# Trabajo Final

## Aplicación Web con Framework Python

**Integrantes del grupo**

Casas Oriana Denisse, López Franco Benjamín, Lisandro Martín Emiliano

**Días 0 y 1| 9 y 10 de Junio.**

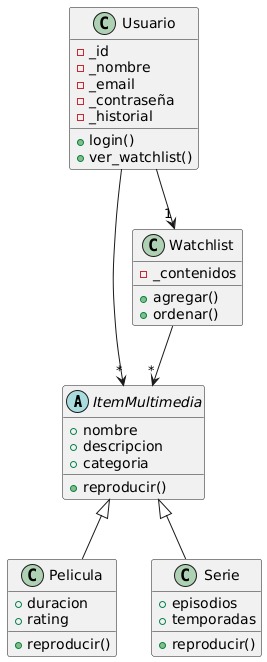
**La noche anterior al 10 de junio, cuando el profesor mandó las pautas del TPI, nuestro grupo se reunió por Discord para debatir qué tema íbamos a elegir. Y tras decidirnos por la Aplicación Web con Framework Python, debatimos qué framework utilizaríamos para el desarrollo de nuestra aplicación web. Evaluamos distintas alternativas como Flask, Django y FastAPI, considerando aspectos como facilidad de uso, curva de aprendizaje, compatibilidad con Python y capacidad para construir interfaces gráficas de manera intuitiva.**

**Nuestra segunda opción más fuerte fue Flask, ya que es un microframework ampliamente utilizado, con buena documentación y una estructura flexible. Sin embargo, requería complementar la interfaz con HTML y JavaScript, lo cual iba en contra de nuestro objetivo de trabajar exclusivamente con Python. Y teniendo solo 10 días para realizar el proyecto, no había mucho tiempo de aprender cosas nuevas.**

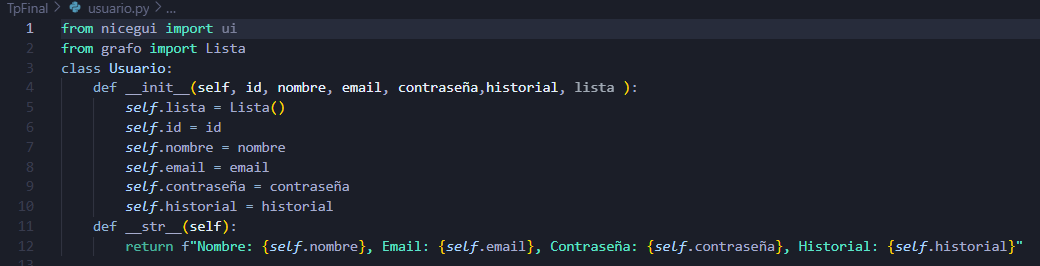
**Finalmente, optamos por NiceGUI, un framework que permite crear interfaces gráficas modernas y responsivas utilizando únicamente Python. A diferencia de otros frameworks que requieren combinar varios lenguajes, NiceGUI permite diseñar y controlar toda la interfaz desde código Python puro, lo que nos resultó ideal dada nuestra experiencia previa. Además, su integración directa con componentes visuales, su capacidad de actualización en tiempo real mediante WebSockets, y la facilidad para estructurar la aplicación en base a clases, lo convirtieron en una opción coherente con los objetivos del trabajo.**

