Universidad ORT Uruguay
Facultad de Ingeniería

Obligatorio 1
Diseño de Aplicaciones 2
Descripción del Diseño

Repositorio: https://github.com/IngSoft-DA2-2023-2/242493-260956-281651
Video detallando pruebas: https://youtu.be/Mo39Tbi6L1s

Romina Arour - 242493
Daiana Aysa - 281651
Ana Gutman - 260956

Tutores:

2023

ÍNDICE

Descripción del Diseño

En esta sección, vamos a ofrecer una visión general de nuestro sistema. La construcción de esta solución se basa en los requisitos delineados en el documento proporcionado por nuestros tutores.

Para cumplir con los objetivos establecidos, creamos varios proyectos que, en conjunto, constituyen la solución completa de la aplicación. Más adelante en el informe, detallaremos estos proyectos para aclarar aún más el propósito del sistema.

Construimos una Web API, implementando el backend y la API en sí misma, creando los endpoints necesarios para satisfacer las solicitudes. La interacción con la misma fue mediante Postman.

En lo que respecta al trabajo en equipo, utilizamos un repositorio en GitHub, siguiendo la metodología de trabajo de GitFlow.

El proyecto se basa en un E-Commerce donde se compra ropa.
Los usuarios se pueden registrar como compradores, administradores o pueden ser ambos.
Los usuarios compradores solo pueden ver los productos disponibles, comprarlos y ver su propio historial de compras.

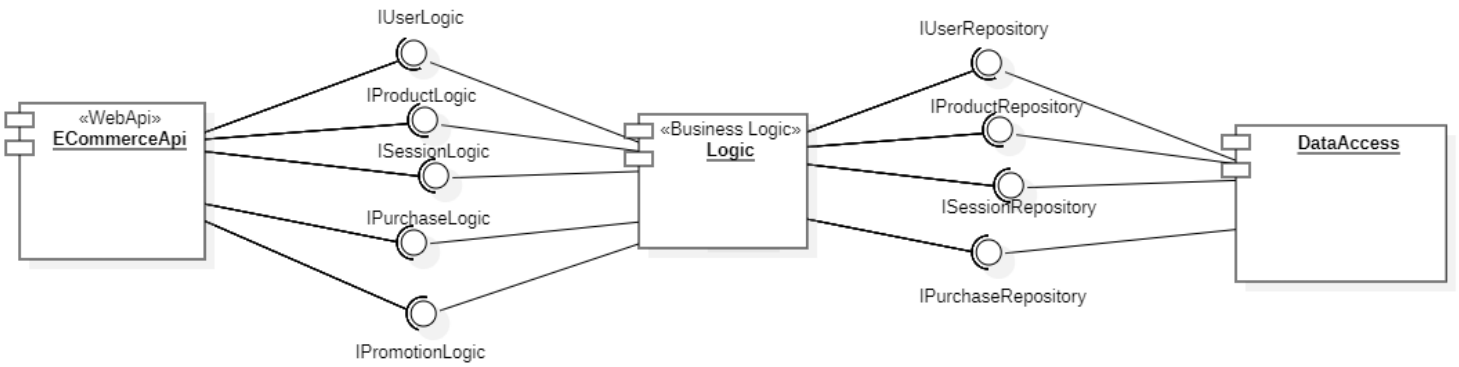
Los administradores son aquellos quienes publican, modifican, borrar productos en la aplicación web junto con ver todo el historial de compras de la página. También tienen la opción de ver los perfiles de los otros usuarios registrados, tanto una lista de usuario como también verlos específicos. Si se desea, los administradores pueden eliminar a otros usuarios también.

