

## TRABAJO PRÁCTICO

### Buscador Rick & Morty

*Introducción a la Programación*

*Segundo Semestre, 2024*

**Integrantes:** Gómez Micaela ([micaelagomez.1iaa@gmail.com](mailto:micaelagomez.1iaa@gmail.com)); González Lautaro ([lautarogonzalez421@gmail.com](mailto:lautarogonzalez421@gmail.com)); Oliva, Nadia ([nadia.santinonchu23@gmail.com](mailto:nadia.santinonchu23@gmail.com)) y Zayas, Lucía Belén, ([luzayas@abc.gob.ar](mailto:luzayas@abc.gob.ar))

**Resumen:** En el presente trabajo práctico, se implementó una aplicación web utilizando Django (framework de desarrollo escrito en Python) que permite buscar imágenes de los personajes de la serie Rick & Morty. Para eso, buscará su API (conjunto de funciones y procedimientos que integra sistemas y permite que sus funciones puedan ser reutilizadas en apps y software). Cada información de la API fue renderizada por el framework en diferentes cards que mostrarán la imagen del personaje, estado, última ubicación y el episodio inicial que apareció en la serie.

También se desarrolló un buscador central y un módulo de autenticación básica (usuario y contraseña) para almacenar uno o más resultados como favoritos y que estará a disposición para la consulta. Con eso, la app guarda y verifica qué imágenes fueron seleccionadas.

### 1. Introducción

El trabajo, en primera instancia, consistía en completar y arreglar los fragmentos de códigos que estaban en blanco o pendientes de reedición para que la página web permitiese buscar imágenes de los personajes, además de un pequeño resumen sobre cada uno de ellos. Cada imagen renderizada estaba dispuesta en pequeñas *cards* que mostraban todas sus características principales. La app debía garantizar la autenticación del usuario y guardar cada personaje escogido en la carpeta de favoritos.

**2. Desarrollo** Este trabajo tenía tres fragmentos de código que necesitaban revisarse:

- a) **Views.py:** se agregó el layer transport, para que pudiera así agregar las imágenes de la API, los favoritos del usuario y así dibujar el correspondiente template. Todas estas imágenes fueron devueltas renderizadas. Además, en la función home(request), se le asigna a la variable images lo que devuelve la api a través del método transport.getAllImages().
- b) **services.py:** la función getAllImages() (recibiendo y pasando el parametro del tipo correspondiente) llama a transport.getAllImages(input) para obtener la información en formato json de la api. Una vez obtenido lo trata de convertir a formato Card y loguea lo que se obtuvo.
- c) **home.html:** se cambió el style del div que contiene a una Card para que, dependiendo del valor de status de cada personaje, pinte el borde del color correspondiente. Se cambió en el <img> que dibujaba la imagen de img.url a img.image para usar la url correcta y de igual forma con los <p><small> que mostraban la última ubicación y el Episodio inicial.

**Conclusión:** En resumen, lo que se hizo en este trabajo práctico, fue implementar algunas funcionalidades que no estaban dadas en el código de origen, para que la página web fuera funcional y acorde a las necesidades de los usuarios. Es una página web bastante rudimentaria, pero funcional.

**Anexo:** Enunciado (Enunciado del trabajo copiado textual)

#### Buscador Rick & Morty

- El trabajo consiste en implementar una aplicación web usando Django que permita buscar imágenes de los personajes de la serie Rick & Morty, usando su API homónima. La información que provenga de esta API será renderizada por el *framework* en distintas *cards* que mostrarán - como mínimo- la imagen del personaje, el estado, la última ubicación y el episodio inicial. Adicionalmente -y para enriquecerla- se prevee que los estudiantes desarrolle 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.
- Gran parte de la aplicación ya está resuelta: solo falta implementar las funcionalidades más importantes 😊 .



