Trabajo Práctico: Buscador Rick & Morty

Nombres de los Estudiantes: Pereyra, Rocio Belen - Andrés Geiler - Ataun, Ignacio Ezequiel

Curso: Introducción a la Programación

Profesores: Franco Costantini, Daniel Bressky, Esteban Fassio

Fecha: 27/11/2024

Institución: Universidad Nacional de General Sarmiento

Introducción:

Este trabajo práctico tiene como objetivo el desarrollo de una página web interactiva sobre los personajes de la serie *Rick y Morty*, donde los usuarios podrán visualizar los personajes a través de “cartas” que indican su estado (vivo, muerto o desconocido). A través de la modificación y creación de diversas funciones, se busca que la página permita, entre otras cosas, buscar personajes, agregar a favoritos las cartas y realizar otras acciones para mejorar la experiencia de usuario, haciendo que la página sea completa y visualmente atractiva.

La implementación de las funcionalidades se realizó utilizando Visual Studio Code para escribir el código, Python para la lógica de la aplicación y Github para gestionar los cambios y compartir el progreso con los demás miembros del equipo. Además, el trabajo se centró en la colaboración entre los participantes para resolver problemas y tomar decisiones sobre las funcionalidades a implementar, destacando las dificultades encontradas en el proceso.

El sistema incluye diversas características, como la visualización, búsqueda y modificación de los personajes guardados como favoritos, lo que facilita una experiencia de usuario dinámica y fluida. Este documento detalla las funciones implementadas, las decisiones tomadas y los archivos clave, además de las dificultades superadas durante el desarrollo de la aplicación.

Desarrollo:

A continuación, se describen las funciones implementadas y los archivos involucrados:

1. **Función Home**:
    - En la función **home**, se recorre una lista de personajes de la serie obtenidos desde la API. Si el usuario está autenticado, se marca cuáles de estos personajes ya han sido añadidos a su lista de favoritos, brindando una visualización dinámica y personalizada de los personajes.
2. **Función Guardar Favorito**:

- Para permitir a los usuarios guardar personajes como favoritos, se ha creado la función **Guardar Favorito**. Esta función se encarga de marcar un personaje como favorito en la base de datos del usuario autenticado.
- **Modificaciones en archivos**:
  - **models.py**: Se creó un modelo Favourite para almacenar la información de los favoritos, con los campos user, url, name, status, last_location y first_seen. Estos campos permiten registrar los detalles del personaje añadido como favorito.
  - **views.py**: Se implementó la función saveFavourite, que procesa los datos enviados desde el formulario en la página de galería. Esta función verifica que el personaje no haya sido marcado previamente como favorito por el usuario. Si no existe, se crea un nuevo registro en la base de datos con los datos proporcionados y se redirige al usuario de vuelta a la galería.

3. **Comentario en Favoritos**:
   - Se implementó una funcionalidad que permite a los usuarios agregar comentarios a sus personajes favoritos.
   - **Modificaciones en archivos**:
     - **models.py**: Se añadió un campo comentario al modelo Favourite, que almacena los comentarios asociados a cada personaje favorito. Para reflejar este cambio, se ejecutó una migración en la base de datos.
     - **views.py**: Se creó la función guardar_comentario, que recibe los datos del formulario con el comentario, verifica que el personaje pertenece al usuario autenticado, y actualiza el campo comentario en la base de datos.
     - **favourites.html**: Se añadió un formulario que permite a los usuarios ingresar y guardar un comentario para cada personaje favorito. El formulario incluye un campo de texto para el comentario y un botón de envío. La función guardar_comentario procesa este formulario y guarda el comentario en la base de datos. Además, se implementaron mensajes flash para notificar al usuario sobre el éxito o posibles errores en la operación.
4. **Formulario y Seguridad**:
   - En **home.html**, se añadió un botón "Agregar a favoritos" para cada personaje. Cada botón envía un formulario con los datos necesarios (URL, nombre, estado, ubicación, etc.) a la función saveFavourite, lo que permite a los usuarios agregar personajes a sus favoritos de forma sencilla.
   - Se incluyó protección CSRF (Cross-Site Request Forgery) para garantizar la seguridad en los formularios.
5. **Rutas y URL**:
   - En **urls.py**, se agregó una nueva ruta para procesar los formularios enviados desde la página de galería. Esta ruta enlaza con la función saveFavourite para gestionar la adición de personajes a los favoritos.

Lógica de la Función Guardar Favorito:

1. Se comprueba si el método HTTP es **POST**.
2. Se obtienen los datos del personaje desde el formulario enviado desde la página de galería.

3. Se verifica que el personaje no haya sido marcado previamente como favorito por el usuario.
4. Si el personaje no existe en la base de datos, se crea un nuevo registro en el modelo Favourite con los datos proporcionados.
5. Después de guardar el favorito, se redirige al usuario de vuelta a la página de galería para asegurar una experiencia fluida y sin interrupciones.