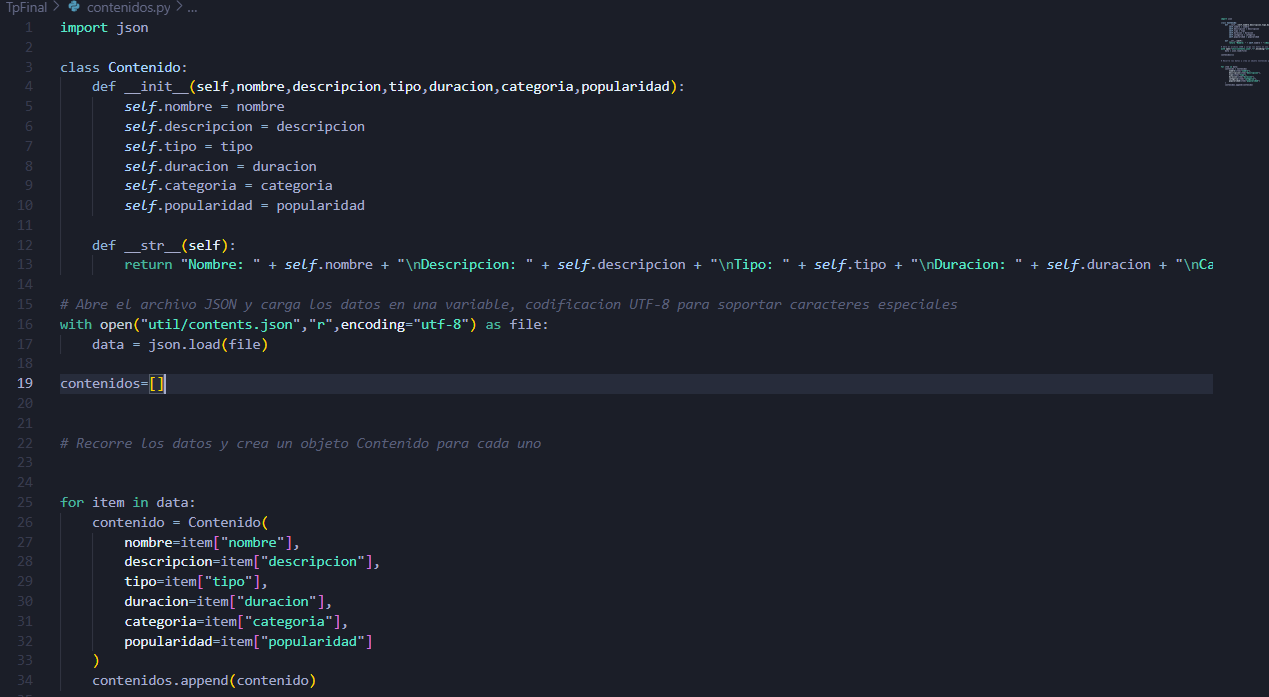
**Durante la mañana del 10 de junio, una vez recibido el visto bueno del profesor para utilizar este framework, comenzamos a ver videos introductorios sobre su funcionamiento y características. Paralelamente, empezamos a organizarnos en torno al diseño general de la aplicación. Decidimos construir una plataforma de streaming ficticia, ya que nos permitiría aplicar múltiples aspectos de la programación orientada a objetos (POO), incluyendo el modelado de usuarios, catálogos de contenido y funciones de reproducción.**

**Para dar los primeros pasos en la planificación del código, realizamos un diagrama UML preliminar en PlantUML, donde esbozamos las clases principales y sus relaciones, aplicando principios de encapsulamiento, herencia y organización modular.**



