

Universidad ORT Uruguay
Facultad de Ingeniería

Diseño Aplicaciones 2
Obligatorio 2

Entregado como requisito para el Obligatorio 2 de Diseño de Aplicaciones 2

Diego Acuña – 222675

Felipe Brioso - 269851

Nicole Uhalde – 270303

Profesores:

Francisco Bouza

Santiago Tonarelli

Juan Irabedra

2023

Repositorio: <https://github.com/IngSoft-DA2-2023-2/222675-269851-270303>

Índice

Descripción general del trabajo	4
Mejoras con respecto a la primera entrega	5
Errores y posibles mejoras	5
Vista de implementación	6
Diagrama general de paquetes (namespaces)	6
Diagrama de paquetes y arquitectura de la solución	6
Descripción de las capas y los paquetes:.....	6
Diagrama de paquetes utilizando nesting	7
Diagrama de componentes.....	7
Vista lógica	8
Backend: Justificaciones del diseño y diagrama de clases por paquete	8
Domain	8
Utilities	9
BusinessLogic.....	9
LogicInterface	10
DataAccess.....	11
DataAccessInterface.....	11
ApiModels	12
WebApi	13
ServiceFactory	14
Proyectos de promociones.....	14
Promotion20off.....	14
Promotion3x1Fidelity	15
Promotion3x2.....	15
PromotionTotalLook	15
Modelo de tablas de Base de datos	16
Patrones de diseño utilizados e inyección de dependencias	17
Solid paquetes	19
REP.....	19
CCP CRP	19
Reflection	21
Mecanismos utilizados	21
Agregar/Desactivar promociones	21

Frontend: Principales decisiones de diseño	22
Componentes y servicios creados	22
Decisiones de diseño	22
Modificación de la sesión para mejorar la UX	22
Endpoints nuevos como mejora de UX	23
Vista de proceso	24
Vista de despliegue	25
Diagrama de despliegue	25
Interpretaciones de letra	26
Anexo	26
API cambios	26
Cobertura de tests	27
Algunas aclaraciones	28
Gitflow y metodología de trabajo	28
Postman	29
Pasos para probar la solución	29

Descripción general del trabajo

La aplicación entregada cumple con todos los requerimientos solicitados. Esto incluye:

Para todos los usuarios:

- Sistema de roles: Un usuario puede tener 0 o más roles, aunque actualmente solo hay roles de: admin y comprador(buyer).
- Creación de una nueva cuenta (creado por el mismo usuario), que asigna por defecto el rol de comprador.
- Creación de una nueva cuenta por parte del admin (que asigna los roles a su gusto).
- Log in si un usuario se encuentra ya registrado.
- Log out si un usuario está registrado y quiere terminar su sesión.
- Una opción para que los usuarios puedan modificar su propio perfil.
- Visualizar todos los productos, aunque no se esté logeado.
- Poder añadir productos al carrito, aunque no se esté logeado.
- Que no se añadan al carrito productos si se superara el stock disponible, además de notificar en ese caso al usuario.
- Que al recargar el navegador se mantenga la sesión iniciada y el carrito

Para los admins:

- Una opción para que el admin pueda modificar otros perfiles.
- Una opción para que el admin pueda borrar otros perfiles, aunque no puede borrar otros admins (según la respuesta del foro)
- El admin puede acceder a la lista de todos los usuarios registrados y ver sus atributos (menos las contraseñas de cada uno [esto lo hablamos con Santi]).
- Un admin puede ver las compras de todos los compradores, incluso habiéndose borrado dicho usuario.
- Un admin puede ver los productos creados y todos sus datos (incluido el stock).
- Un admin puede modificar un producto. Incluso puede hacer que ese producto no sea computable para ninguna promoción.
- Un admin puede crear un nuevo producto.
- Un admin puede filtrar los productos por un rango de precios y la opción de “excluir de promociones”.
- Un admin no tiene permitido comprar.

Para los compradores

- Poder conocer en tiempo real el valor del carrito (incluyendo el descuento en caso de que aplique una promoción).
- Que los compradores puedan comprar los productos de su carrito con los métodos de pago disponibles: Tarjetas de Crédito (Visa, MasterCard), Débitos Bancarios (Banco Santander, Banco ITAU, Banco BBVA), Paypal, Paganza.
- Un comprador puede ver su carrito y editarlo.
- Un comprador puede ver su historial de compras.
- Validación de disponibilidad del stock al momento de pagar.

Para la empresa (los que manejan el servidor)

- Inicialmente se encuentran 4 promociones disponibles, aunque se puede activar y/o desactivar una promoción sin tener que recompilar la solución.
- Se pueden agregar nuevas promociones sin tener que recompilar la solución.

Mejoras con respecto a la primera entrega

1. Un problema documentado en la entrega anterior fue que EF al realizar el mapeo a tablas, trataba a cada producto comprado como un producto diferente (mismos datos, pero distinto guid), por lo que al hacer un GET de productos era imposible diferenciar si era el producto original o uno comprado por un comprador, lo cual para esta entrega hacía imposible llevar al día el stock y la validez para promoción.

Para esta entrega, logramos diferenciarlos y traer únicamente los productos originales, basándonos en el diseño de tabla de EF y que en caso de los productos originales, existe una foreign key que es nula.

2. Respecto a la indentación, decidimos que respetaríamos que sea un salto de línea entre métodos, mientras que, dentro de un método, los saltos de línea son consecutivos. En el caso de líneas muy extensas, se realiza un salto de línea para facilitar la legibilidad. Además, para las migraciones, hemos decidido denominarlas usando el formato 'NumeroVersion_lo_que_se_hizo_nuevo'

Errores y posibles mejoras

Con respecto a errores de funcionalidad, se encontró que, si un comprador tiene productos en el carrito, y estos son modificados, al comprar se guardan con los datos. Esto podría solucionarse modificando el CartController para que este revise si los productos fueron modificados y notifique al usuario en ese caso.

Otro defecto es que, si el stock se modifica a cero y se tiene al producto en el carrito, si bien no se permitirá realizar la compra, el mensaje que se le envía al usuario es muy vago y podría mejorarse.

Con respecto a la UI, algunos defectos son:

- La landing page tiene un ancho superior al ancho de la ventana
- Las ventanas de listar productos y usuarios hay botones al final de las listas. En caso de que dichas listas sean muy grandes se tendría que scrollear todo para acceder a dichos botones. Por ende, reubicar dichos botones de manera tal que se posición por encima de los listados

Con respecto al borrado de usuarios, este funciona con normalidad cuando el usuario no tiene compras asociadas. En caso contrario, este pierde su rol de comprador, pero no es borrado. Lo cual es correcto por si queremos recuperarlo y además para poder visualizar sus compras

anteriores. El problema sería la respuesta http brindada por el servidor a la aplicación cliente la cual devuelve un 500 de error.

Vista de implementación

Diagrama general de paquetes (namespaces)

Por problemas de espacio y calidad de las imágenes de los diagramas presentados en todo el documento, se recomienda visualizarlos por fuera, en la carpeta Diagramas.

Diagrama de paquetes y arquitectura de la solución

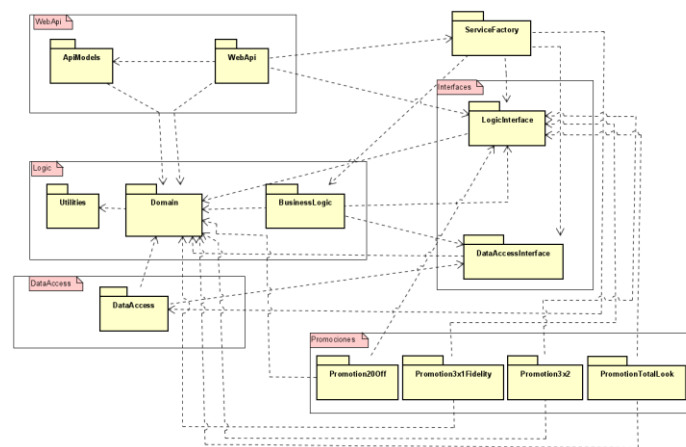


Figura 1: Diagrama de paquetes que muestra las distintas capas del sistema.

El equipo decidió mantener la solución separada en cuatro capas bien diferenciadas según su propósito, pero esta vez agregando otra capa más llamada Promociones donde se encuentran los proyectos de las distintas promociones y sus implementaciones, aunque el motivo de esto será explicado más adelante.

Descripción de las capas y los paquetes:

La primera de ellas es la WebApi. Dentro de esta capa tenemos el paquete de WebApi, que incluye los controladores, sus filtros y los .dll de las promociones activas. A su vez, también tenemos los ApiModels, encargados de la comunicación con el cliente.

En segundo lugar, se encuentra la lógica. El Dominio, BusinessLogic y Utilities son los paquetes que componen esta capa, son clases de alto nivel, encargadas de modelar las entidades. El Dominio es el paquete en el cual se encuentran las entidades del sistema y el cual utiliza el paquete Utilities para validaciones, BusinessLogic se encuentra la lógica de negocio.

En tercer lugar, está la capa de DataAccess. Es la capa encargada de manejar el contexto y la persistencia de datos del sistema.

En cuarto lugar, los paquetes LogicInterface y DataAccessInterface contienen las interfaces de sus respectivos paquetes. LogicInterface para interfaces de la lógica y DataAccessInterfaces para interfaces de los repositorios. Esto fue realizado con el fin de ocultar la implementación y por ende minimizar el impacto al cambio.

