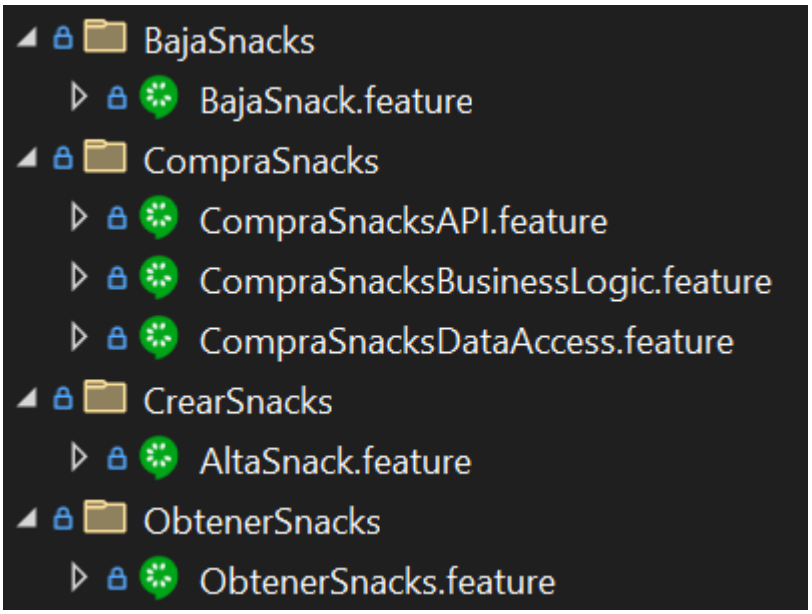
Implementación de los casos de prueba con Specflow

Features:

Se transformaron los escenarios definidos anteriormente en las siguientes features de Specflow:
Todas las features:



Baja de snack:

```
Feature: BajaSnack

A short summary of the feature

Scenario: Apretar el botón de Eliminar
Given me encuentro en la pestaña de los snacks
When apreto el boton de Eliminar
Then el snack se eliminó del listado
And las personas que compraron este snack pueden consumir las unidades que compraron
And los espectadores no pueden comprar de ahora en adelante unidades de este producto
```

Compra de Snack para la caja de API:

```
Feature: CompraSnacksAPI

Definicion de los casos de test para la feature "Compra de Snacks" que testeam la capa ""

@tag1
Scenario: Selecting a quantity of snacks less than or equal to 0
    Given I have selected a (non-empty) set of snacks
    And for any of the snacks I have selected a quantity of snacks less than or equal to 0
    When I press the Confirm snack purchase button
    Then an error message is displayed that says "No se puede comprar una cantidad de snacks menor o igual a 0"

@tag2
Scenario: Not selecting any snack
    Given that I didn't select any snacks
    When I press the Confirm snack purchase button
    Then an error message is displayed that says "Tiene que seleccionar al menos un snack"

@tag3
Scenario: Successful snack purchase
    Given I have selected a (non-empty) set of snacks
    And I have selected a quantity greater than 0 for each selected snack
    When I press the Confirm snack purchase button
    Then the price corresponding to the selected quantity of snacks is added to the total price of the tickets.
```

Compra de Snack para la caja de Business:

```
Feature: CompraSnacksBusinessLogic

A short summary of the feature

@tag1
Scenario: Selecting a quantity of snacks less than or equal to 0
    Given I have selected a (non-empty) set of snacks
    And for any of the snacks I have selected a quantity of snacks less than or equal to 0
    When I press the Confirm snack purchase button
    Then an error message is displayed that says "No se puede comprar una cantidad de snacks menor o igual a 0"

@tag2
Scenario: Not selecting any snack
    Given that I didn't select any snacks
    When I press the Confirm snack purchase button
    Then an error message is displayed that says "Tiene que seleccionar al menos un snack"

@tag3
Scenario: Successful snack purchase
    Given I have selected a (non-empty) set of snacks
    And I have selected a quantity greater than 0 for each selected snack
    When I press the Confirm snack purchase button
    Then the price corresponding to the selected quantity of snacks is added to the total price of the tickets.
    And the purchase is completed successfully
```

Compra de Snack para la capa de DataAccess:

```
Feature: CompraSnacksDataAccess

A short summary of the feature

@tag3
Scenario: Successful snack purchase
    Given I have selected a (non-empty) set of snacks
    And I have selected a quantity greater than 0 for each selected snack
    When I press the Confirm snack purchase button
    Then the purchase is completed successfully
```