Descripción de paquetes

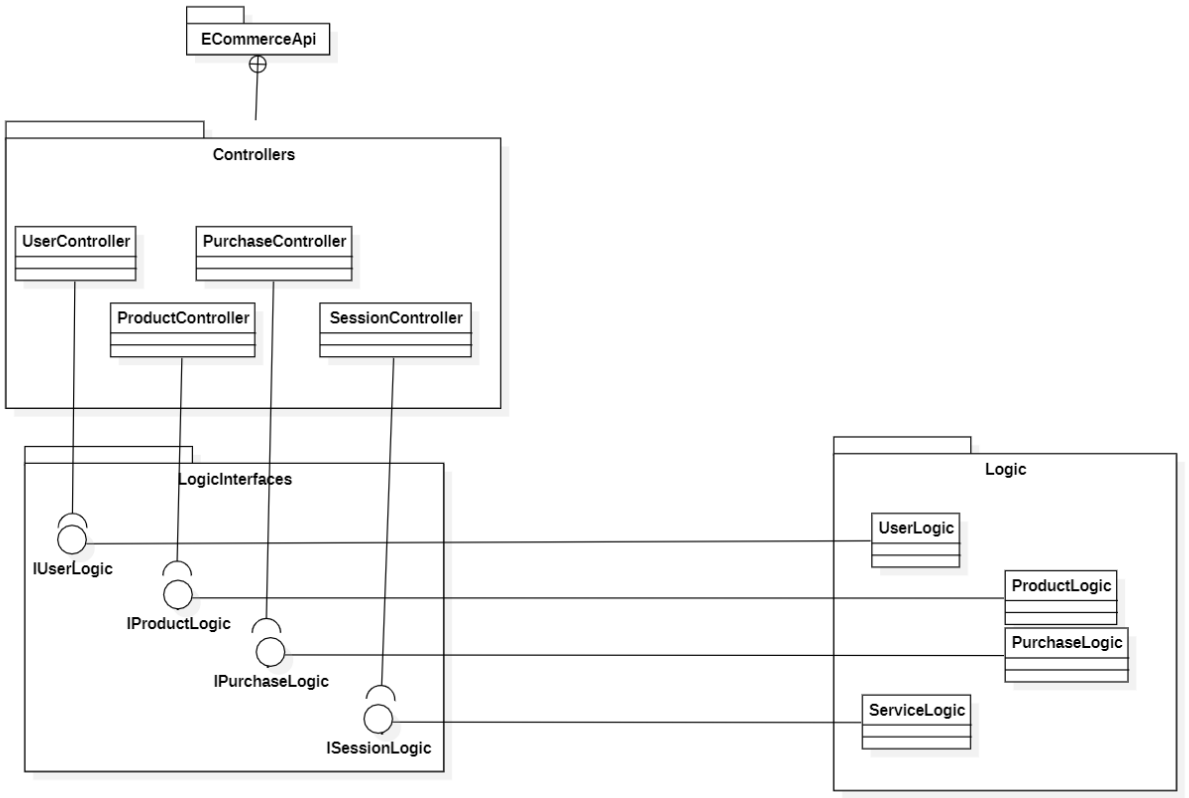
Uso del modelo 4 + 1 para la arquitectura del sistema.