- A nivel de template, se cuenta con 5 HTMLs: header (cabecera de la página), footer (pie de página), home (sección donde se mostrarán las imágenes y el buscador), index (contener principal que incluye a los 3 HTMLs anteriores) y favourites (panel para mostrar los favoritos en caso de implementarlos). Para el caso del *header*, se implementó cierta lógica para determinar si un usuario está logueado (o no) y obtener así su nombre; para el caso del *home*, éste permite recorrer cada elemento de la API y dibujar su información en pantalla. El *footer* no posee acciones a nivel código relevantes para el desarrollo.
- A nivel de vistas, en el archivo `views.py` encontrarán algunas funciones semidesarrolladas: `index_page(request)` que renderiza el contenido de 'index.html'; `home(request)` que obtiene todas las imágenes mapeadas de la API -a través de la capa de servicio- y los favoritos del usuario, y muestra el contenido de 'home.html' pasándole dicha información.
- A nivel de lógica, se incluye el archivo `transport.py` completo con todo el código necesario para consumir la API. Además, se anexa un `translator.py` con la lógica necesaria para convertir/mapear los resultados en una Card (objeto que finalmente se utilizará en el template para dibujar los resultados).
- El proyecto está construido sobre una arquitectura multicapas, donde cada capa posee una única responsabilidad, a saber:
  - Persistence (empleada para el alta/baja/modificación (ABM) de objetos en una base de datos tipo fichero, llamada SQLite).

- Services (usada para la lógica de negocio de la aplicación).
- Transport (utilizada para el consumo de los datos de la API).
- Utilities (almacena *translators* y demás elementos propios de la aplicación, usados en los templates).

Si bien no es un parámetro de evaluación dónde colocan las funciones, es altamente recomendado que las funciones que se agreguen estén en las capas que correspondan (consultar con los docentes en caso de dudas).

### **¿Qué voy a ver al iniciar la app?**

- Al iniciar la aplicación y hacer clic sobre Galería, verás lo siguiente:



### **Lo que falta hacer (OBLIGATORIO)**

- Aún faltan implementar ciertas funciones de los módulos views.py - services.py y modificar el template home.html. Éstas son las encargadas de hacer que las imágenes de la galería se muestren:

- (1) views.py:

- *home(request)*: 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.

```
# esta función 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 = []
    favourite_list = []

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

- (2) services.py:

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

```

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

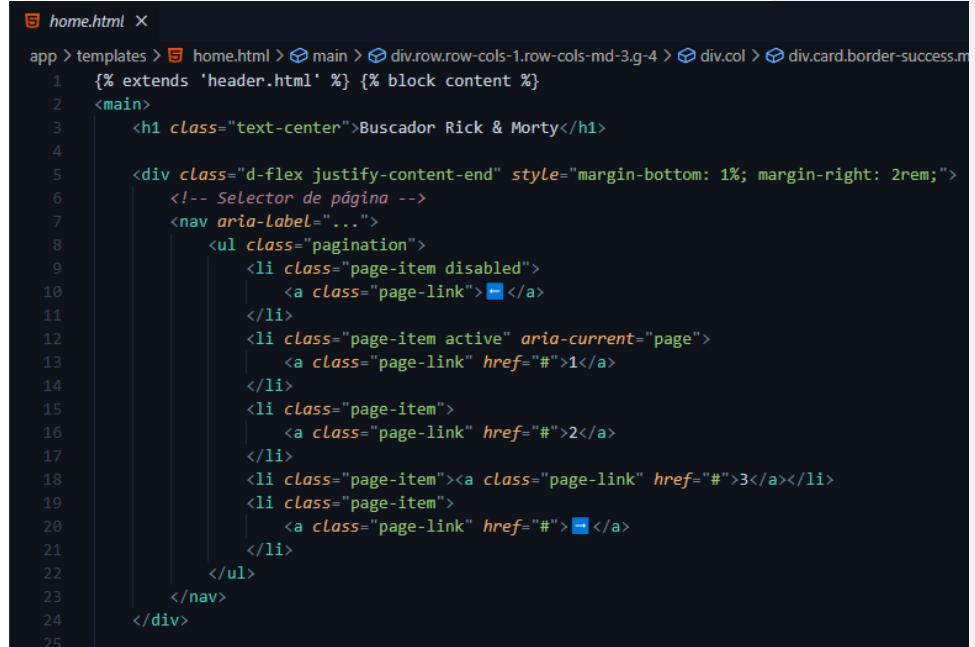
    # recorre cada dato crudo de la colección anterior, lo convierte en una Card y lo agrega a images.
    images = []

    return images

```

○ (3) 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. Se sugiere consultar la [documentación de Bootstrap sobre Cards](#) y [cómo generar condicionales en Django](#) para tener un mejor acercamiento a la solución.