**Días 2 – 4| del 11 al 13 de Junio.**

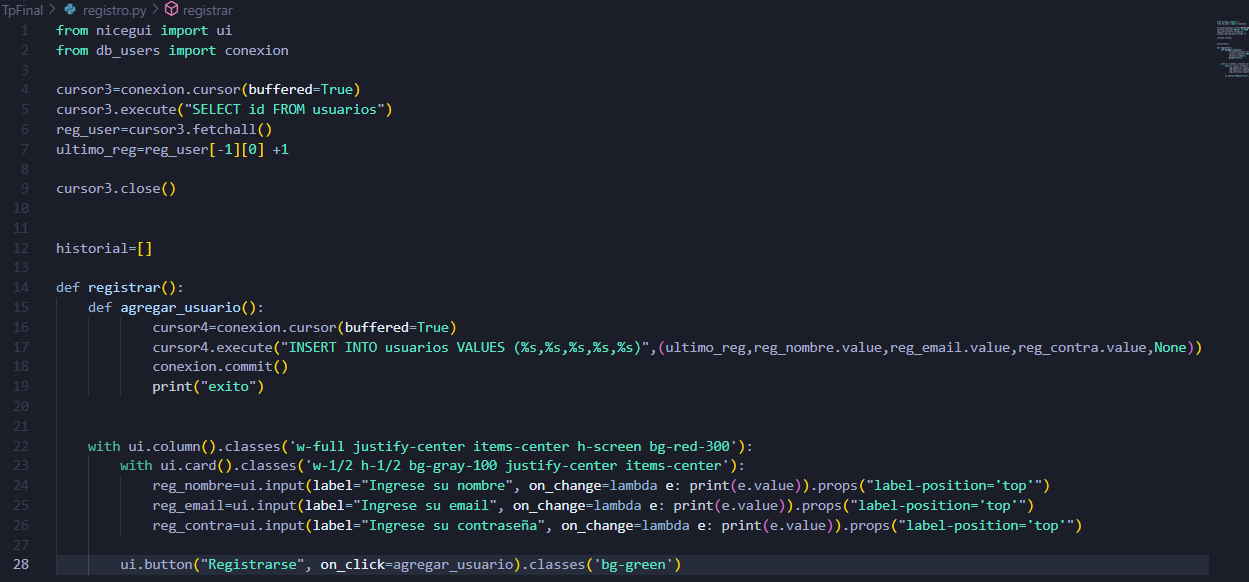
Durante las primeras etapas del proyecto, Lisandro y Franco se encargaron de la estructuración básica de las clases *Usuario* y *Contenido*, que constituyen el punto de partida del desarrollo de nuestra aplicación ficticia de plataforma de streaming. Para facilitar la colaboración y el seguimiento del progreso mutuo, compartieron pantalla a través de Discord.



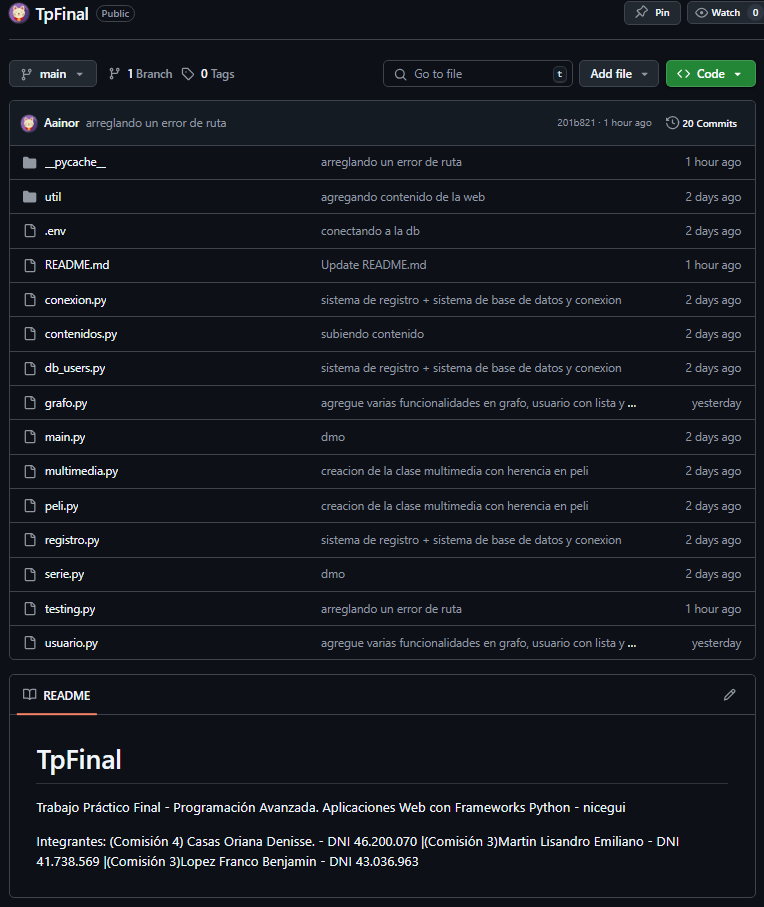
Posteriormente, se unió Oriana a la llamada, y comenzamos con la configuración de la base de datos. Optamos por utilizar la biblioteca mysql-connector-python, ya que nos ofrecía mayores funcionalidades y flexibilidad en comparación con SQLite, que si bien viene integrado en Python, nos resultaba limitado para los fines de nuestro proyecto.