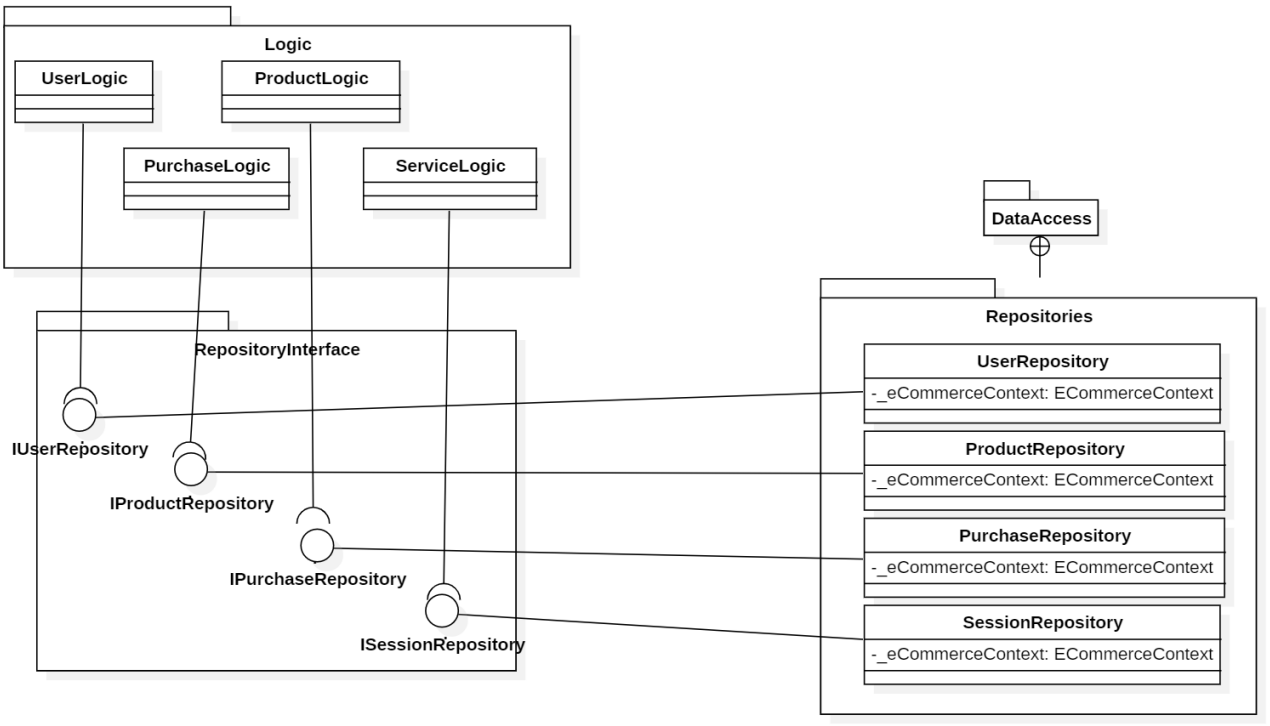
Vista de Componentes

Para la vista de desarrollo se diseñó el siguiente diagrama de componentes



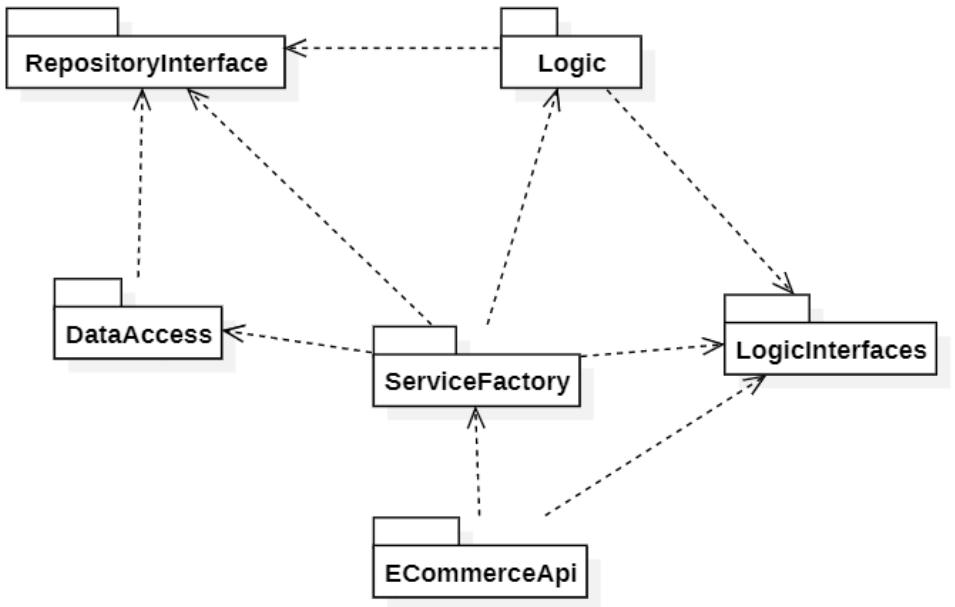
A su vez, diseñamos los siguientes diagramas de paquetes:





En los diagramas anteriores no se incluyó al manejo de las promociones dado que estos son vistas lógicamente de la conexión entre los controladores y la lógica, y la lógica con los repositorios respectivamente. En cambio, aunque tengamos una clase PromotionLogic en el proyecto Logic, y una interfaz IPromotionLogic en el proyecto de Logic interface, estos son utilizados dentro de la lógica de las compras en Purchase logic, por lo que no participan de las interacciones que se ilustran.