Con respecto a cómo están conformados los paquetes de la figura 1, a continuación, se presenta un diagrama de ello:

Diagrama de paquetes utilizando nesting

Para una mejor organización del sistema se decidió separar en subpaquetes cada paquete. Se trato de utilizar nombres claros para facilitar la comprensión y entender la función de cada uno.

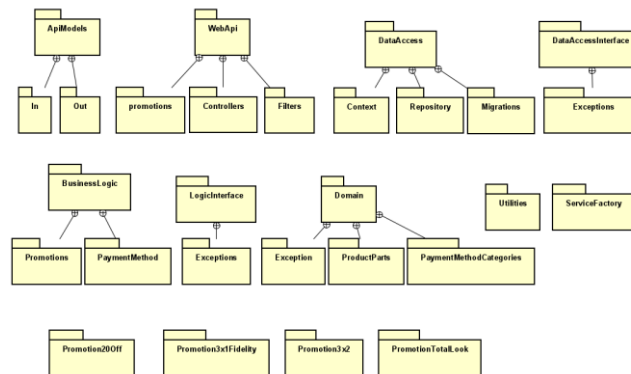


Figura 2: Diagrama de paquetes usando nesting

Diagrama de componentes

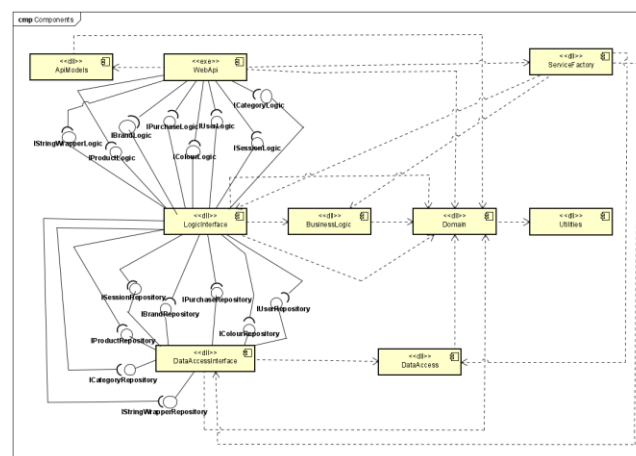


Figura 3: Diagrama de componentes

En el diagrama previo, se evidencia la relación entre los componentes de la aplicación. Se destaca, en primera instancia, la utilización de interfaces entre DataAccessInterface, LogicInterface y WebApi. Este enfoque se implementó con el objetivo de mejorar la separación entre los componentes, aspecto que será detallado más adelante.

Vista lógica

Backend: Justificaciones del diseño y diagrama de clases por paquete

Domain

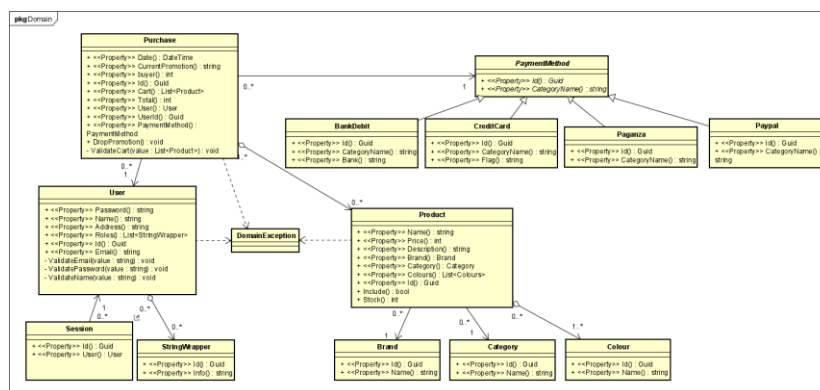


Figura 4: Diagrama de paquetes - Domain

El diagrama anterior se encuentran las clases del dominio, las cuales la mayoría siguen siendo las mismas de la primera versión, agregando las nuevas de Payment Method. Además, se encuentran la clase de DomainException, se optó por crear una excepción por capa de la solución, esto con el fin de poder saber de dónde viene el error y poder manejarlo de mejor manera.

Mantenemos las siguientes decisiones de diseño:

- Aunque el correo electrónico del usuario es único, existe la posibilidad de que cambie en el futuro. Por lo tanto, en lugar de utilizar claves que puedan cambiar, se ha decidido emplear un identificador único (Guid).
- Para simplificar la gestión de la base de datos, se ha incorporado un identificador (Id) en todas las demás entidades.
- El equipo ha creado una clase llamada "Session" para gestionar los inicios de sesión y cierres de sesión. Inicialmente, se consideró utilizar el identificador del usuario como token de acceso, pero esta no resultó ser la mejor opción. En su lugar, se ha optado por crear una nueva entidad que incluye un ID de sesión. Esto permite a un usuario conectarse en múltiples dispositivos sin afectar las sesiones de los otros dispositivos.
- Se ha introducido la clase "StringWrapper" para abordar la situación en la que el usuario tenía una lista de roles almacenados como cadenas de texto. Dado que Entity Framework

no reconocía esta estructura, se ha creado una nueva clase que representa los roles, y se le ha dado este nombre para reflejar su propósito y fomentar la reutilización de código.

- Aunque no se han representado relaciones con clases de otros paquetes en el diagrama, es importante destacar que varios de estos paquetes utilizan la funcionalidad proporcionada por el paquete "Utilities" para llevar a cabo validaciones.

Las nuevas decisiones de diseño que se tomaron para esta versión son las siguientes:

- Se decidió tener una clase abstracta `PaymentMethod` en donde las distintas categorías de métodos de pago como `BankDebit`, `CreditCard`, `Paganza` y `Paypal` hereden de ella, y en el caso de `BankDebit` y `CreditCard` agregando los atributos `Bank` y `Flag` respectivamente. Para hacer la estructura el equipo decidió utilizar el patrón de diseño `Template Method`.

Utilities

Sabemos que, aunque anteriormente el uso de "Utilities" era una práctica común en la industria, ha perdido popularidad en la actualidad. En nuestro caso, esta carpeta contiene una única clase destinada a facilitar la realización de validaciones en el dominio, como las relacionadas con el formato de direcciones de correo electrónico o contraseñas. De esta manera, conseguimos mejorar el Principio de Responsabilidad Única (SRP) y promover la reutilización de código. Además, consideramos que es una opción más organizada en lugar de incorporar directamente estos métodos en las clases que los requieran.

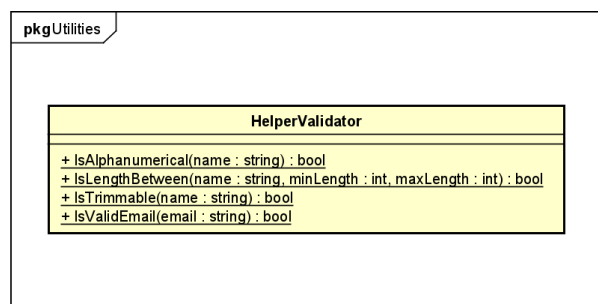


Figura 5: Diagrama de paquetes - Utilities

Los métodos de esta clase son todos estáticos ya que la clase no tiene atributos propios. Sus métodos son utilizados por clases externas y su comportamiento es el de auxiliar a otras clases a realizar validaciones.

BusinessLogic

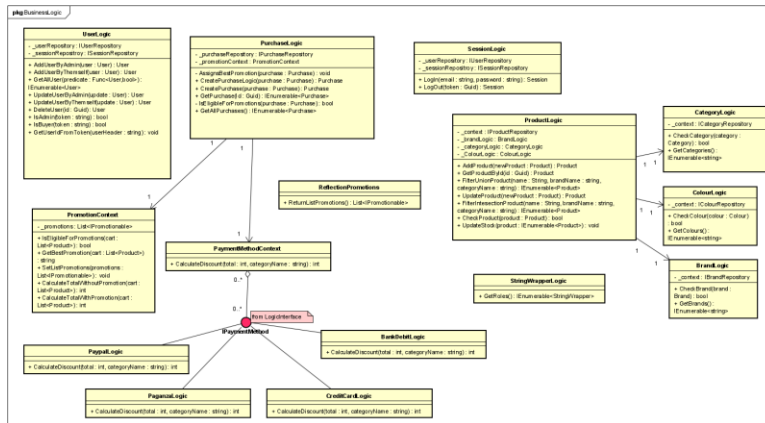


Figura 6: Diagrama de paquetes - BusinessLogic

El diagrama anterior se encuentran las clases que representan la lógica del sistema. Se agregaron y se hicieron varios cambios, los cuales son los siguientes:

- Se añadió ReflectionPromotions el cual se explicará más adelante en profundidad, pero resumidamente devuelve una lista con las promociones activas actuales del sistema, que luego será la que se use dentro de PromotionContext. Por ende, la lógica de las diferentes promociones dejaron de estar en BusinessLogic para pasar a tener sus propios .dll, así se podrán desactivar y activar todas las promociones que cumplan con la interfaz IPromotionable
- Se añadió la clase StringWrapperLogic para la función “GetRoles()” la cual se añadió también para CategoryLogic, ColourLogic y BrandLogic con sus respectivos nombres, esto se hizo para mejorar la UX del usuario, aunque se profundizará más adelante.
- El equipo decidió que la mejor solución para la lógica de los PaymentMethods es el patrón de diseño Strategy, donde se encuentra el PaymentMethodContext, y las diferentes categorías de métodos de pago PayPalLogic, PaganzaLogic, CreditCard y BankDebit.

