Noelia Fritz

Informe del Trabajo Práctico – Aplicación Web Django

El presente trabajo práctico consiste en el desarrollo de una aplicación web utilizando el framework Django. El objetivo principal fue implementar un buscador de personajes de Los Simpsons, permitiendo visualizar información relevante y gestionar una lista de favoritos asociada a cada usuario.

La aplicación permite al usuario buscar personajes, visualizar tarjetas informativas y guardar personajes como favoritos. El flujo general del sistema es: búsqueda de personajes, visualización de resultados y almacenamiento opcional en la sección de favoritos.

El sistema fue organizado siguiendo una arquitectura por capas para separar responsabilidades:

• Templates: interfaz visual mostrada al usuario.

• Views: manejo de requests y respuestas HTTP.

• Services: lógica de negocio.

• Utilities/Translators: conversión de datos entre capas.

• Models: representación de datos persistidos en base de datos.

Funcionalidades Implementadas

• Buscador de personajes mediante consumo de API.

• Visualización de tarjetas con información del personaje.

• Sistema de favoritos asociado al usuario autenticado.

• Persistencia de datos utilizando base de datos SQLite.

Problemas Encontrados y Soluciones

Durante el desarrollo se presentaron distintos problemas técnicos que permitieron comprender el funcionamiento interno de Django:

• Errores de mapeo de datos en translators debido a diferencias entre objetos de request, API y base de datos. Se solucionó ajustando funciones de conversión.

• Error AttributeError por métodos inexistentes, causado por inconsistencias en nombres de funciones entre capas.

• Error IntegrityError por restricción UNIQUE al intentar guardar favoritos duplicados, lo que permitió comprender el rol de las restricciones de integridad en la base de datos.

• Error ValueError al guardar la edad cuando el campo era None. Se resolvió asignando un valor por defecto y convirtiendo correctamente a entero.

Se logró implementar una aplicación funcional con búsqueda y visualización de personajes. El sistema de favoritos quedó parcialmente operativo, permitiendo comprender el flujo completo entre frontend, backend y base de datos.

Cambios realizados para que funcione la página DETALLADAMENTE:

- **Corrección de la vista home (problema principal)**
  Al inicio, la página cargaba, pero **no mostraba imágenes** porque la vista no llamaba a la capa de servicios.
  La planilla recibía listas vacías, así que agregue la obtención real de datos desde servicios.
  Images= servicies.getAllimages()
  Esto permitió llamar a la API, transformar los datos y enviar tarjetas a la plantilla.
- Implementación de **los servicios de la capa:**
  Acá se complementaron funciones que estaban con pass.
  GetAllImages()

Responsabilidades: pedir datos a transport , convertir JSON a tarjeta usando translator y retomar listas de cards.
Esto activo toda la arquitectura por capas del tp.

Conexión correcta con la capatransport:

- Acá se verifico que requests.get(config.SiMPSONS_CHARACTER_URL) Realmente trajera datos y que existiera portrait_path y que también se armara correctamente la colección JSON.
  Antes de esto la lista quedaba vacía.

Uso correcto deltranslator:

- Acá se usó el flujo de tranformacion:
  API= FromRequestIntoCard
  Planilla= FromTemplateIntoCard
  Repositorio= FromRepositoryIntoCard

  Para que esto funcione fue super importante la construcción correcta de la URL de imagen y la creación del objeto Card.
  Esto permitió que aparezcan las imágenes, la edad, el estado, la ocupación y el género.

Corrección del templatehome.html:

Acá se reemplazó el bloque completo por el fragmento corregido que paso el profesor.

Ya que faltaba cerrar un .

Implementación de búsqueda por nombre:

Incorporación del formulario de búsqueda en la interfaz

Se agregó un formulario en la página principal que permite al usuario ingresar un nombre.

La página dejó de ser únicamente visual y pasó a enviar información al backend.

Se implementó una función específica encargada de:

- recibir el texto ingresado por el usuario,
- validar que la búsqueda no esté vacía,
- ejecutar el filtrado de personajes.

Antes de este cambio, la aplicación solo tenía una vista que mostraba todos los elementos sin lógica de filtrado.

En lugar de trabajar con los datos ya mostrados en la pantalla, la búsqueda solicita nuevamente la lista completa de personajes mediante la capa de servicios.

Esto se realizó para respetar la arquitectura del proyecto y mantener separadas las responsabilidades.

Se agregó una lógica que:

- recorre todos los personajes,
- comparar el nombre de cada uno con el texto buscado,
- guarda únicamente los que coinciden.

Además, la comparación se realiza sin distinguir mayúsculas y minúsculas para mejorar la experiencia de uso.

La misma plantilla principal se reutiliza, pero ahora recibe una lista filtrada en lugar de la lista completa.

De esta forma:

- si no hay búsqueda, se muestran todos los personajes,
- sí hay búsqueda, sólo aparecen los resultados coincidentes.

Configuración correcta del servidor Django:

Mediante la terminal de Visual studio se verifico el acceso a la página mediante Python mange.py runserver ,para ver la página actual de nuestro proyecto, y no desde el puerto 3000 del fronted original.

De esta manera confirmaba que estaba usando la versión Django del tp y no la plantilla original.

En resumen, los códigos modificados fueron:

- Services.py
- Transport.py
- Home.html
- Views.py

Con estas modificaciones la página ya funcionaba.

Dificultades adicionales durante el desarrollo

Durante la implementación se detectaron errores que no estaban relacionados con la lógica del programa sino con la configuración del proyecto, especialmente: cambios no reflejados en la página, archivos que no se actualizaban correctamente y problemas al sincronizar el repositorio. Esto dificultó identificar si el error provenía del código o del entorno de trabajo.

En varias ocasiones la aplicación mostró errores generales o páginas que no cargaban, sin indicar claramente cuál era la causa. Esto generó dificultades para localizar el problema exacto. Para solucionarlo revise cuidadosamente la consola y aísle los cambios pequeños para identificar qué modificación produjo el error y tuve que remover varias veces cambios.

Al agregar la búsqueda, algunas funcionalidades que ya funcionaban dejaron de hacerlo temporalmente.

Esto ocurrió porque:

- se modificaron los datos enviados a la plantilla,
- o cambió la estructura esperada por la vista.

Así que decidí mantener el mismo formato de datos que utilizaba la página originalmente para evitar efectos secundarios.

Al principio resultó complejo entender qué archivo debía modificarse para cada cambio:

- vistas,
- plantillas,
- servicios,
- rutas.

Esto generó intentos de solución en lugares incorrectos. Así que analizamos la estructura general del proyecto antes de implementar nuevas funcionalidades.

Al trabajar con repositorio surgieron conflictos al subir cambios, me costó demasiado poder subir los cambios de mi código al repositorio. También tuve problemas porque se duplico el tp en mi computadora, tuve que actualizar el repositorio antes de realizar nuevos cambios y verificar el estado local.

Conclusión

El trabajo permitió aplicar conceptos fundamentales de desarrollo web con Django, incluyendo separación por capas, manejo de requests HTTP, persistencia de datos y resolución de errores reales de integración. La experiencia contribuyó a comprender la importancia del diseño estructurado y la depuración progresiva.