Una vez instalada la biblioteca, procedimos a configurar la conexión con una base de datos MySQL. Esta conexión se establece utilizando el método mysql.connector.connect, al que se le deben proporcionar ciertos parámetros: host, port, user, password y database. Una vez conectados, es posible definir uno o varios cursores (por ejemplo, cursor = conexion.cursor()) para ejecutar comandos SQL mediante el método execute().

Como parte de esta etapa, decidimos inicializar el historial de contenidos como una lista vacía. Para ello, implementamos un diccionario cuyas claves (keys) representan los nombres de las columnas en la base de datos.

Con la conexión funcionando correctamente, desarrollamos una interfaz de registro de usuario, donde se solicitan el nombre, correo electrónico y contraseña, y estos datos se almacenan directamente en la base de datos MySQL.****

**Día 5 | 16 de Junio.**

**Oriana creó el repositorio del proyecto en GitHub, permitiéndonos subir el código ya desarrollado y trabajar de forma colaborativa. Se clonó el repositorio en cada computadora y comenzamos a actualizarlo progresivamente hasta la fecha de entrega del trabajo práctico. Además, incluimos en el archivo README.txt la información correspondiente a cada integrante del grupo y añadimos al profesor Gianluca como colaborador para que pudiera acceder al repositorio.**

**Ese mismo día volvimos a reunirnos por Discord. Mientras Oriana y Franco se encargaban de crear los archivos *multimedia.py, serie.py y peli.py*, Lisandro desarrolló el archivo *main.py* y agregó funcionalidad a varios archivos previamente creados. Por ejemplo, completó la clase *Serie* añadiendo un método de instanciación que había sido omitido inicialmente.**

**También se implementó un sistema de login, que valida si el usuario ingresado existe en la base de datos. En caso afirmativo, se instancia un objeto Usuario con sus datos y se asigna a una variable global session\_usuario, la cual representa al usuario logueado durante su sesión activa. Esto nos permitió personalizar la navegación y conservar la información del usuario mientras usa la plataforma. Asimismo, se definió un sistema de rutas en la aplicación para organizar la navegación entre diferentes secciones. Algunas de las rutas implementadas fueron:**

**/login: pantalla de inicio de sesión.**

**/home: página principal con botones de navegación.**

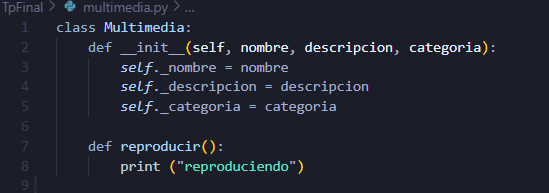
**/peliculas y /series: catálogos filtrados por tipo de contenido.**

**/{nombre}: detalle de una serie con sus capítulos.**

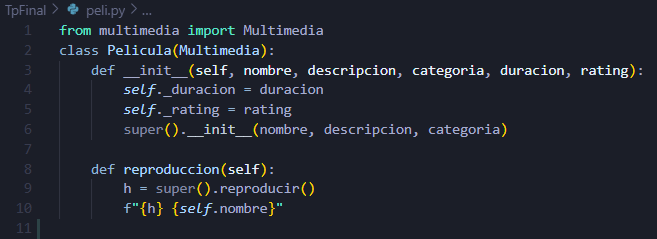
**/top: sección que muestra los contenidos más populares.**

**Estas rutas están asociadas a funciones específicas y componentes visuales de NiceGUI, lo que permitió una experiencia de usuario fluida e intuitiva.**