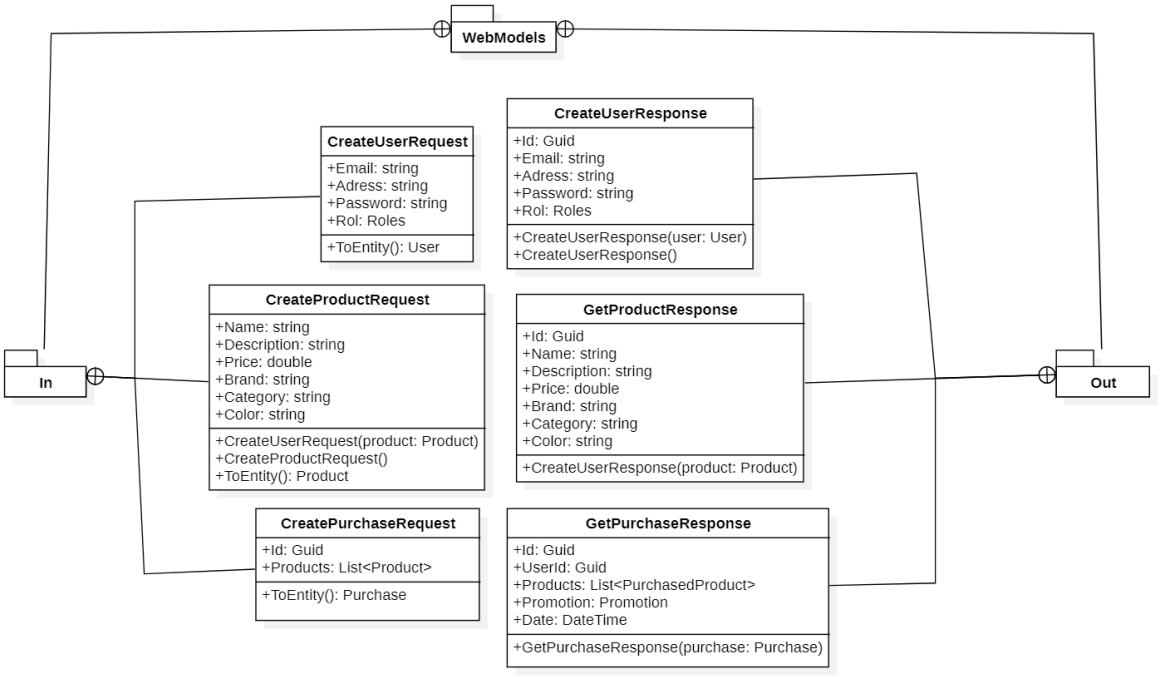
Finalmente, el siguiente diagrama de paquetes evidencia la inyección de dependencia realizada gracias a la herramienta provista por Visual Studio.