La interfaz que se encuentra en rosa, se encuentra diferenciada de las demás entidades debido a que no pertenece a este paquete. No obstante, se reconoció que las categorías de método de pago tenían los mismos métodos, por lo que se decidió que implementaran una interfaz

LogicInterface

Este paquete contiene las interfaces que son implementadas por BusinessLogic. Además, tiene la clase que crea las excepciones a nivel de lógica que también usan las clases de BusinessLogic. Se utilizaron interfaces con el fin de minimizar el impacto a cambio y de favorecer DIP.

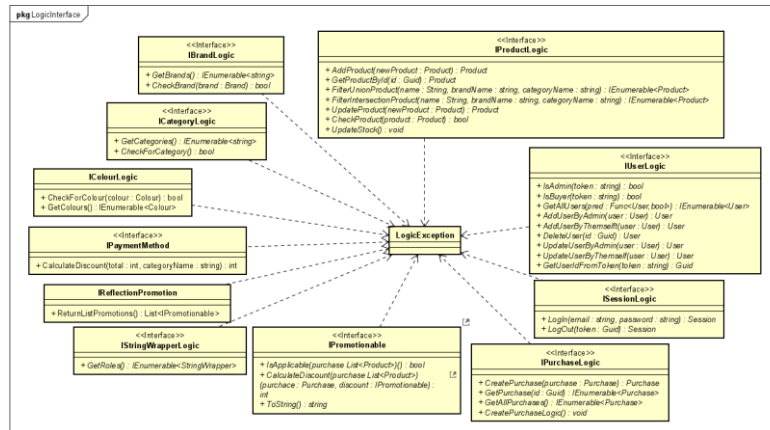


Figura 7: Diagrama de paquetes - LogicInterface

DataAccess

Contiene el contexto y la implementación de los repositorios. En este paquete no se tomaron mayores decisiones de diseño.

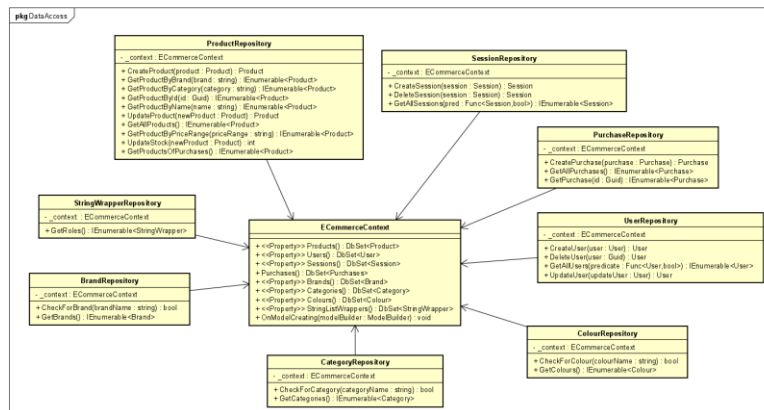


Figura 8: Diagrama de paquetes - DataAccess

DataAccessInterface

Contiene las interfaces de DataAccess y una clase encargada de crear excepciones relacionadas con la base de datos. Fue diseñado con el objetivo de promover el Principio de Inversión de Dependencias (DIP).

Se utilizaron interfaces por diversos motivos. En primer lugar, nos permitió generar un nivel de abstracción al definir contratos. En segundo lugar, porque promueve el polimorfismo. Por último, utilizando interfaces, podemos mockear su comportamiento, lo cual nos permite implementar tests de capas superiores sin tener que implementar las capas de niveles inferiores.

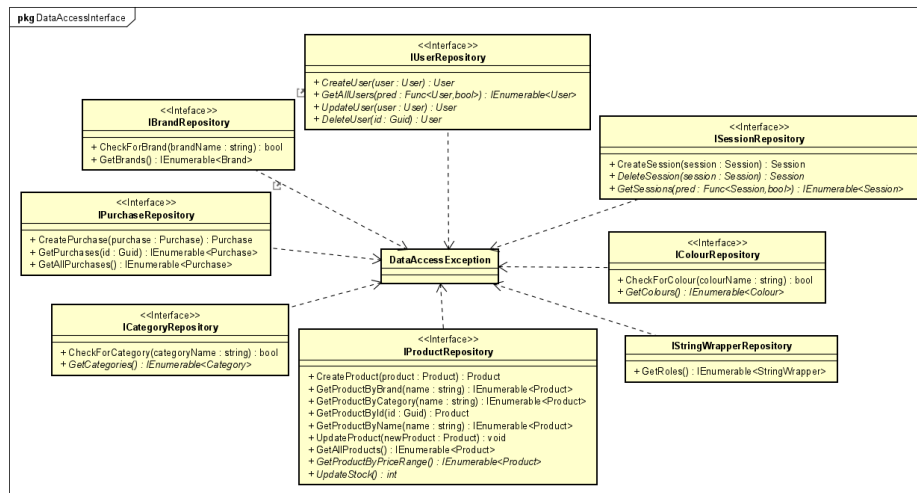


Figura 9: Diagrama de paquetes - DataAccessInterface

ApiModels

Contiene todos los modelos in y out de la aplicación. Se encarga de recibir las requests y devolver las respuestas. Se realizó para cumplir con SRP y de poder elegir qué datos recibir y retornar.

En el diagrama que sigue, se puede notar que la mayoría de las clases constan principalmente de propiedades. Por lo tanto, se optó por implementar varias de estas entidades como structs.

Además, en algunos de los requests, se puede observar la presencia del método "ToEntity()". Esto se hace con el propósito de transformar las solicitudes en clases del dominio. De esta forma, la lógica recibe un objeto del dominio, generando dependencias entre la lógica y las clases del dominio, cumpliendo DIP.

Por otro lado, si se hubiera optado que la lógica recibiera directamente una clase de request, se habría generado una dependencia desde la lógica hacia los Modelos de API, lo cual habría infringido el DIP.

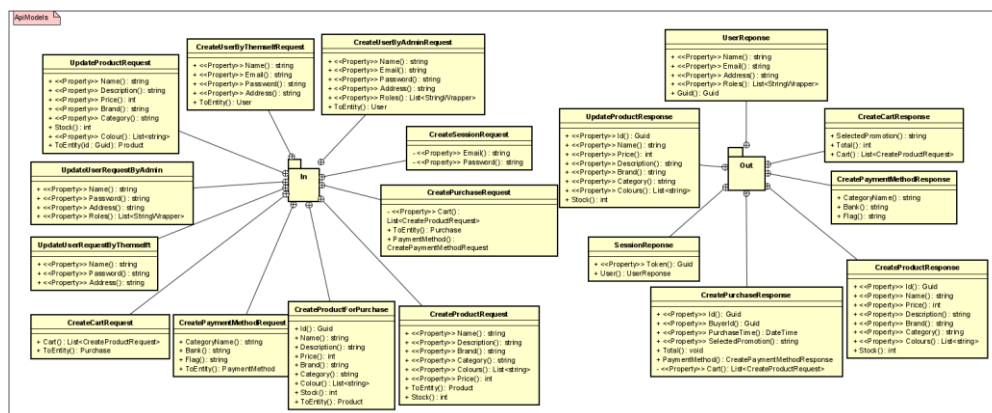


Figura 10: Diagrama de paquetes – ApiModels

Debido a que utilizamos una herencia en PaymentMethod al traerlos o mandarlos en las clases de los modelos de in y out del método de pago hacemos uso del typeof y de un switch para poder modelarlos.

```
1 reference | diegoacutiz34, 3 days ago | 3 authors, 3 changes
public PaymentMethod ToEntity()
{
    switch (CategoryName)
    {
        case "CreditCard":
            return new CreditCard
            {
                CategoryName = CategoryName,
                Flag = Flag
            };
        case "BankDebit":
            return new BankDebit
            {
                CategoryName = CategoryName,
                Bank = Bank
            };
        case "Paganza":
            return new Paganza()
            {
                CategoryName = CategoryName,
            };
        case "PayPal":
            return new Paypal()
            {
                CategoryName = CategoryName,
            };
        default:
            throw new Exception("Invalid Payment Method Category");
    }
}
```

Figura 11: PaymentMethod ToEntity()

```
public CreatePaymentMethodResponse(PaymentMethod paymentMethod)
{
    CategoryName = paymentMethod.CategoryName;
    if (paymentMethod.GetType() == typeof(BankDebit)) Bank = ((BankDebit)paymentMethod).Bank;
    if (paymentMethod.GetType() == typeof(CreditCard)) Flag = ((CreditCard)paymentMethod).Flag;
}
```

Figura 12: PaymentMethod CreatePaymentMethodResponse

WebApi

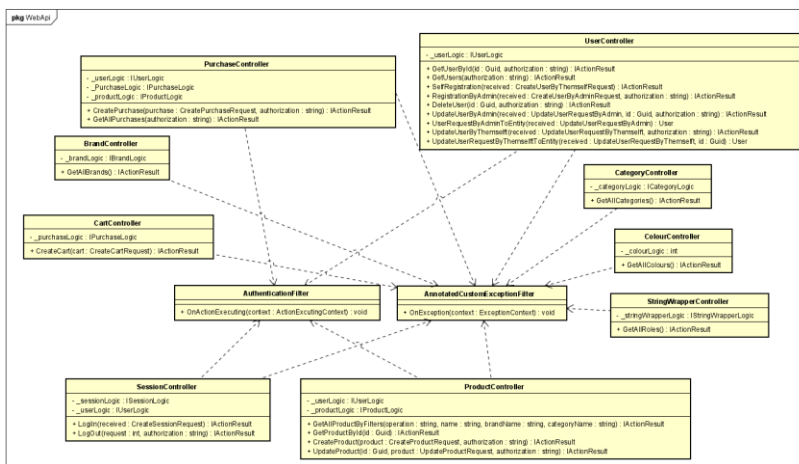


Figura 13: Diagrama de paquetes - WebApi

En cuanto a los filtros, eliminar los bloques try-catch de los controladores mejoró la legibilidad. En relación con AnnotatedCustomExceptionFilter, evalúa el tipo de excepción y devuelve una respuesta diferente según dicho tipo. Aunque reconocemos que consultar por RTTI no es una buena práctica, debido a la escasa cantidad de excepciones manejadas, no consideramos que valiera la pena resolverlo mediante polimorfismo.