**En esta instancia, comenzamos a aplicar los principios de Programación Orientada a Objetos (POO) según lo diagramado previamente en el modelo UML. La clase *Multimedia* fue definida como clase padre, con un método *reproducir* común. Las clases *Serie* y *Película* heredaron tanto atributos públicos como privados de *Multimedia*, y cada una implementó su propia versión del método *reproducir*, ajustada a su tipo de contenido.**

****

****

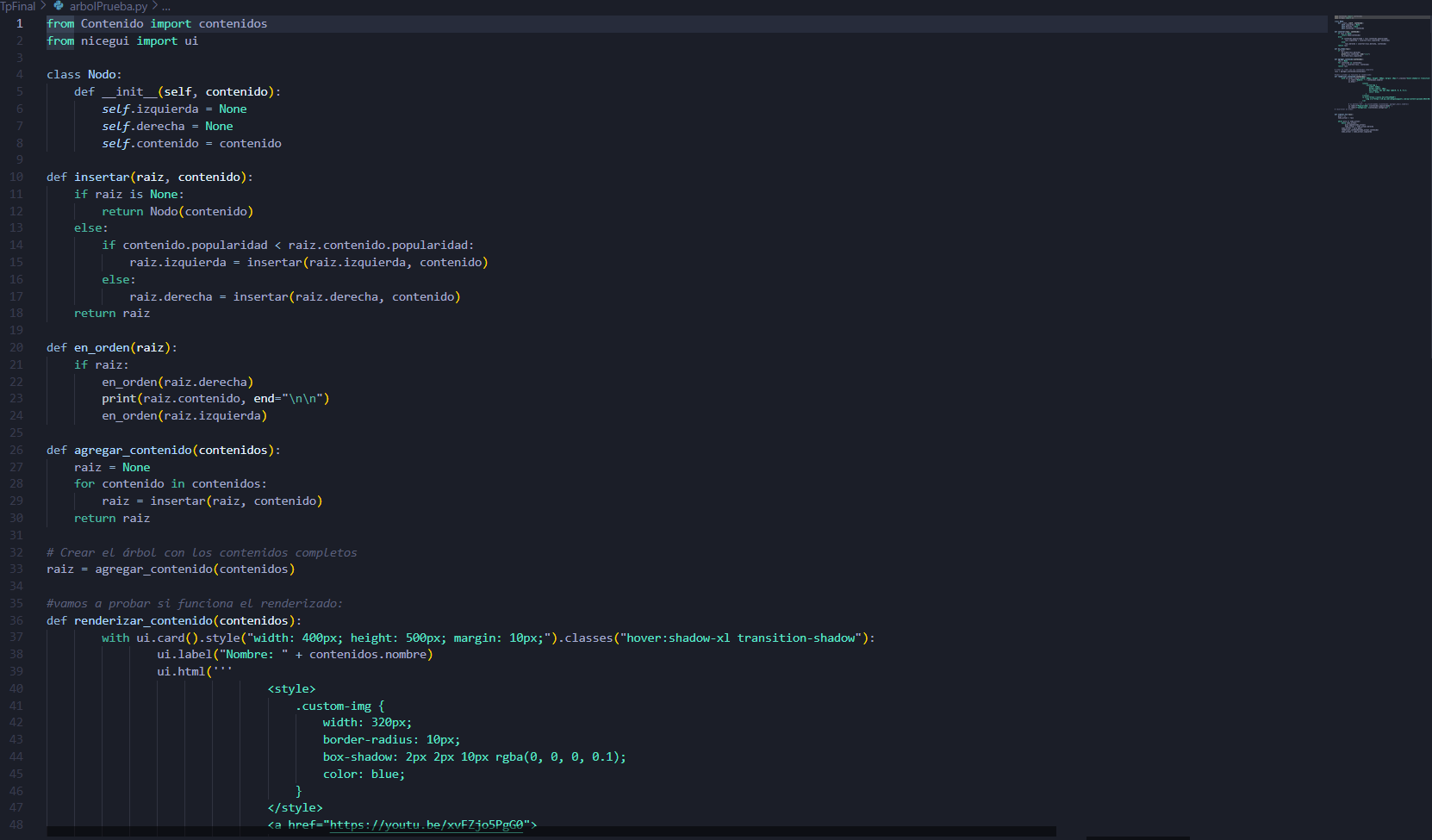
****

**Días 6 – 7 | del 17 al 18 de Junio**

**Durante esta etapa, nos enfocamos en la implementación de estructuras de datos avanzadas para optimizar la organización y navegación del contenido en nuestra plataforma. En particular, utilizamos árboles para representar la estructura jerárquica de las series, y grafos para implementar un sistema de recomendaciones de contenido