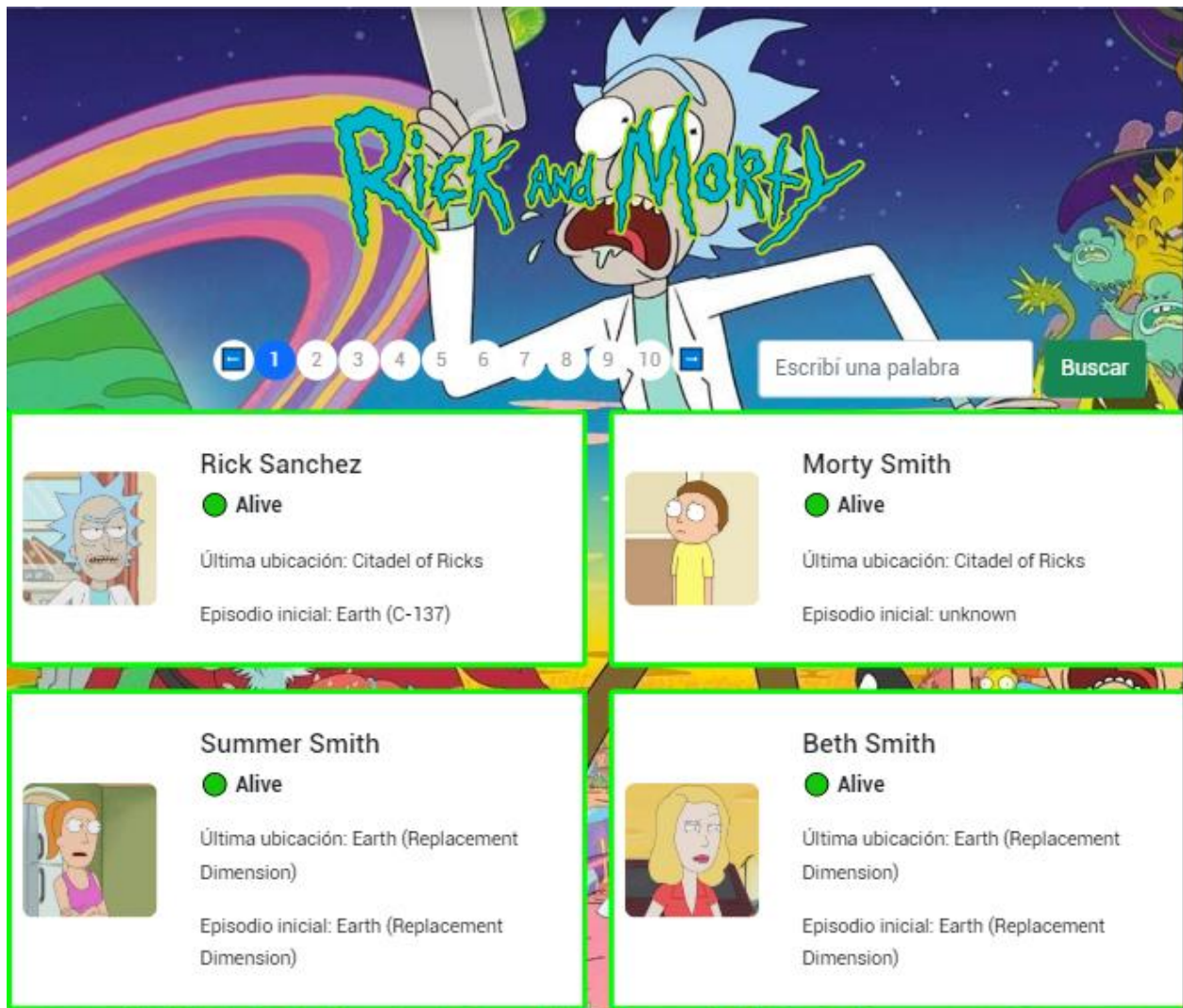
header.html:

- Contiene la barra de navegación principal
- Muestra enlaces diferentes según el estado de autenticación del usuario
- Proporciona acceso a la galería, favoritos, inicio de sesión y registro
- Su forma se adapta según la resolución, colapsándose si es necesario



home.html:

- Muestra la galería principal de personajes
- Implementa el buscador y la paginación de resultados
- Presenta cada personaje en una tarjeta con su imagen, nombre, estado, última ubicación y episodio inicial
- Permite marcar personajes como favoritos para usuarios autenticados
- El borde cambia según el estado del personaje (vivo, muerto o desconocido)



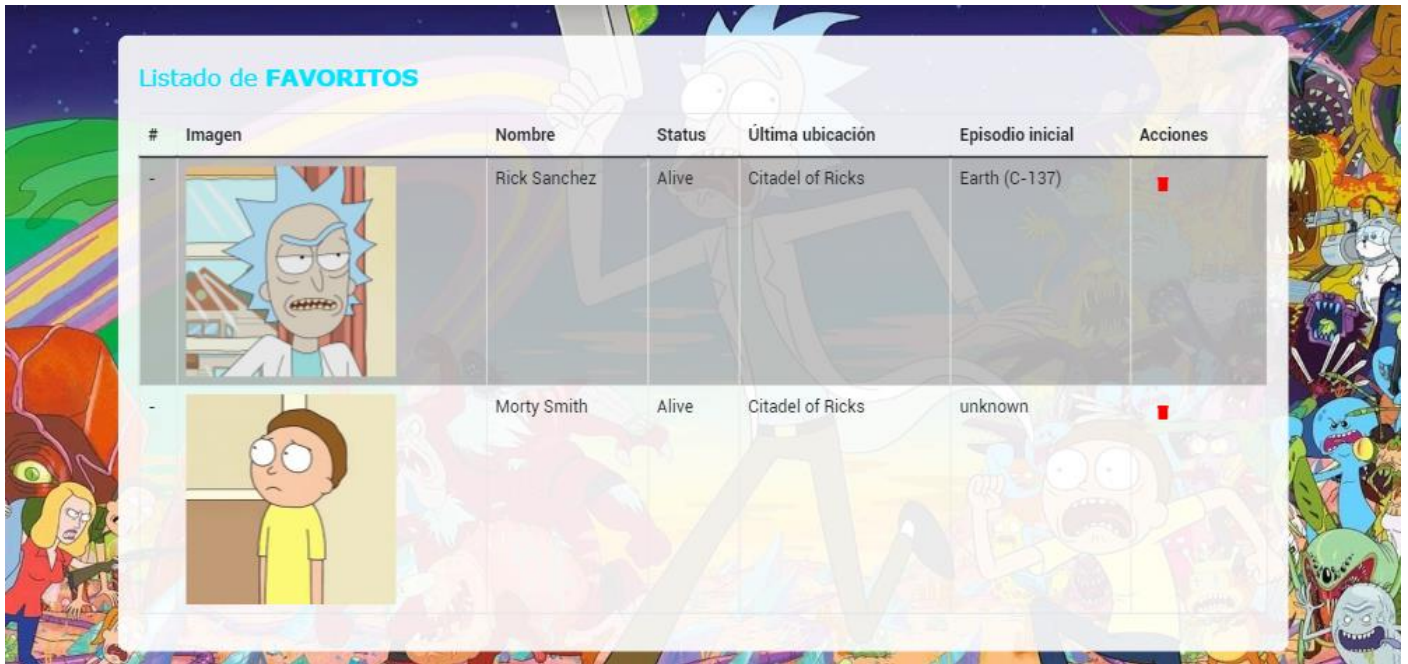
footer.html:





- Se mantiene fijo en la parte inferior de todas las páginas
- Muestra información de la universidad
- Incluye la versión actual de la aplicación
- Utiliza un fondo semi-transparente para mejor visibilidad



favourites.html:

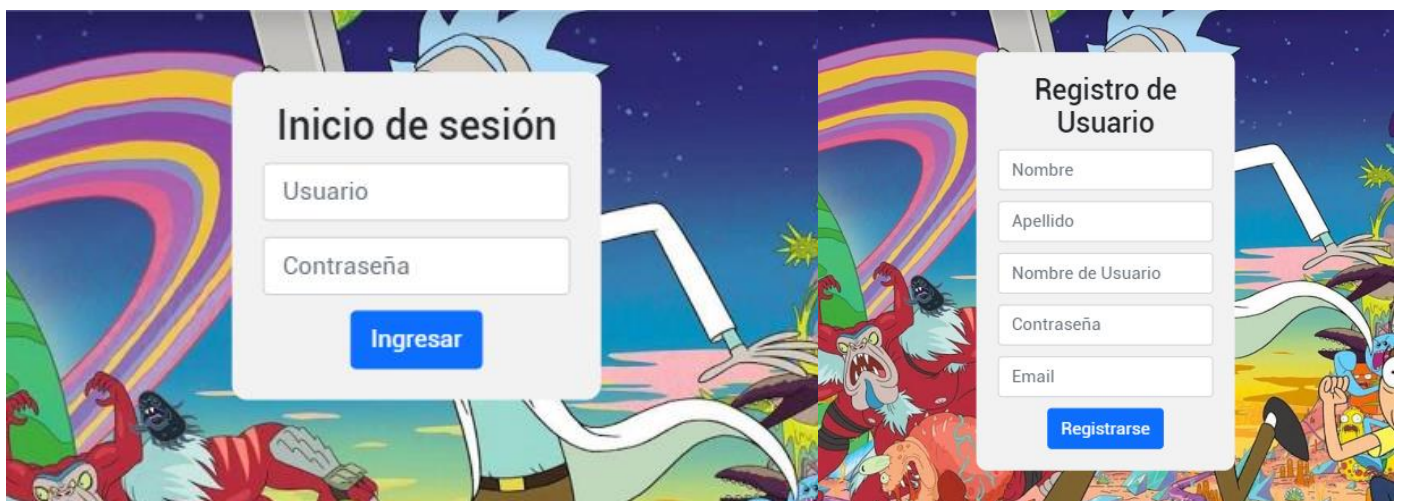
- Muestra la lista de personajes favoritos del usuario
- Permite eliminar personajes de favoritos
- Solo accesible para usuarios autenticados



#	Imagen	Nombre	Status	Última ubicación	Episodio inicial	Acciones
-		Rick Sanchez	Alive	Citadel of Ricks	Earth (C-137)	
-		Morty Smith	Alive	Citadel of Ricks	unknown	

Páginas de autenticación (login.html y register.html):

- Presentan formularios centrados con diseño minimalista
- Manejan la entrada de datos del usuario:
 - Login: usuario y contraseña
 - Registro: nombre, apellido, usuario, contraseña y email
- Muestran mensajes informativos sobre el resultado de las operaciones



Inicio de sesión

Registro de Usuario

En conclusión, nuestras implementaciones resultaron en una aplicación funcional que demuestra conceptos fundamentales de programación y desarrollo web.

División de tareas

Desde el comienzo del proyecto, cada integrante fue asignado tareas específicas, procurando la mayor productividad del grupo y la mejor organización al realizarlas.

El desarrollo de la aplicación fue dividido en:

Nicolás Toledo: Estructura del proyecto y configuración básica

- Supervisión de la estructura de la aplicación, asegurando que las capas y archivos estén correctamente organizados.
- Creación de repositorio en GitHub, organización del README, realización de primeros commits de estructura y organización de metodología de trabajo colaborativo.
- Redacción de informe final, incluyendo el código de las funciones implementadas, explicaciones, y decisiones de diseño.

Ulises Gómez: Implementación de registro y permisos

- Configuración de lógica de autenticación básica (registro de usuario, inicio de sesión y cierre de sesión).
- Supervisión de uso correcto de permisos para que sólo usuarios autenticados pueden agregar favoritos.
- Colaboración en el desarrollo de funciones de favoritos

Zoe Jaimez: Estilos de la aplicación

- Configuración de cambio de color de borde de las tarjetas según el estado del personaje (vivo, muerto, desconocido).
- Revisión y personalización de estilos usando Bootstrap y CSS para mejorar la interfaz de la aplicación, en todas las páginas

Ignacio Funes: Paginación y favoritos

- Desarrollo de paginación de resultados
- Desarrollo de la lógica de la página principal para la gestión de favoritos.
- Creación de lógica para agregar/eliminar personajes en favoritos
- Colaboración con el diseño de la sección de favoritos