Para el filtrado de productos el equipo tomo la decisión que, en cuanto al filtrado por precio, por más que el usuario coloca int, el pasaje al backEnd se hace a través de un string para: enviar 1 solo parámetro en vez de 2 y como sabemos que este se envía siempre en el mismo formato (precioDesde-precioHasta) cuando llega a la base de datos es cuando lo separamos. Somos conscientes que posiblemente no es lo más ideal, pero por temas de tiempo el equipo decidió mantener esta implementación.

ServiceFactory

Tiene únicamente la clase que se encarga de las inyecciones de dependencia de las interfaces y de marcar el connection string con la base de datos.

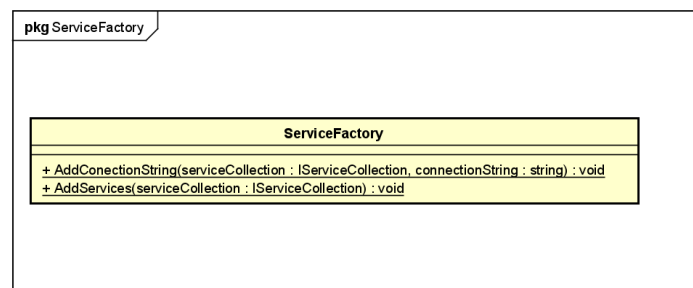


Figura 14: Diagrama de paquetes - ServiceFactory

Proyectos de promociones

Para que cada promoción este en un .dll distinto, el equipo decidió separar las promociones existentes en proyectos distintos. Cada proyecto de estos incluye únicamente una clase que contiene la lógica de la promoción: si aplica y el cálculo del descuento.

Promotion20off

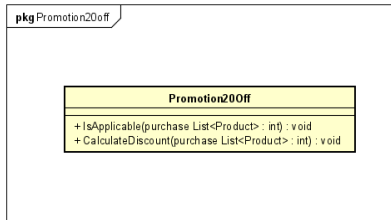


Figura 15: Diagrama de paquetes – Promotion20off

Promotion3x1Fidelity

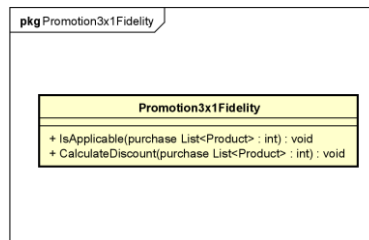


Figura 16: Diagrama de paquetes – Promotion3x1Fidelity

Promotion3x2

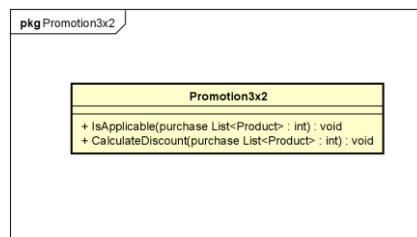


Figura 17: Diagrama de paquetes – Promotion3x2

PromotionTotalLook

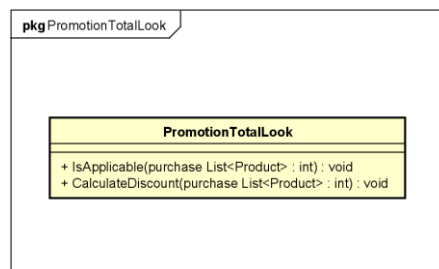


Figura 18: Diagrama de paquetes – PromotionTotalLook

Modelo de tablas de Base de datos

La base de datos fue modelada mediante *Entity Framework*. A continuación, se muestra un diagrama que representa las tablas resultantes.

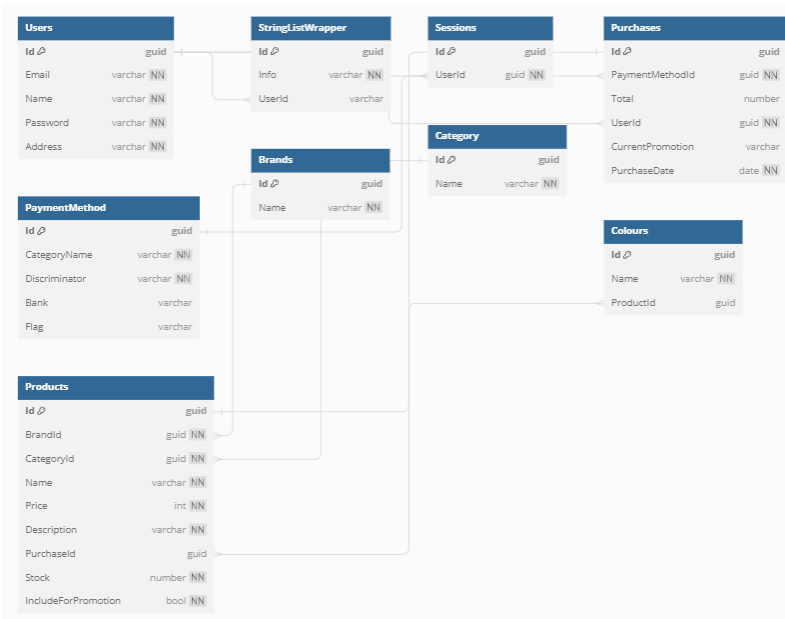


Figura 19: Modelo de tablas de Base de datos

El diseño de la base de datos se ha realizado mediante el modelo Code First, centrándose exclusivamente en modelar las clases del dominio. En el diagrama se pueden visualizar las tablas, sus atributos y tipos respectivos. Los atributos etiquetados con "NN" a la derecha indican que no pueden ser nulos. Es importante señalar que el diagrama no especifica qué atributos deben ser únicos ni la cardinalidad de las relaciones. Para obtener estos detalles, se recomienda consultar [este enlace](#)

Al examinar el modelado, se destacan ciertas relaciones que llaman la atención al convertirse en tablas de Entity Framework. Por ejemplo, que Product tenga su id de compra, aspecto que como se mencionó en el apartado de mejoras, nos trajo diversas complicaciones.

Otro aspecto para remarcar es el criterio utilizado por el equipo para mapear herencias. Dado que hay diversos métodos de pagos y que apenas varían los atributos entre ellos, el equipo resolvió usar tabla por herencia (TPH).

Para ello se tuvo que escribir el siguiente código:


```

0 references | - changes | - authors, - changes
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Paganza>();
    modelBuilder.Entity<Paypal>();
    modelBuilder.Entity<BankDebit>();
    modelBuilder.Entity<CreditCard>();
}

```

Figura 20: Tabla de PaymentMethod

Patrones de diseño utilizados e inyección de dependencias

Patrón Strategy y Template Method

PaymentMethod

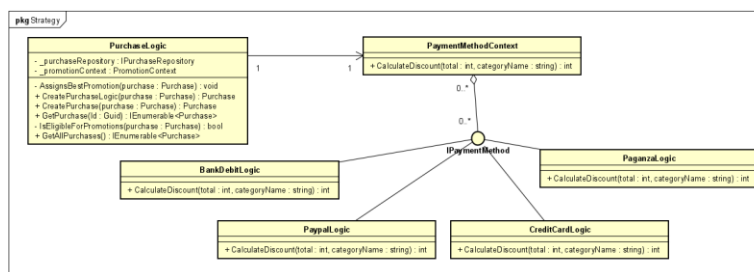


Figura 21: Patrón Strategy para PaymentMethod

Cómo se mencionó anteriormente, se decidió tener una clase abstracta PaymentMethod en donde las distintas categorías de métodos de pago como BankDebit, CreditCard, Paganza y Paypal hereden de ella, y en el caso de BankDebit y CreditCard agregando los atributos Bank y Flag respectivamente. Para hacer la estructura en las entidades del dominio el equipo decidió utilizar el patrón de diseño Template Method.

Aunque sabemos que solo el método de pago Paganza tiene un descuento en particular, en un futuro podría pasar que otro método de pago tenga también, por lo que para respetar el OCP decidimos que la mejor solución era realizar el patrón de diseño Strategy.

Además, dos factores adicionales influyeron en esta decisión. Primero, los métodos de pago no eran configurados por los compradores o administradores, y tampoco se almacenaban en una base de datos. Esto nos llevó a la conclusión de que podíamos codificar estos directamente en la lógica, lo que facilitaría la adición de nuevos métodos en el futuro, simplemente agregándolas y permitiendo que el contexto tuviera acceso a ellas.

En resumen, el patrón Strategy se presentó como la solución idónea para nuestro problema de aplicar un descuento en las compras después de aplicada una promoción, permitiéndonos flexibilidad, fácil mantenimiento y la capacidad de agregar nuevos métodos de pago de manera sencilla y el patrón Template Method se utilizó para diseñar la estructura en las entidades del dominio

Patrón Strategy Promotions

En el proceso de implementar descuentos en nuestras compras, nos encontramos con un desafío crucial. Contábamos con una variedad de algoritmos para calcular descuentos, pero nos enfrentábamos a la dificultad de hacer que estos algoritmos fueran intercambiables, de manera que pudiéramos probar diferentes enfoques durante la ejecución del programa.