personalizado. Nuevamente, esto se organizó a través de llamadas por Discord entre los tres integrantes del grupo, pero fueron Lisandro y Franco quienes realizaron los commits esta vez.**

**Para modelar de forma eficiente la relación entre series y sus episodios, decidimos representar cada serie como la raíz de un árbol, cuyos hijos son los objetos de tipo Capitulo. Esta estructura nos permitió acceder rápidamente a los episodios de una serie y trabajar con sus propiedades específicas (como duración, número de episodio, o si fue visualizado), manteniendo una jerarquía clara entre contenidos.**

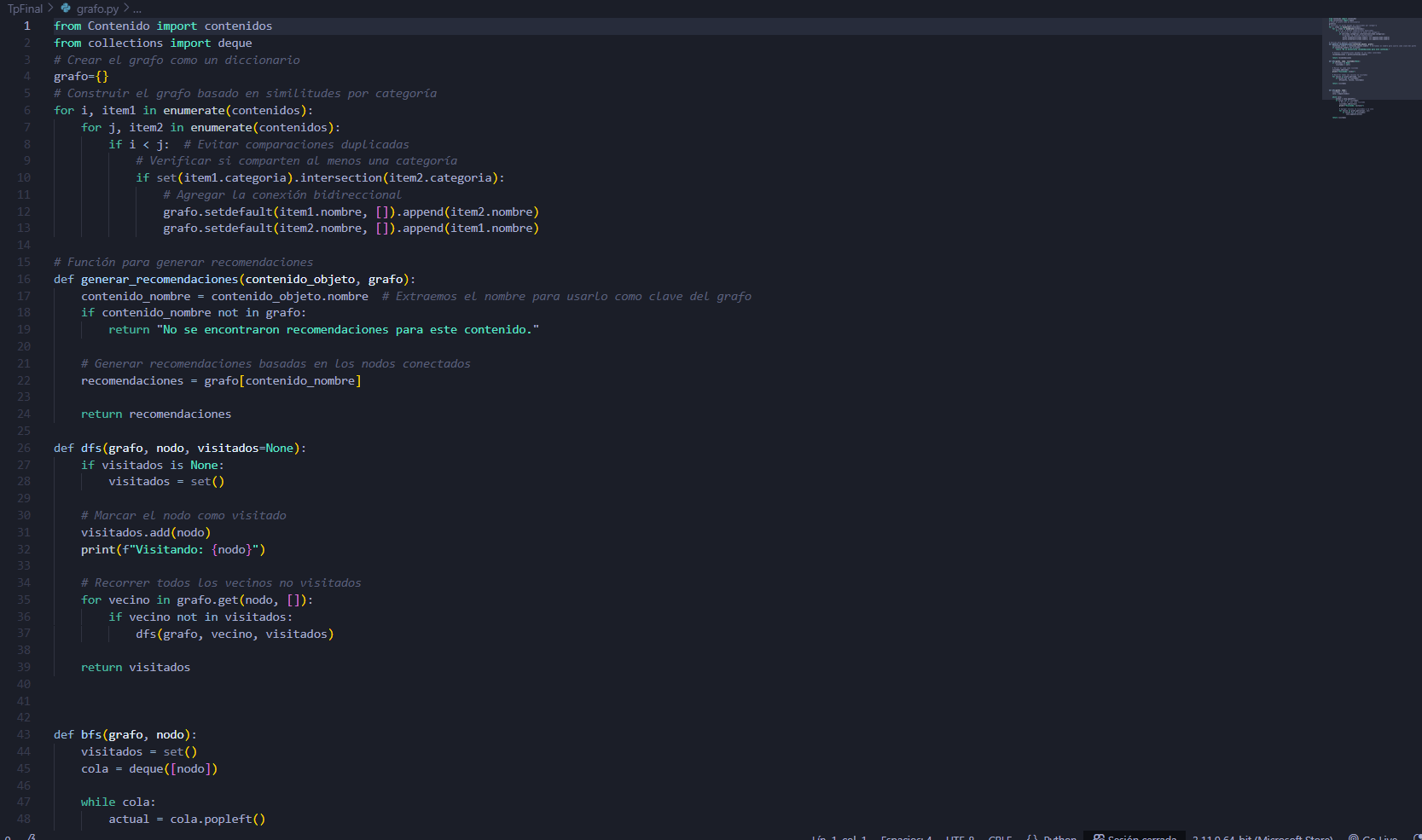
**Esta decisión resultó especialmente útil al momento de mostrar los capítulos en la interfaz y al implementar funciones como “reproducir siguiente episodio”, ya que podíamos recorrer los hijos del nodo raíz (la serie) de forma ordenada.**



