Documentación Proyecto Final – Plataforma Netflix

# 1. Explicación de cada carpeta del proyecto

* models/

Contiene las clases que representan las entidades del dominio: Usuario, Película, Serie, Visualización. Estas clases encapsulan atributos y comportamientos clave del sistema.

* services/

Contiene la lógica de negocio, como el registro de usuarios, la autenticación y la visualización de contenidos. Implementa el patrón Service Layer.

* factories/

Aquí se ubican las fábricas de objetos (Factory Pattern), como la creación de usuarios y contenidos. Permiten desacoplar la lógica de instanciación.

* repositories/

Encapsulan el acceso a la base de datos. Implementan el patrón Repository, y permiten obtener, insertar o modificar datos.

* db/

Contiene exclusivamente el módulo de conexión a la base de datos (SQL Server), separando la infraestructura del resto del proyecto.

* utils/

Contiene funciones auxiliares, como validaciones o utilidades compartidas. También puede contener excepciones personalizadas (pendinete).

* data/

Incluye archivos de datos como contenidos.csv o usuarios.csv para pruebas o carga inicial del sistema.

* tests/

Contiene pruebas unitarias usando pytest, permitiendo verificar el comportamiento del Código(ejemplo).

* main.py

Archivo principal desde el cual se ejecuta la aplicación.

# 2. ¿Por qué es conveniente el patrón Factory en este proyecto?

El patrón Factory permite centralizar la creación de objetos como Usuario, Película o Serie. Esto es útil porque abstrae la lógica de instanciación fuera del código principal. Además, mejora la mantenibilidad, permite validar o transformar los datos antes de crear el objeto, y facilita la extensión futura del sistema (por ejemplo, para crear nuevos tipos de contenido).

# 3. Otro patrón aplicable: Strategy

El patrón Strategy permite definir una familia de algoritmos, encapsularlos y hacerlos intercambiables. Podría usarse, por ejemplo, para distintos tipos de visualización o recomendación de contenido:

Ejemplo:

class EstrategiaVisualizacion:  
 def visualizar(self, contenido):  
 pass  
  
class VisualizacionHD(EstrategiaVisualizacion):  
 def visualizar(self, contenido):  
 print(f"Reproduciendo {contenido.titulo} en HD")  
  
class Visualizacion4K(EstrategiaVisualizacion):  
 def visualizar(self, contenido):  
 print(f"Reproduciendo {contenido.titulo} en 4K")  
  
def reproducir\_con\_estrategia(contenido, estrategia):  
 estrategia.visualizar(contenido)

# 4. ¿Cómo lo haría sin usar patrones de diseño?

Si no conociera los patrones de diseño, estructuraría el proyecto de forma más básica, agrupando funcionalidades en módulos separados. Por ejemplo:

- Un archivo para la lógica de usuarios (`usuarios.py`).

- Un archivo para la lógica de contenidos (`contenidos.py`).

- Un archivo para las operaciones de base de datos (`basedatos.py`).

- El archivo principal (`main.py`) incluiría las llamadas entre estos módulos.

Este enfoque es más simple, pero a medida que el proyecto crece se vuelve más difícil de mantener, escalar y probar. Por eso, usar patrones como Factory, Repository y Service Layer es altamente recomendable en entornos reales.