Fue en este punto donde identificamos que el patrón de diseño Strategy ofrecía la mejor solución, ya que abordaba precisamente el problema que estábamos enfrentando. Además, dos factores adicionales influyeron en nuestra decisión. En primer lugar, las promociones no eran configuradas por los compradores o administradores, y tampoco se almacenaban en una base de datos. Esta observación nos llevó a la conclusión de que podíamos codificar estas promociones directamente en la lógica del programa. Esto facilitaría la adición de nuevas promociones en el futuro, simplemente agregándolas y permitiendo que el contexto del programa tuviera acceso a ellas. La diferencia de la versión uno y esta es que ahora implementamos esto con reflection que se explicará más adelante.

En resumen, el patrón Strategy se reveló como la solución óptima para nuestro desafío de aplicar descuentos en las compras. Nos brindó flexibilidad, facilitó el mantenimiento y nos permitió agregar nuevas promociones de manera sencilla.

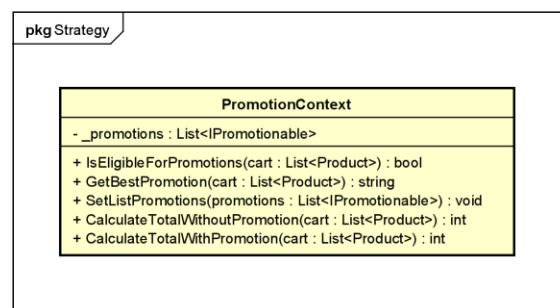


Figura 22: Patron Strategy para PromotionContext

Inyección de dependencias

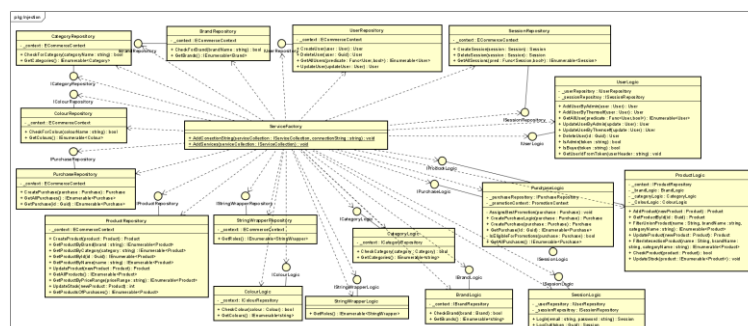


Figura 23: Inyección de dependencias

Optamos por implementar la inyección de dependencia en diversas secciones del proyecto obligatorio. En particular, establecimos esta práctica entre el acceso a datos y la lógica, utilizando sus respectivas interfaces. Esta elección ha mejorado el desacoplamiento, la reusabilidad, la mantenibilidad y la flexibilidad del sistema, al mismo tiempo que ha cumplido con el Principio de Inversión de Dependencia (DIP). Esto se logró al hacer que las clases de alto nivel dependieran de interfaces en lugar de depender directamente de clases concretas.

Solid paquetes

REP

Dado que en el proyecto se manejan frameworks externos de versionado (EF y moq) y las entregas funcionan con versionados cumplimos con la granularidad así que podemos afirmar que se cumple REP.

CCP CRP

Dado que se utiliza un modelo de capas e interfaces todas las clases están colocadas correctamente en cada paquete siguiendo con el modelo de arquitectura de capas.

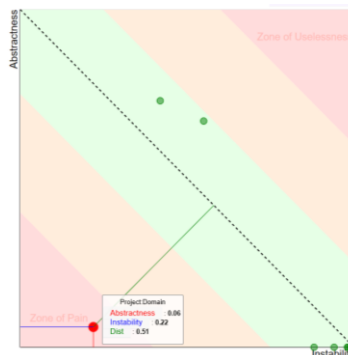


Figura 24: Grafica de distancia(D) NDepend

Vemos que solo el dominio no se encuentra en la zona verde, sino en la 'zone of pain', esto se debe a que cualquier cambio en el dominio impacta de gran forma al resto de la aplicación lo que la hace muy inestable. El resto de los paquetes están en un punto ideal.

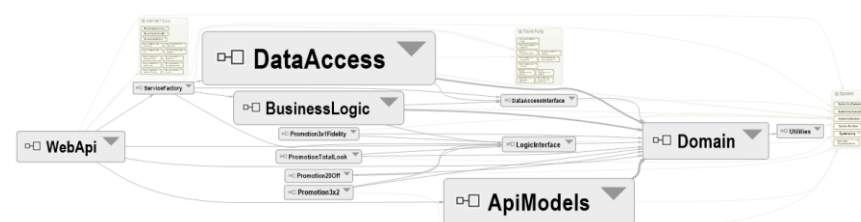


Figura 25: Dependencias entre assemblies

Si miramos la matriz de dependencias entre paquetes como se observa a continuación, vemos que se toma en cuenta las dependencias con las clases del sistema para el cálculo de métricas. Si bien la diferencia con el valor real no es muy distinta, el equipo partió de la siguiente figura, y se realizaron los cálculos correspondientes excluyendo dichas clases.

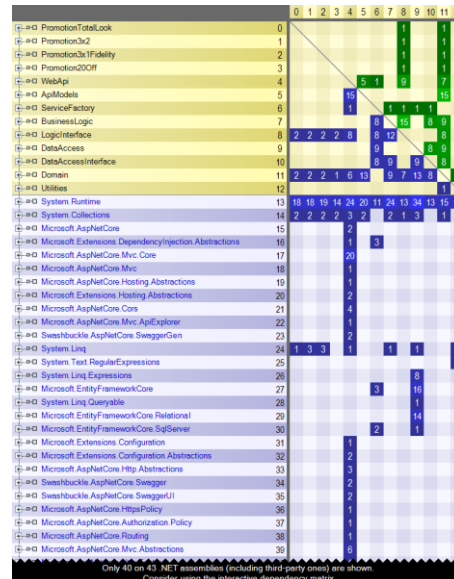


Figura 26: Matriz de dependencias

El resultado final fue:

	CA	CE	I	A	D'
utilities	1	0	0	0	1
domain	60	1	0,01639344	0,06	0,92360656
dataAccess	1	17	0,94444444	0	0,05555556
dataAccessInterface	17	8	0,32	0,67	0,01
LogicInterface	29	8	0,21621622	0,73	0,05378378
BusinessLogic	1	32	0,96969697	0	0,03030303
ServiceFactory	1	4	0,8	0	0,2
ApiModels	5	15	0,75	0	0,25
WebApi	0	22	1	0	0
Promotion200Off	0	2	1	0	0
Promotion203x1Fidelity	0	2	1	0	0
Promotion3x2	0	2	1	0	0
PromotionTotalLook	0	2	1	0	0

Figura 27: Tabla de SDP y SAP

Vemos que SDP no se cumple del todo. Esto se debe a que por ejemplo 'ServiceFactory' con $I=0,8$ utiliza 'BusinessLogic', de $I=0,97$. El equipo es consciente de que no es lo ideal y que podría mejorarse, aun así, consideramos que dada que la diferencia no es tan grande lo vemos como un detalle menor.

Como mencionamos anteriormente el dominio no cumple con SAP por lo ya explicado. El resto de los paquetes se encuentran en un buen punto de D'.

ADP sabemos que se cumple dado que por un lado el código compila (lo que nos evita dependencias directas) y a su vez nuevamente debido a que se construye en capas y el uso de inyecciones por parte del servicio.

Reflection

Mecanismos utilizados

Para esta versión, se podrá desactivar o activar promociones en el sistema durante la ejecución del mismo sin necesidad de recompilar la aplicación. Para lograr esto, utilizamos reflection, particularmente la funcionalidad de plugins. Contamos con una clase llamada ReflectionPromotions dentro del paquete BusinessLogic que implementa el método de la interfaz IReflectionPromotions ReturnListPromotions() el cual devuelve una lista de las promociones actuales activas. Para hacer eso, el método busca los .dll dentro de la carpeta promotions dentro del proyecto WebApi que cumplan con la interfaz de IPromotionable.

Una vez encontrados, devuelve la lista con todas las promociones activas.

Agregar/Desactivar promociones

Para poder agregar nuevas promociones estas deberán seguir con la interfaz de IPromotionable, la cual cuenta con los siguientes métodos:

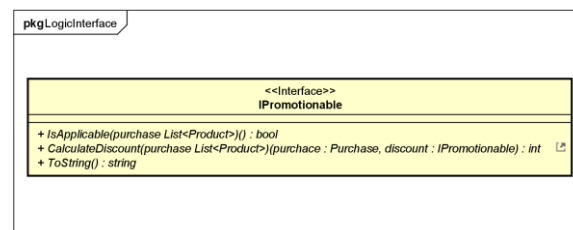


Figura 28: Interfaz IPromotionable

IsApplicable es una función que recibe una lista de productos y devuelve un booleano, “true” si esa promoción es aplicable con los productos que se reciben, y “false” si no lo es.

CalculateDiscount, a partir de que IsApplicable retorna “true” recibe la lista de productos y calcula el descuento que aplica según la promoción.

Por último, tenemos ToString que simplemente retorna el nombre de la promoción, ejemplo: “20% off”, con el fin de poder conocer qué promoción fue aplicada.

Para activar o desactivar promociones, se tendrán que agregar o eliminar (según corresponda) los .dll dentro de la carpeta de promotions, ubicada en el ejecutable.

De esta manera incluso durante la ejecución del sistema se podrá activar y desactivar promociones.

Frontend: Principales decisiones de diseño

Componentes y servicios creados

El proyecto cuenta de 15 componentes y 3 servicios que fueron creados con el fin de fomentar el reuso y la modularidad del código.