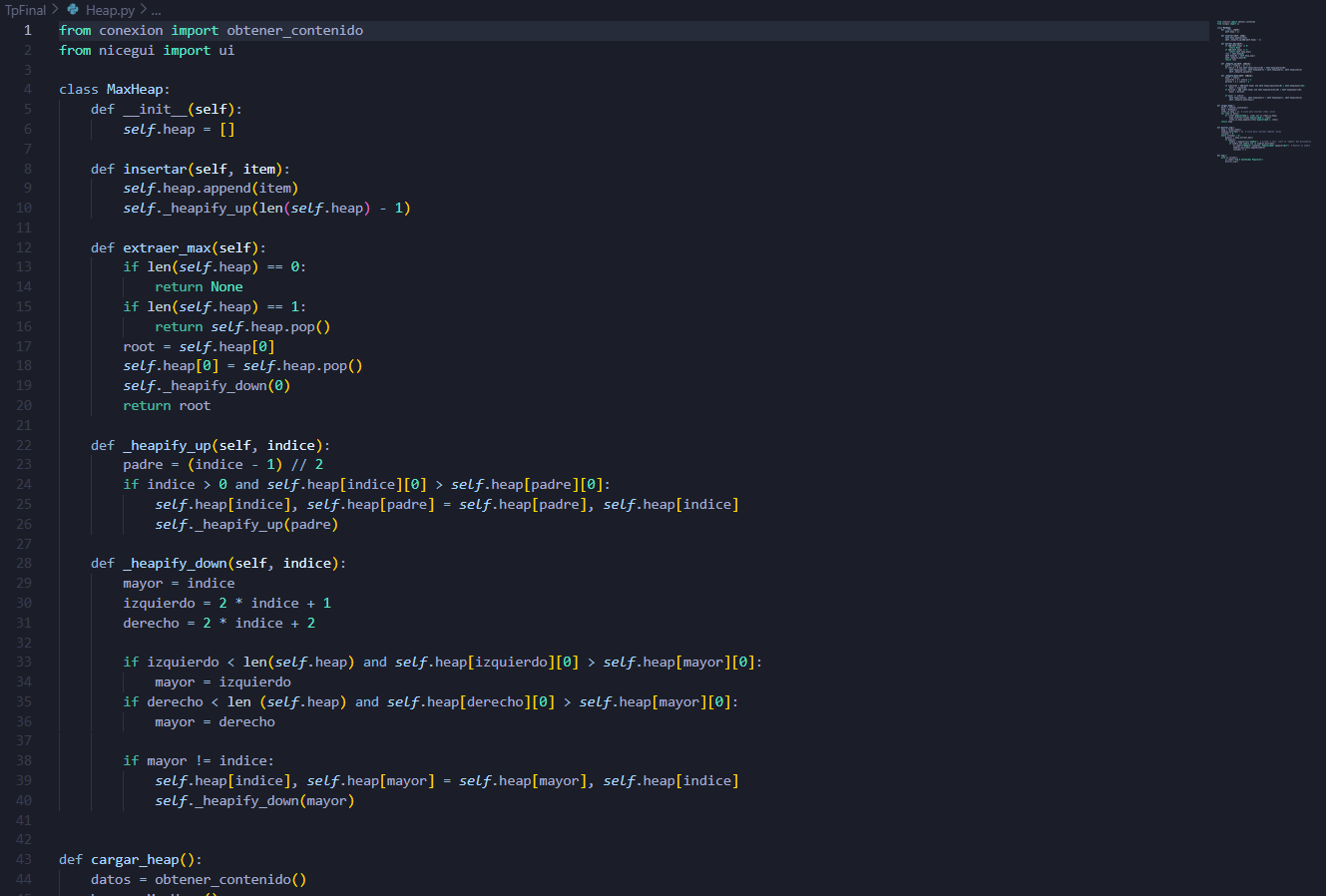
**Además de la estructura en árbol, implementamos un grafo no dirigido con el objetivo de ofrecer recomendaciones basadas en similitudes temáticas entre contenidos. El grafo se construyó como un diccionario de adyacencia, donde:**