Alta Snacks:

```
Feature: AltaSnack

Scenario: Crear un snack sin descripción
Given me encuentro en la pestaña de creación de los snacks
And escribo un precio
When apreto el boton de Confirmar
Then me sale un mensaje de error que dice "Tiene que ingresar una descripción para el snack"

Scenario: Crear un snack sin precio
Given me encuentro en la pestaña de creación de los snacks
And escribo una descripción
When apreto el boton de Confirmar
Then me sale un mensaje de error que dice "Tiene que ingresar un precio para el snack"

Scenario: Crear un snack sin precio ni descripción
Given me encuentro en la pestaña de creación de los snacks
When apreto el boton de Confirmar
Then me sale un mensaje de error que dice "Tiene que ingresar un precio y una descripción para el snack"

Scenario: Crear un snack con un precio negativo
Given me encuentro en la pestaña de creación de los snacks
And escribo una descripción
And escribo un precio negativo
When apreto el boton de Confirmar
Then me sale un mensaje de error que dice "Tiene que ingresar un precio mayor o igual a 0 para el snack"

Scenario: Crear un snack con un precio positivo
Given me encuentro en la pestaña de creación de los snacks
And escribo una descripción
And escribo un precio positivo
When apreto el boton de Confirmar
Then me sale un mensaje que dice "Snack creado con éxito"


Scenario:  Crear un snack con un descripción existente
Given me encuentro en la pestaña de creación de los snacks
And escribo una descripción que ya existe previamente
And escribo un precio positivo
When apreto el boton de Confirmar
Then me sale un mensaje de error que dice "Snack creado previamente"
```

Traerse todos los snacks:

```
Feature: Obtener snacks

A short summary of the feature

@tag1

Scenario: Seleccionar la opción de comprar snacks
    Given que realice los pasos de compra de tickets correctamente
    When apreto el boton de “Comprar tickets”
    Then se muestra un menú del cual puedo seleccionar distintos snacks y distintas cantidades
```

Todos las features(specflow) corresponden a una User Story, con los escenarios adaptados para que tengan contexto en el backend. La excepción es la feature ObtenerSnacks, el cual tiene un escenario que corresponde a la compra de snacks pero como requería un endpoint distinto decidimos asignarle una feature distinta así se podían desarrollar de manera independiente.

Implementación de los casos de prueba:

Como se puede observar por la distribución de las features, se crearon 3 features para la implementación de la user story “Comprar Snacks”. Esto ocurrió porque fue la primer feature que implementamos, y debido a un desconocimiento de cómo aplicar BDD en una estructura de capas, creímos necesario probar una capa a la vez por lo que se creo una feature por capa y se mockearon las dependencias a las capas inferiores. Esto es buena

practica en TDD, sin embargo es un test de caja blanca lo cual va en contra de como se desarrolla en BDD. Esto ocasionó además que la definición de los “steps” no tuviera casi ningún sentido ya que el comportamiento de pasaje de datos entre capas no refleja el caso de uso. Esto provocó que fuera muy complicado definir los steps, lo que enlentece enormemente el desarrollo. Para rematar, dado que se mockeron la mayoría de las dependencias externas, las pruebas terminaron teniendo poco valor.
Sin embargo, aprendimos de nuestros errores y para las siguientes features testeamos los resultados interactuando solamente con el controller, el cual se definió a través del uso de una factory (clase llamada FactorySnackService), el cual se encargaba de realizar la dependency injection. El unico mockeo que se realizó fue el de la base de datos por razones de performance. De esta manera, se podian definir los “steps” sin tener en cuenta la implementación del controller.

Implementacion del codigo de backend

Se creó un nuevo controller llamado “SnackController” para atender a las llamadas correspondientes a los nuevos endpoints. Reciben y mandan dtos los cuales además tienen la responsabilidad de mapearse - creando la entidad del dominio correspondiente si son dtos de entrada y creándose en base a una entidad del dominio si son dtos de salida.
Para la US “Alta Snack”, el dto tambien lanzo errores correspondientes a escenarios borde definidos en la US. Estos no se realizaron en la capa de Business ya que tenían que ver con omisión de datos en tipos que no admiten nulos en C#, por lo que para poder crear la entidad del dominio no se podía delegar este manejo de errores.
El controller delega el procesamiento a un service de la capa Business, en este caso SnackService.

La clase SnackService tiene la responsabilidad de realizar las validaciones correspondientes a la logica de negocio.
Para la US “Alta Snack”, se asegura de que el precio sea mayor a 0 y de que no haya un Snack creado con la misma descipción.
Para la US “Compra Snacks”, se asegura que la compra tenga el formato adecuado (las id de los snacks correspondan a un snack, que ninguna cantidad comprada sea menor o igual al 0) y define el parámetro “Precio Total”.
Una vez hechas las validaciones, se delega a la clase SnackManagement de la capa DataAccess la implementación de la acción correspondiente (Crear,Borrar,Traer) en la base de datos.

La clase SnackManagement interactúa con un contexto de EntityFramework para realizar las acciones necesarias para cumplir con los requisitos.
Se creó una tabla que relaciona las compras de snacks con los tickets, tal que a cada compra le corresponde un ticket y cada ticket le corresponde como máximo una compra. Se decidió no modificar la clase Ticket para no vernos forzados a actualizar los tests de TDD que venían con el proyecto, además de tener que refactorizar una cantidad considerable del código fuente.
Para el borrado de un snack se realizó un borrado lógico, ya que si no ocurrirán problemas al traernos las compras de snacks de la base de datos.

Cabe aclarar que no hay dependencia directa entre los componentes previamente mencionados, sino que realizan una inversión de dependencias y cada uno depende de una interfaz.