En la primera fase del proyecto se utilizó como criterio que cada ruta posea un único componente. Sin embargo, con el avance de este detectamos que había componentes que tenían más de una responsabilidad, por lo que decidimos dividirlos en componentes más pequeños y fomentando el reuso en casos donde apenas había cambios, como por ejemplo en la barra de navegación.

Con respecto a los servicios, estos se utilizaron con el fin de pasar datos entre diferentes componentes. La decisión de utilizar un servicio se basó en que los componentes que debían comunicarse no estaban relacionados (ni padre e hijo, ni hermanos).

Decisiones de diseño

Modificación de la sesión para mejorar la UX

Cuando empezamos a diseñar los primeros bocetos de la UI, queríamos que cuando un usuario se logue, aparezca un mensaje del estilo: "Welcome nombreDelUsuario". El problema de eso era que era imposible devolver eso sin modificar el endpoint de sesión (antes solo devolvía el userId y el token de la sesión), por lo que se modificó la response de session para que devuelva más datos del usuario y poder mostrar su nombre.

Además de ello, devolver toda la info del usuario nos facilitó el control de acceso a los recursos.

Dado que al loguearse podíamos conocer los roles del usuario, podíamos mostrarle únicamente con lo que tiene permitido interactuar. A continuación, se muestra la navigation bar para un comprador y para un admin:

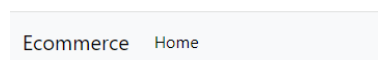


Figura 29: Navbar para no admins

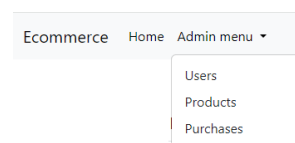


Figura 30: Navbar para admins

Si bien con lo planteado anteriormente limitamos el acceso vía UI a otras rutas a solo usuarios autorizados, en caso de conocer la ruta se podría acceder directamente a ella por vía del navegador. La solución que encontramos fue que en caso de que no se tuvieran los permisos

para acceder a dicha ruta, ser redirigido a la landing page, criterio que también utilizamos en caso de que se ingrese una URL incorrecta.

Endpoints nuevos como mejora de UX

Originalmente cuando se quería seleccionar un rol, categoría, color y marcas era una tarea laboriosa. En primer lugar, debía escribirse igual al valor almacenado en el backend. En segundo lugar, con respecto a los colores y roles, estos debían de ser ingresados seguidos por comas en un text input. Todo esto hacía que la aplicación no fuera fácil de usar. Por ende, se añadieron endpoints para cada uno de los mencionados que traen los datos del backend. De esta manera, las categorías y marcas se seleccionan con un radiobutton y los colores y roles con checkboxes, mejorando considerablemente la usabilidad.

Un aspecto que se debatió mucho en incluir fue incluir el endpoint de roles, ya que por letra entendimos que no es muy común que se añadan nuevos roles. No obstante, para cumplir OCP decidimos incluirlo. De modo que, si se desea crear un nuevo rol, solo habría que hacer un insert en la tabla de roles y ya podría verse desde el front. A continuación, se muestra el formulario para crear un producto con las marcas, categorías y colores traídas desde el backend.

The screenshot shows a web form titled "Create a new product". It contains several input fields and selection options. The "Product Name" field has a red error message "Name must not be empty". The "Description" field has a red error message "Description must not be empty". The "Price" field has a red error message "Must be larger than zero". The "Stock" field has a red error message "Must be larger than zero". Below these are three sections of radio buttons: "Brands" with options Nike, Adidas, and Vans; "Categories" with options Winter, Summer, Shoes, and Spring; and "Colors" with options Black, White, and Blue. There is a checkbox labeled "Include for Promotions". At the bottom are two buttons: "New product" and "Go Back".

Figura 31: visualización de usuario para crear producto

Otro endpoint creado fue el de promociones, esto con el fin de poder mostrar al usuario el precio de su carrito antes de comprar. Para ello era necesario enviar el carrito y recibir el precio final y la promoción en caso de que hubiera.

El problema con ello fue qué verbo usar. Si bien parecería ser obvio que sea un GET, Angular no permite que un GET incluya un body, por lo que, si optábamos por ese camino iba a ser necesario poner una lista de productos como query string, lo cual no es nada prolijo.

La alternativa consistía en emplear el verbo POST y enviar los datos en el body de la solicitud, lo cual resultaba más sencillo. Sin embargo, la utilización de POST es objeto de debate, ya que no se está creando un recurso en el backend, aunque sí una acción. De alguna manera, se está “creando” una promoción específica aplicada a esa lista particular de productos en el carrito.

Ambas propuestas fueron debatidas con Santi y nos sugirió que optáramos por la segunda solución, sugerencia que fue seguida por el equipo.

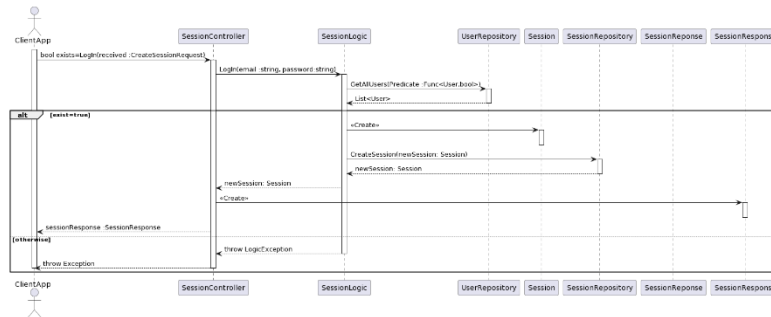


Figura 33: Diagrama de secuencia Log In

Este segundo diagrama es más sencillo y nos muestra el recorrido cuando un usuario quiere logearse (crear una nueva sesión). Nos muestra la existencia del token de sesión que nos parece algo importante mostrar en una aplicación que tendrá muchos usuarios conectados simultáneamente.

También se observa a través del diagrama el diseño vertical de la solución, la clara separación entre las capas y el cumplimiento de Demeter.

Vista de despliegue

Diagrama de despliegue

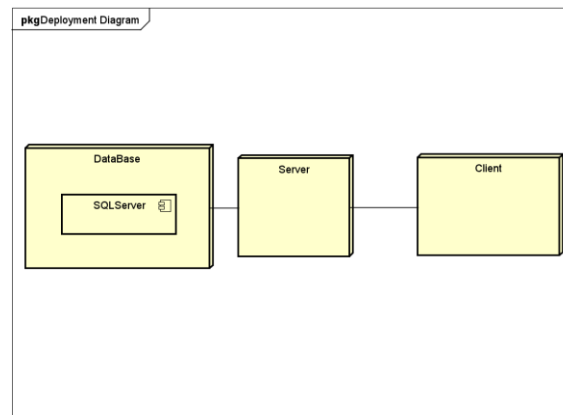


Figura 32: Diagrama de despliegue

Del diagrama se concluye que la arquitectura es cliente-servidor. El servidor se encuentra conectado a una base de datos que utiliza SQL Server.

Interpretaciones de letra

Filtrado por precio

El equipo tomo la decisión de que el filtrado de precio fuera para todos los usuarios, (no solo los admins) dado que el filtrado en sí mismo es para todos.

Address null

El equipo considera que la dirección de los usuarios no es un factor importante a la hora de crear un nuevo usuario. Se tomó la decisión que se permita dejar este atributo en nulo (con la posibilidad de poder colocarse más adelante), dado que la aplicación no cuenta con sistema de envíos donde sería obligatorio una dirección.

Anexo

API cambios

En esta versión, se han incorporado tres controladores adicionales:

1. **BrandController:** Ahora cuenta con el método **GetAllBrands()**, el cual devuelve la lista completa de todas las marcas disponibles.
2. **CartController:** Se ha introducido el método **CreateCart([FromBody] CreateCartRequest cart)**, que sigue la lógica de crear una compra, pero con la particularidad de que la información no se almacena en la base de datos. Este controlador funciona como una visualización de lo que sería la compra en el momento previo a su ejecución, ofreciendo un vistazo sin persistencia de datos.
3. **ColourController:** Integra el método **GetAllColours()**, el cual proporciona un listado de todos los colores existentes.
4. **StringWrapperController:** Ahora incluye el método **GetAllRoles()**, diseñado para recuperar y retornar la lista de roles disponibles.
5. **CategoryController:** Finalmente, se ha incorporado el controlador **CategoryController** con el método **GetAllCategories()**, el cual devuelve todas las categorías existentes.

Estas adiciones amplían la funcionalidad de la aplicación, permitiendo acceder y visualizar información relevante sobre marcas, carritos de compra, colores, roles y categorías sin comprometer la persistencia de datos en el caso del controlador de carritos.

Se han creado nuevos y modificado Models In y Out para los controladores, entre ellos está el **CreateCartResponse** y **CreatePaymentMethodResponse** el cual se hablo previamente.