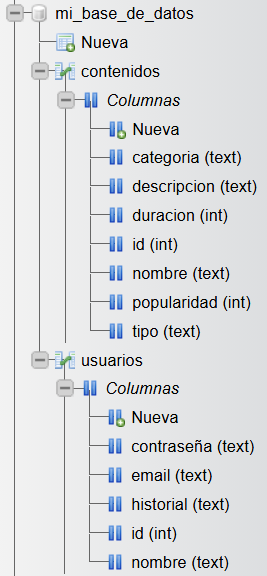
**Las claves son los nombres de los contenidos (series o películas).**

**Los valores son listas de contenidos relacionados, conectados por compartir al menos una categoría en común (por ejemplo, "Drama", "Comedia", etc.). La lógica de conexión se basa en comparar todas las combinaciones posibles de contenidos. Si dos contenidos comparten una o más categorías, se establece una arista entre ellos.**



También se desarrolló una sección /top dedicada a mostrar los cinco contenidos más populares de la plataforma. Para ello, utilizamos un heap binario máximo, que nos permitió mantener actualizada una lista de los contenidos más vistos sin necesidad de ordenar toda la base cada vez. Esta estructura hizo más eficiente la consulta y visualización dinámica del Top 5.



****Durante estos dos días también surgieron algunos errores, que Lisandro y Oriana lograron identificar como problemas en las rutas utilizadas para acceder a los archivos JSON. **Una vez corregidas estas rutas, la aplicación funcionó correctamente, y solo restaba un último detalle: Oriana se encargó de la creación y organización final de las tablas en la base de datos, completando así la funcionalidad integral del proyecto.**



Con la aplicación terminada, comenzamos también el desarrollo de la presentación en PowerPoint y la grabación de un video demostrativo para mostrar el funcionamiento de nuestra plataforma, los cuales se encontrarán en la misma carpeta que este informe.

**Conclusión**

**El desarrollo de esta aplicación web nos permitió aplicar múltiples conceptos de programación orientada a objetos, estructuras de datos y trabajo colaborativo. Pudimos integrar correctamente tecnologías como NiceGUI, MySQL y algoritmos de grafos y árboles para lograr una plataforma funcional, visual y organizada.**

**Si bien enfrentamos algunos desafíos técnicos durante el proceso, logramos resolverlos en equipo y completar todas las funcionalidades propuestas dentro del plazo. Este proyecto no solo reforzó nuestros conocimientos teóricos, sino que también nos dio una experiencia práctica valiosa en el desarrollo de software real.**

**Repositorio del proyecto: https://github.com/Aainor/TpFinal**