Cambios en home.html:

1. **Carga de CSS**:
   - **DemoTP** carga un archivo CSS personalizado (styles.css), mientras que **Original** no.
2. **Estructura de Cards**:
   - **DemoTP** muestra tarjetas de personajes con bordes de color según su estado (Alive, Dead, Unknown) y más detalles como la última ubicación y el primer episodio.
   - **Original** muestra tarjetas de imágenes con un manejo más simple del estado (iconos de colores) y menos detalles.
3. **Favoritos**:
   - **Original** incluye un formulario para agregar imágenes a favoritos (solo para usuarios autenticados).
   - **DemoTP** no tiene esta funcionalidad de favoritos.
4. **Variables y Condiciones**:
   - **DemoTP** usa la variable personajes y evalúa el estado de los personajes para mostrar colores y bordes.
   - **Original** usa la variable images y tiene una lógica más simple para mostrar el estado con iconos de color.
5. **Paginación**:
   - Ambos incluyen paginación, pero **DemoTP** tiene un código de paginación más detallado y dinámico.

Cambios en def search(request):

1. Obtención del término de búsqueda

- **DemoTP**: Utiliza request.POST.get('query', '').strip() para obtener el texto de búsqueda y eliminar espacios en blanco al inicio y al final.
- **Original**: Usa request.POST.get('query', '') sin el .strip(), lo que significa que los espacios en blanco no se eliminan automáticamente.

2. Lógica de la búsqueda

- **DemoTP**: Si no se ingresa texto de búsqueda (cadena vacía), redirige a la página de inicio con return redirect('home').
- **Original**: Tiene una estructura similar, pero solo verifica si el texto no es vacío con if (search_msg != ''), aunque no realiza ninguna acción adicional (el bloque pass está vacío).

3. Obtención de Datos desde una API Externa

- **DemoTP**: Realiza una llamada a la API de *Rick and Morty* (requests.get(url_api)) para obtener personajes. Luego, procesa la respuesta para extraer información sobre cada personaje, como nombre, estado, imagen, ubicación, y primer episodio.
- **Original**: No realiza ninguna llamada a una API ni procesa datos externos, ya que el bloque relacionado con la búsqueda está vacío (pass).

4. Construcción de la lista de personajes

- **DemoTP**: Si la API devuelve resultados, procesa esos datos para construir una lista de personajes. Cada personaje tiene campos como nombre, estado, imagen, última ubicación y primer episodio.
- **Original**: No realiza ningún procesamiento de datos, ya que la lógica está incompleta en este fragmento.

5. Manejo de Favoritos

- **DemoTP**: Verifica si el usuario está autenticado (request.user.is_authenticated), y si es así, asigna una lista vacía lista_favoritos (aunque no se detallan los favoritos en este código).
- **Original**: No maneja favoritos, ya que el código no implementa ninguna lógica relacionada.

6. Renderización del Contexto

- **DemoTP**: Después de procesar la búsqueda y los datos de personajes, crea un contexto con las variables personajes, favoritos y texto_busqueda, y lo pasa a la plantilla home.html para renderizarla.
- **Original**: No se genera un contexto completo ni se renderiza ninguna plantilla en el código mostrado, ya que no hay procesamiento de datos.

Resumen de los Cambios y Funcionalidades:

1. **Búsqueda**: En **DemoTP**, la búsqueda se realiza llamando a una API externa, se procesan los datos y se renderiza una lista de personajes en la plantilla. En **Original**, la lógica de búsqueda está incompleta, sin realizar ninguna acción significativa.
2. **API**: **DemoTP** consulta la API de *Rick and Morty* para obtener personajes, mientras que **Original** no realiza ninguna llamada a una API.
3. **Favoritos**: **DemoTP** tiene una estructura preparada para manejar favoritos (aunque la lista es vacía), pero **Original** no maneja favoritos en su lógica.

4. **Contexto y Renderización**: **DemoTP** construye un contexto y pasa la información a la plantilla para su visualización, mientras que **Original** no hace nada con la información obtenida.

Durante el proceso de investigación y reestructurado de los códigos y archivos surgieron distintos conflictos que fuimos solucionando poniéndonos de acuerdo en la toma de decisiones.

Algunos de los conflictos que surgieron fueron:

Cuando modificamos el Home HTML para hacer que aparezcan los círculos de estado, no tuvimos en cuenta que también teníamos que modificar Styles.css e importarlo en Home HTML para que se puedan visualizar.

Otro conflicto que surgió fue que cuando se agregó para poner el comentario en favoritos tuvimos que crear una nueva función y por ende modificar el Home y el favourite html

En services.py al principio no entendíamos absolutamente nada y no sabíamos como continuar, pero nos tomamos el tiempo de investigar y luego de varias pruebas-errores, (entre ellos a un participante se le borro media PC con un mal código en el Bash), logramos resolver todos los conflictos.