Cobertura de tests

Se corrió la cobertura y se obtuvo lo siguiente:

hierarchy	Covered (Lines)	Partially Covered (Lines)	Not Covered (Lines)	Covered (%Lines) ▲
diego_DIEGOO_2023-11-15_14_51_44.coverage	1566	19	84	93.83%
webapi.dll	193	3	15	91.47%
WebApi.Controllers	193	3	15	91.47%
UserController.<>c__DisplayClass2_0	0	0	1	0.00%
UserController.<>c	1	0	1	50.00%
UserController	74	3	9	86.05%
PurchaseController	26	0	3	89.66%
ProductController	36	0	1	97.30%
BrandController	7	0	0	100.00%
CartController	10	0	0	100.00%
CategoryController	7	0	0	100.00%
ColourController	7	0	0	100.00%
SessionController	16	0	0	100.00%
StringWrapperController	7	0	0	100.00%
ColourController.<>c	1	0	0	100.00%
StringWrapperController.<>c	1	0	0	100.00%
businesslogic.dll	466	10	30	92.09%
BusinessLogic	360	10	30	90.00%
BusinessLogic.Promotions	68	0	0	100.00%
BusinessLogic.PaymentMethod	38	0	0	100.00%
dataaccess.dll	346	5	20	93.26%
DataAccess.Repository	346	5	20	93.26%
apimodels.dll	370	1	17	95.36%
ApiModels.Out	149	1	14	90.85%
ApiModels.In	221	0	3	98.66%
promotion20off.dll	27	0	1	96.43%
Promotion20Off	27	0	1	96.43%
promotiontotallook.dll	57	0	1	98.28%
PromotionTotalLook	57	0	1	98.28%
utilities.dll	14	0	0	100.00%
Utilities	14	0	0	100.00%
promotion3x2.dll	43	0	0	100.00%
Promotion3x2	43	0	0	100.00%
promotion3x1fidelity.dll	46	0	0	100.00%
Promotion3x1Fidelity	46	0	0	100.00%
logicinterface.dll	2	0	0	100.00%
LogicInterface.Exceptions	2	0	0	100.00%
dataaccessinterface.dll	1	0	0	100.00%
DataAccessInterface.Exceptions	1	0	0	100.00%
domain.dll	1	0	0	100.00%
Domain.Exceptions	1	0	0	100.00%

Figura 33: Cobertura de test

Se logró que las pruebas cubran prácticamente el 94% del código escrito y que en cada proyecto la cobertura no sea menor al 91%.

Con respecto a qué pasó con el 6% de código restante que no fue probado, presentamos algunos motivos de ello:

1) Funcs en Moq:

Una explicación de porqué varias líneas no se cubrieron puede ser explicado por el uso de delegates en Moq. La única opción que encontró el equipo para mockear funcs fue la que se observa marcado en azul en la siguiente imagen:

```

[TestMethod]
[ExpectedException(typeof(LogicException))]
public void CreateUserByThemselfWithExistingEmailThrowsLogicException()
{
    Mock<ISessionRepository> sessionRepo = new Mock<ISessionRepository>(MockBehavior.Strict);
    sessionRepo.Setup(logic => logic.CreateSession(It.IsAny<Session>())).Returns(session);

    Mock<IUserRepository> repo = new Mock<IUserRepository>(MockBehavior.Strict);
    repo.Setup(logic => logic.CreateUser(It.IsAny<User>())).Returns(expected);
    repo.Setup(logic => logic.GetAllUsers(It.IsAny<Func<User, bool>>())).Returns(new List<User> { expected });
    var userLogic = new UserLogic(repo.Object, sessionRepo.Object);

    var result = userLogic.AddUserByThemself(expected);
}

```

Figura 34: Funcs en Moq

Como consecuencia, la lógica dentro del func no pudo probarse, dando como resultado casos como el siguiente:

```

6 references | 4/4 passing | nuhalde, 1 day ago | 3 authors, 4 changes
public User AddUserByThemself(User user)
{
    try
    {
        if (_userRepository.GetAllUsers(u => u.Email == user.Email).Any())
        {
            throw new LogicException("Existing user with that email.");
        }
        user.Roles = new List<StringWrapper> { new StringWrapper() { Info = "buyer" } };
        user.Id = Guid.NewGuid();
        return _userRepository.CreateUser(user);
    }
    catch (DataAccessException e)
    {
        throw new LogicException(e);
    }
}

```

Figura 35: AddUserByThemself

2) Errores de integración a último momento:

Como es costumbre, la última semana emergieron diversos errores. Luego de hablar con los docentes de tecnologías, el equipo decidió priorizar la funcionalidad. Por ende, para algunos cambios de último momento, por motivos de tiempo, se modificó directamente el código sin realizar la prueba correspondiente.

Algunas aclaraciones

Gitflow y metodología de trabajo

El equipo optó por adherirse en gran medida a los principios de Gitflow, aunque realizó ciertas adaptaciones. En primer lugar, se introdujeron ramas con nombres como "quickfix", "refactor/" o "Fix..." con el propósito de abordar errores o implementar mejoras en el código. Esta estrategia se implementó para agilizar la corrección de código.

A pesar de desviarse ligeramente de la metodología de Gitflow, se mantuvieron varios aspectos clave, como nombrar las ramas de acuerdo con la función que desempeñan, asegurar que las ramas se fusionen tanto hacia como desde la rama principal de desarrollo ("develop"), y reservar únicamente el commit final para la rama principal ("main").

Postman

En esta versión, también incorporaremos la colección de Postman, la cual emplea los datos de la base de datos adjunta. Esto implica que al realizar los endpoints existentes mediante Postman, como la creación de compras, se utilizarán productos que ya se encuentran en la base de datos mencionada.

Pasos para probar la solución

Para realizar el deploy el equipo siguió la guía proporcionada en aulas tanto para la aplicación cliente como para la api.

Se aclararán las respectivas diferencias con lo proporcionado en la guía.

Api:

La primera parte se hizo sin ningún tipo de dificultad siguiendo la guía, primero se compilo en reléase y luego se creó el publish a partir de la consola. Se modifíco el WebDAV.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.webServer>
    <modules runAllManagedModulesForAllRequests="false">
      <remove name="WebDAVModule" />
    </modules>
    <handlers>
      <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModule" resourceType="Unspecified" />
    </handlers>
    <aspNetCore processPath="dotnet" arguments=".\AGT.WebApi.dll" stdoutLogEnabled="false" stdoutLogFile=".\logs\stdout" />
  </system.webServer>
</configuration>
<!--ProjectGuid: E3772596-65C4-4544-8C10-66158F624620-->
```

Figura 36: WebDAV

El connection string se mantuvo dado que se cambiará el día de la defensa. La base de datos se exporto y se colocó junto al publish al nivel de root de github.

Para publicar el publish se descargaron los siguientes archivos que no estaban marcados en la guía:




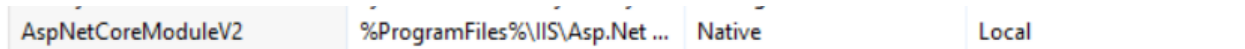
 dotnet-runtime-2.0.0-win-x64	14/11/2023 13:00	Application	22.592 KB
 aspnetcore-runtime-6.0.24-win-x64	14/11/2023 12:36	Application	8.803 KB
 iisexpress_amd64_es-ES	14/11/2023 11:22	Windows Installer ...	11.096 KB

Figura 37: Archivos

Luego de esto ejecutamos el comando “inetmgr” en el run y entramos en module donde deberíamos ver lo siguiente:



La guía muestra una versión anterior a esta pero no la encontramos y la nueva versión funciona sin dificultades.

El resto de la primera guía funciona sin problema.

Extraído de la guía:

A - Copiar los archivos Tomar la carpeta “publish” ubicada en la carpeta de entregables generada en la Parte 1 y pegarla en “C:\inetpub\wwwroot”. Esta carpeta es la carpeta por defecto del IIS, los usuarios de IIS tiene permisos necesarios sobre dicho directorio. Si ya existe una carpeta publish en dicho directorio, pueden renombrar su carpeta antes de pegarla.

B - Volver al IIS y crear un sitio nuevo:

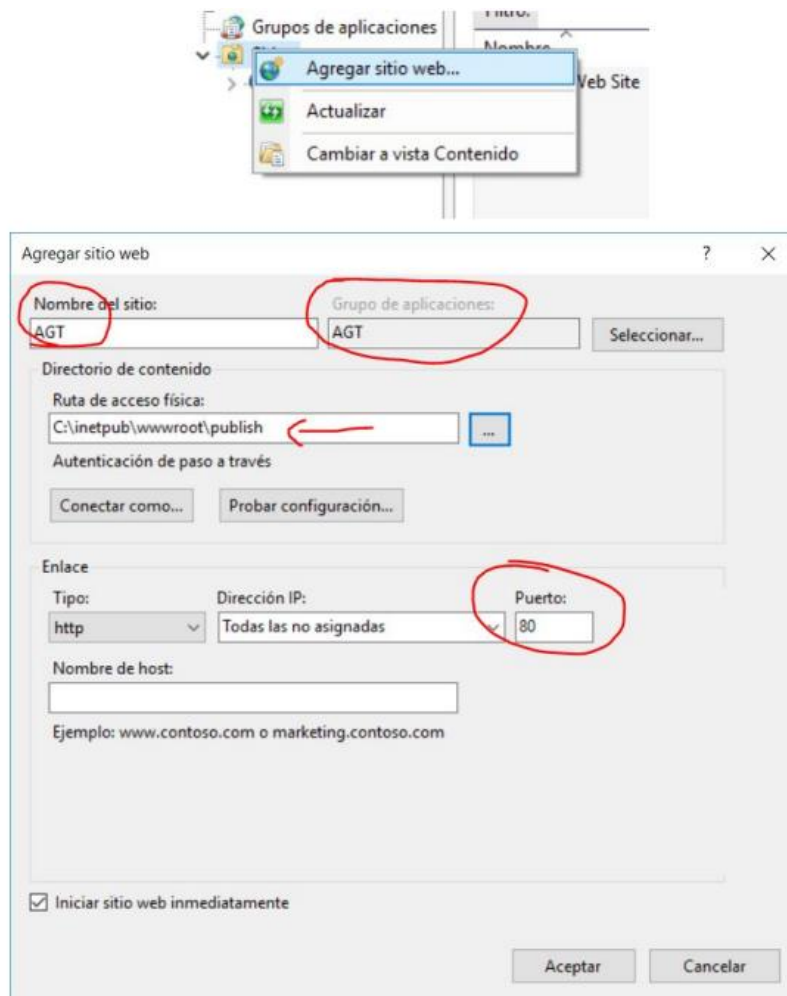


Figura 38

Prestar atención a lo marcado en rojo.

En nuestro caso el nombre es eCommerce

La ruta se mantuvo igual

El puerto seleccionado fue el 7150 para que este no chocara con el puerto 80 por defecto del Default web Site.

C - Quitar CLR por defecto de IIS Cómo ya mencionamos, .NET Core no usa el CLR por defecto configurado en IIS. Para ello debemos configurar nuestro POOL para que no lo utilice

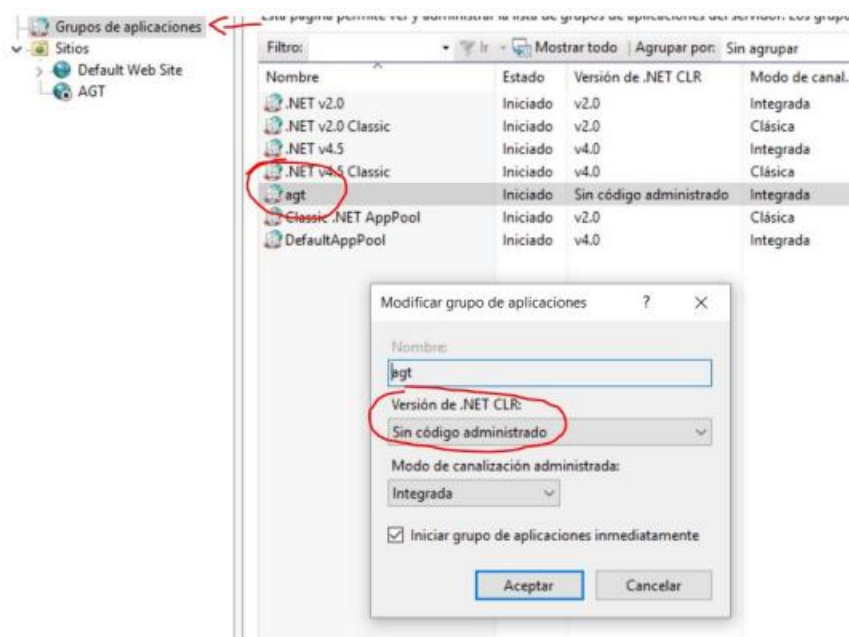


Figura 39

Por ultimo restauramos la base de datos (en nuestro caso nunca se dio de baja), y se creó el nuevo usuario tomando en cuenta nuestros cambios:

4 - Otorgar permisos al usuario del sitio de IIS sobre la base El usuario que interactúa contra el motor de SQL Server es el usuario creado para el sitio de IIS que creamos anteriormente (IIS APPPOOL\XXXX). Debemos entonces configurar un nuevo inicio de sesión en el motor de SQL Server y otorgarle permisos para que pueda leer y escribir de a la base. No nos vamos a detener en explicar los roles y permisos de los usuarios de la base de datos, para esta defensa simplemente crearemos un usuario sysadmin.

A - Ir a la carpeta de Security y crear un nuevo inicio de sesión

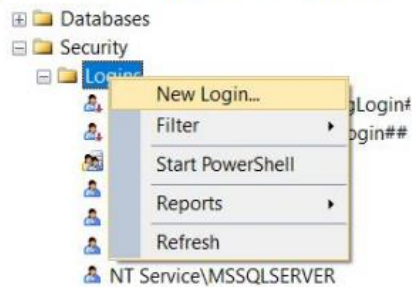
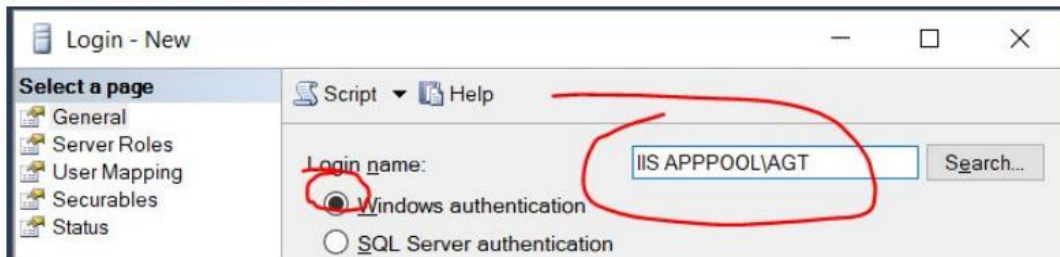


Figura 40

B - El Login Name corresponde a "IIS APPPOOL\[NOMBRE DEL GRUPO DE APLICACIONES DE IIS]" por ejemplo: el sitio que creamos se llama AGT y el grupo de aplicaciones se llama igual. Resultando en lo siguiente:



C - A este nuevo usuario debemos asignarle el ROL de deseado, en nuestro casos sysadmin. Una vez hecho esto, darle OK.

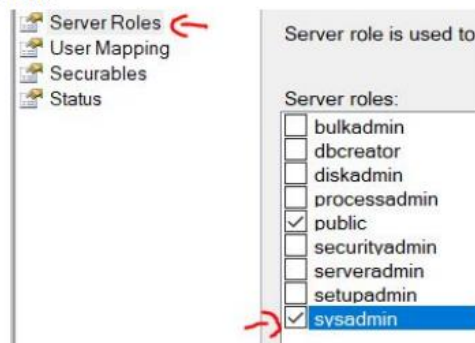


Figura 41

En nuestro caso no hace falta hacer más nada dado que la aplicación se corre desde la misma máquina que el cliente.

Cliente: Se ejecuta ng build (se sacó el "-prod" debido a que fallaba) y esto ya nos genera el deploy dentro de la carpeta generada "dist".

Ya habiendo hecho lo de iis de la primera parte, solamente seguimos la guía para agregar el publish.

A - Copiar la carpeta ubicada en la carpeta de entregables generada en la Parte 1 y pegarla en “C:\inetpub\wwwroot”. Esta carpeta es la carpeta por defecto del IIS, los usuarios de IIS tienen permisos necesarios sobre dicho directorio. Si ya existe una carpeta con el mismo nombre en dicho directorio, pueden renombrar su carpeta antes de pegarla.

B - Volver al IIS y crear un sitio nuevo:

The screenshot shows the 'Add Web Site' dialog box with the following configuration:

- Site name:** AngularApp
- Application pool:** AngularApp
- Content Directory:**
 - Physical path:** C:\inetpub\wwwroot\cityInfo
- Pass-through authentication:** (Buttons: Connect as..., Test Settings...)
- Binding:**
 - Type:** http
 - IP address:** All Unassigned
 - Port:** 80
 - Host name:** (Empty field)
 - Example:** www.contoso.com or marketing.contoso.com
- Start Web site immediately:** ☒

Figura 42

En nuestro caso:

Se llamo: AngularApp

La ruta seria: C:\inetpub\wwwroot\ecommerce-app

Y el puerto utilizado fue el 4200, nuevamente para evitar chocar con otro.

Ahora dentro del main en el publish debemos buscar la ruta para poder enviar cosas a la api:

```
this.url="http://localhost:4200/api",this.currentSession=void 0)updateSession(){this.currentSes
```

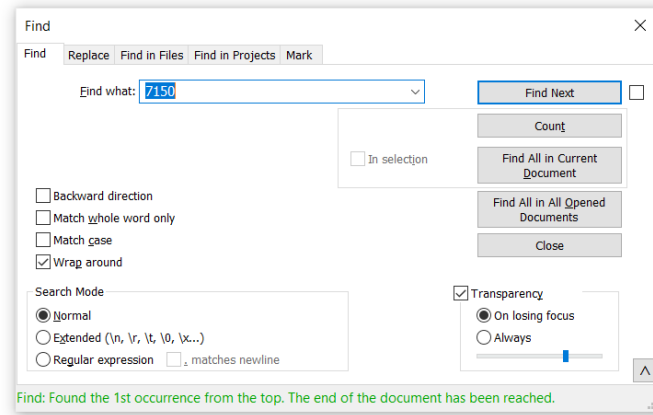


Figura 43

Debido a que fallaba se tuvo que agregar la configuración de IIS ReWrite Module:

1- Agregar archivo web.config. En la carpeta de IIS donde se encuentra la aplicación Angular, crear el archivo web.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
<system.webServer>
<rewrite>
<rules>
<rule name="Angular Routes" stopProcessing="true">
<match url=".*" />
<conditions logicalGrouping="MatchAll">
<add input="{REQUEST_FILENAME}" matchType="IsFile" negate="true" />
<add input="{REQUEST_FILENAME}" matchType="IsDirectory" negate="true" />
</conditions>
<action type="Rewrite" url="./index.html" />
```

```
</rule>  
</rules>  
</rewrite>  
</system.webServer>  
</configuration>
```

Para luego verificar que este el IIS URL Rewrite Module %SystemRoot%\system32\inetsrv\

En esta carpeta deberíamos encontrar rewrite.dll. En nuestro caso no se encontraba y el link proporcionado para su descarga Microsoft ya no lo proporciona. Así que se busco en internet y se llegó al siguiente link: <https://www.iis.net/downloads/microsoft/url-rewrite>

Por último, solo recargamos el sitio.

Dentro del root de la entrega se encuentra una carpeta llamada entrega que contiene:

- Release
 - Publish (backEnd)
 - Ecommerce-app (FrontEnd)
 - Diagramas
 - Documento
-