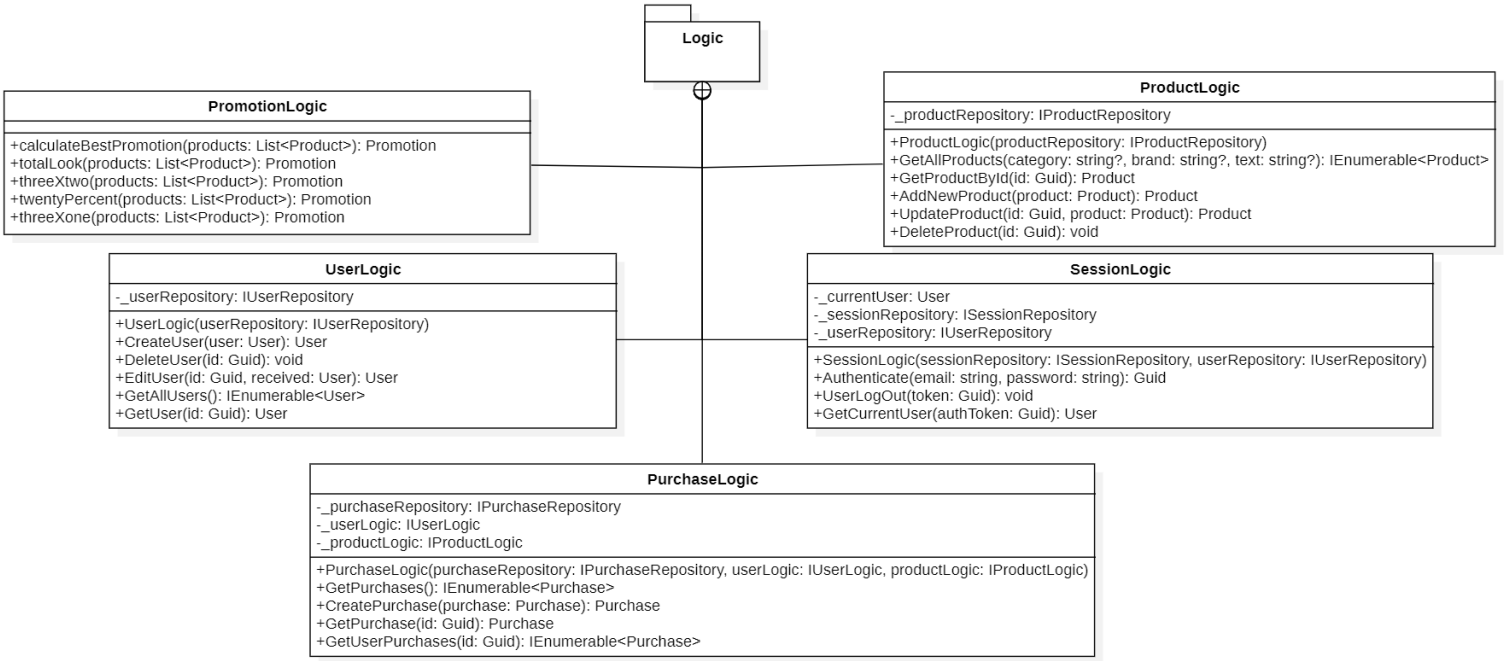


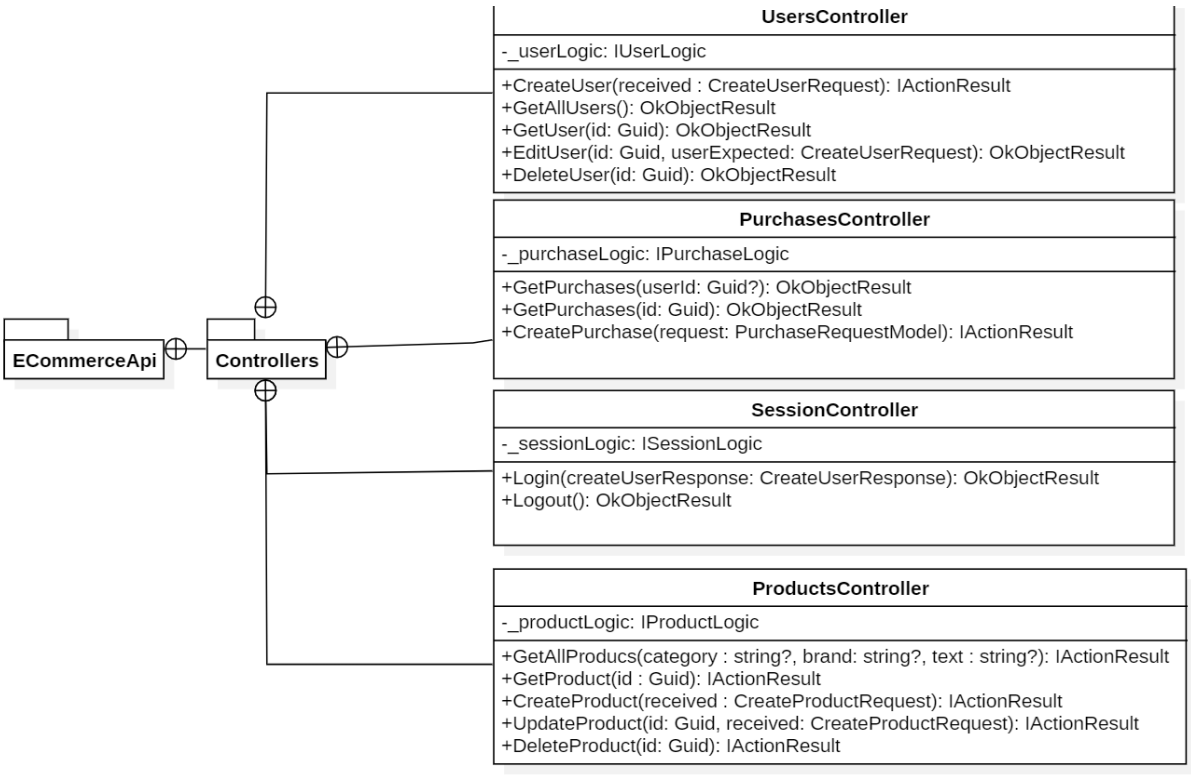
Con el ServiceFactory, buscamos aplicar el patrón de Inyección de Dependencia, que tiene como objetivo proporcionar los objetos de una clase en lugar de requerir que la propia clase los genere. Al adoptar este enfoque, logramos una mayor flexibilidad y mantenibilidad en nuestro código al separar la responsabilidad de la creación de objetos. Logró facilitar la gestión de dependencias en nuestro sistema.

Vista lógica

Para la vista lógica se crearon los siguientes diagramas:







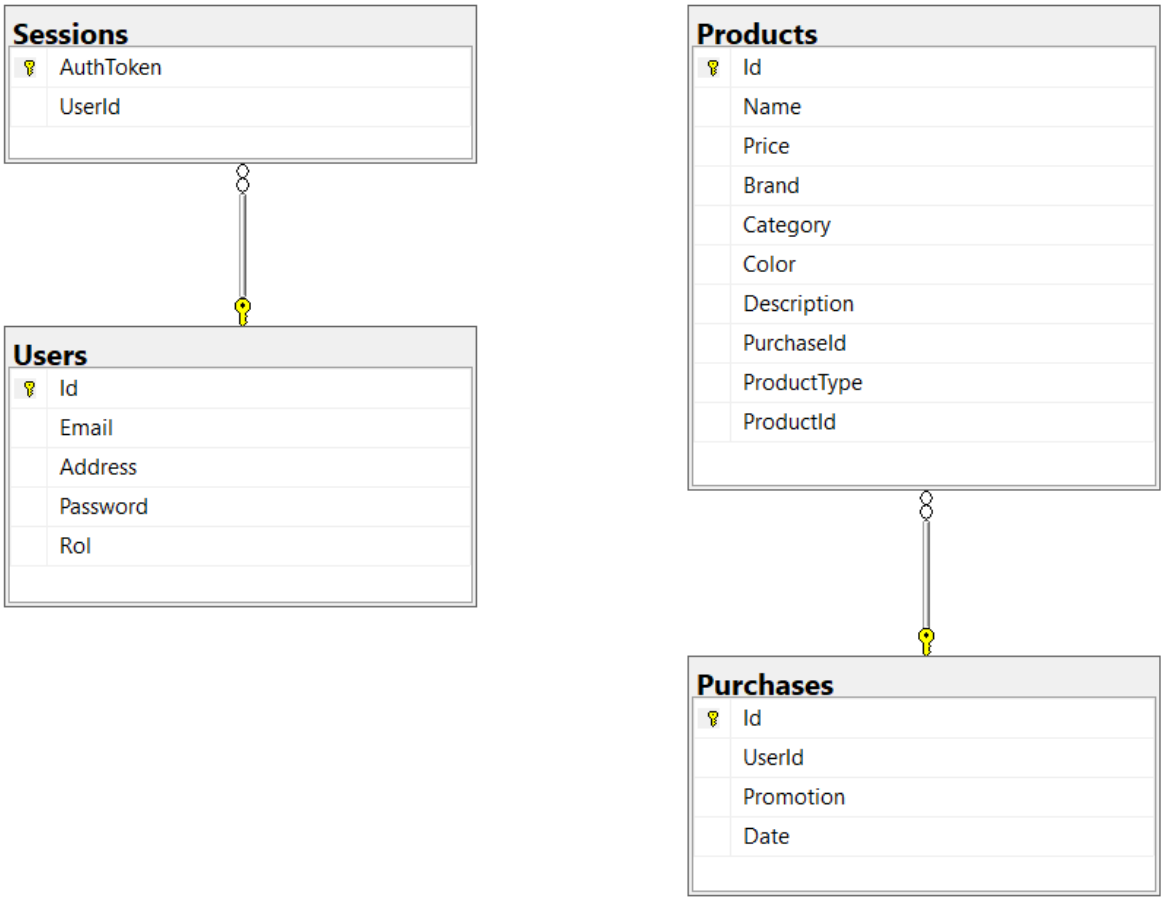
Vista de fisica

Esta vista no fue implementada dado que el alcance del Obligatorio no es suficientemente extenso como para especificarla.

Descripción de jerarquías

Decidimos utilizar una herencia en el diseño con la clase Product como padre y PurchasedProduct cómo su hija. Hicimos la clase hija con el objetivo de poder registrar los valores de los atributos de los productos al momento en el que son comprados, para tener acceso a ellos en el historial de compras a futuro sin que se vean afectados por posibles ediciones al producto luego de efectuada la compra. Esto es de particular importancia para el valor del precio. Esta decisión fue tomada teniendo en cuenta el principio de substitución de Liskov, porque la clase hija se comporta como su clase padre y suma funcionalidad que los productos no poseen.

Modelo de Tablas de la Base de Datos



Justificación de diseño para el manejo de DBContext:

En el diseño de nuestra aplicación, hemos adoptado un enfoque eficiente y organizado para la configuración del modelo de datos utilizando Entity Framework Core en el contexto de base de datos `ECommerceContext`. En lugar de sobrecargar el método `OnModelCreating` con configuraciones detalladas de cada entidad, hemos optado por mantener una estructura limpia y modular. La clave primaria de la entidad `Session` se ha configurado utilizando Fluent API en su propia clase de configuración `SessionConfiguration`, promoviendo la coherencia y facilitando la expansión de configuraciones adicionales si es necesario. Además, hemos utilizado `ApplyConfigurationsFromAssembly` para automatizar la aplicación de configuraciones de todas las entidades presentes en el ensamblado actual, mejorando la escalabilidad y reduciendo la complejidad del código. Este enfoque se alinea con las mejores prácticas de diseño y proporciona una base sólida para el mantenimiento y la evolución continua de nuestra aplicación.

Errores Conocidos:

- No se crearon estructuras de categoría, color o marca en forma de tablas; en su lugar, manejamos estos valores como cadenas de texto individuales.
- Con respecto al color, se estableció previamente que cada producto solo puede tener un color seleccionado, no permitiendo la selección de más de uno.
- Inicialmente, se tenía la intención de crear un punto final (endpoint) para las promociones, pero no se logró concretar, por lo que las promociones quedaron representadas como una clase dentro del dominio, con su propio archivo de lógica independiente.
- La integración de todo lo implementado en promociones con lo relacionado a las compras no pudo ser realizada.