```

home.html ×

app > templates > home.html > main > div.row.row-cols-1.row-cols-md-3.g-4 > div.col > div.card.border-success.m
1   {% extends 'header.html' %} {% block content %}
2     <main>
3       <h1 class="text-center">Buscador Rick & Morty</h1>
4
5       <div class="d-flex justify-content-end" style="margin-bottom: 1%; margin-right: 2rem;">
6         <!-- Selector de página -->
7         <nav aria-label="...">
8           <ul class="pagination">
9             <li class="page-item disabled">
10               <a class="page-link" href="#"> </a>
11             </li>
12             <li class="page-item active" aria-current="page">
13               <a class="page-link" href="#">1</a>
14             </li>
15             <li class="page-item">
16               <a class="page-link" href="#">2</a>
17             </li>
18             <li class="page-item"><a class="page-link" href="#">3</a></li>
19             <li class="page-item">
20               <a class="page-link" href="#"> </a>
21             </li>
22           </ul>
23         </nav>
24       </div>
25

```

Concluido su desarrollo, deberían ver algo como lo siguiente:

**Notar el borde que posee cada imagen (en este caso, verde, pues todos los personajes están vivos).**

Projeto TP Inicio Galeria Iniciar sesión

Buscador Rick & Morty

Escríb una palabra Buscar

1 2 3

Imagen	Nombre	Estado	Última ubicación	Episodio inicial
Rick Sanchez	Rick Sanchez	Alive	Última ubicación: Citadel of Ricks	Episodio inicial: Earth (C-137)
Morty Smith	Morty Smith	Alive	Última ubicación: Citadel of Ricks	Episodio inicial: unknown
Summer Smith	Summer Smith	Alive	Última ubicación: Earth (Replacement Dimension)	Episodio inicial: Earth (Replacement Dimension)
Beth Smith	Beth Smith	Alive	Última ubicación: Earth (Replacement Dimension)	Episodio inicial: Earth (Replacement Dimension)
Jerry Smith	Jerry Smith	Alive	Última ubicación: Earth (Replacement Dimension)	Episodio inicial: Earth (Replacement Dimension)
Abadango Cluster Princess	Abadango Cluster Princess	Alive	Última ubicación: Abadango	Episodio inicial: Abadango

### Condiciones de entrega

- Requisitos de aprobación y criterios de corrección
  - El TP debe realizarse en grupos de 2 o 3 integrantes (no 1). Para aprobar el trabajo se deberán reunir los siguientes ítems:
    - La galería de imágenes se muestra adecuadamente (imagen, nombre, estado, última ubicación y episodio inicial de cada personaje).
    - El código debe ser claro. Las variables y funciones deben tener nombres que hagan fácil de entender el código a quien lo lea -de ser necesario, incluir comentarios que clarifiquen-. Reutilizar el código mediante funciones todas las veces que se amerite.
    - No deben haber variables que no se usan, funciones que tomen parámetros que no necesitan, ciclos innecesarios, etc.
  - El 'correcto' funcionamiento del código NO es suficiente para la aprobación del TP, son necesarios todos los ítems anteriores.

---

### **Fecha de entrega**

El trabajo debe ser entregado en la fecha estipulada en el cronograma. Recordar que es requisito hacer pre-entregas.

### **Formato de entrega**

- La entrega se dividirá de 2 partes: código e informe:
  - Parte 1: código: todo el desarrollo debe estar en un repositorio interno del grupo (*fork* del repo base del TP). Se deben añadir a los docentes de la comisión con motivo de verificar los avances del mismo (corregir funciones, brindar sugerencias o

recomendaciones, etc). Dado el caudal de alumnos, serán responsables los estudiantes de notificar a los docentes para evaluar una pre-entrega, corregir alguna duda o similar que bloquee/impida del avance del TP.

Sugerimos:

- Que cada integrante tenga su propia cuenta de GitHub, NO usar una única en el proyecto.
- Cada integrante debe *commitear* una o varias porciones de código, dependiendo cómo distribuyan el trabajo. Se debe visualizar el aporte individual al TP.
- Parte 2: informe: deben redactar un documento donde exista una introducción que explique de qué se trata el trabajo (sin utilizar lenguaje técnico), que incluya el código de las funciones implementadas y una breve explicación de cada una de ellas junto con las dificultades de implementación y decisiones tomadas -con su correspondiente justificación-. NO incluir explicaciones de funcionalidades de Python, Django o similares. Este documento debe estar en formato PDF anexo dentro de la carpeta del TP.
- 🔥 Se DEBE cumplir con ambas partes (código + informe) para aprobar el trabajo práctico.