Natalia Coronado Romero 2ºDAM

ÍNDICE

Introducción

Proyecto de una aplicación con MAUI en Visual Studio siguiendo el patrón MVVM, permitiendo gestionar una lista de tareas con navegación entre pantallas.

Objetivo de la memoria

El objetivo de esta práctica es Implementar la arquitectura MVVM para separar la lógica de la interfaz, utilizando INotifyPropertyChanged y ICommand.

Material utilizado

Marca y Modelo del Procesador: Intel Pentium CPU G4400 @ 3.30GHz
Tipo Memoria RAM (memoria y slots): 4GB RAM, 2400 MHz **Tipo de dispositivo de almacenamiento capacitad (GiB):** HDD 1TB

Programas utilizados: Spectacle, Visual Studio 2022, OBS Studio

GitHub

Es es el enlace de github al repositorio de la asignatura:
https://github.com/MalvaLego/Desarrollo-de-Interfaces.git

Desarrollo

Interfaz de la Aplicación

-MainPage.xaml y AddItemNewWindow.xaml

Esta es la vista previa al ejecutar la aplicación. En estas páginas se muestran la lista de tareas que se han añadido y en la otra da opción a crear una nueva ya realizada o no.

Figura 1: Vista previa del MainPage de la aplicación

Figura 2: Vista previa del AddItemNewWindow de la aplicación

Código de la Interfaz Visual

-MainPage.xaml

Este es el código que hará se muestre en pantalla lo que queramos. Contiene un Label, un botón para cambiar a la siguiente página y un CollectionView. Se añade el nombre de la clase que tiene y la conexión de viewmodels con la carpeta ViewModel.

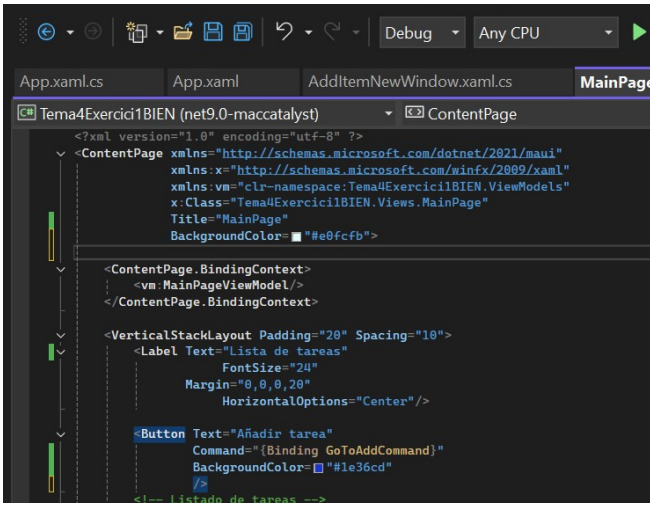

Figura 3: Primera parte del código visual del MainPage: un label y un botón

En el CollectionView mostrará el contenido que haya en Lista, poniendo en este un checkbox, un Label con el nombre y un botón para borrar.

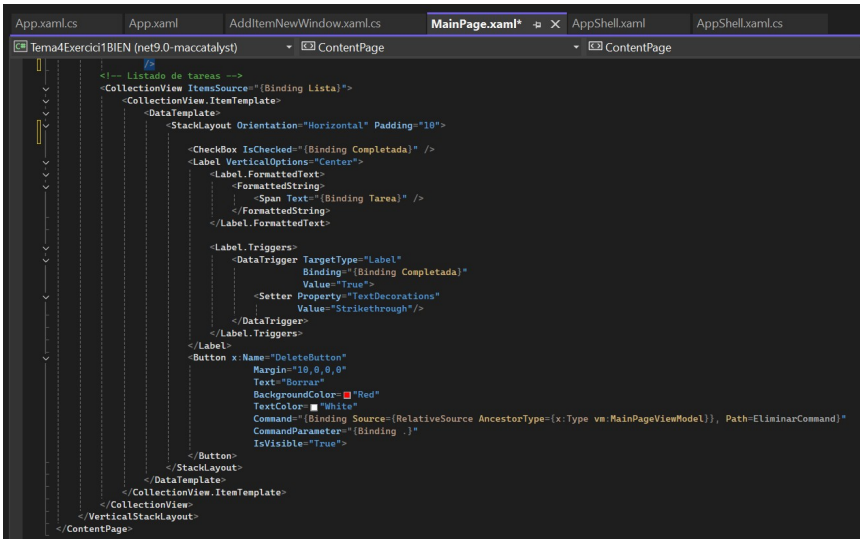

Figura 4: Segunda parte del código visual del MainPage: CollectionView de una lista con un checkbox, un label y un botón

-AddItemNewWindow.xaml

Este es el código que mostrará la segunda página de la aplicación. Contiene un Label, un Entry para escribir el nombre de la tarea, un checkbox para indicar si está terminada o no y dos botones, uno para añadir la tarea y otro para volver atrás sin añadir nada. Se pone el nombre de la clase que tiene y la conexión de viewmodels con la carpeta ViewModel al principio del código.

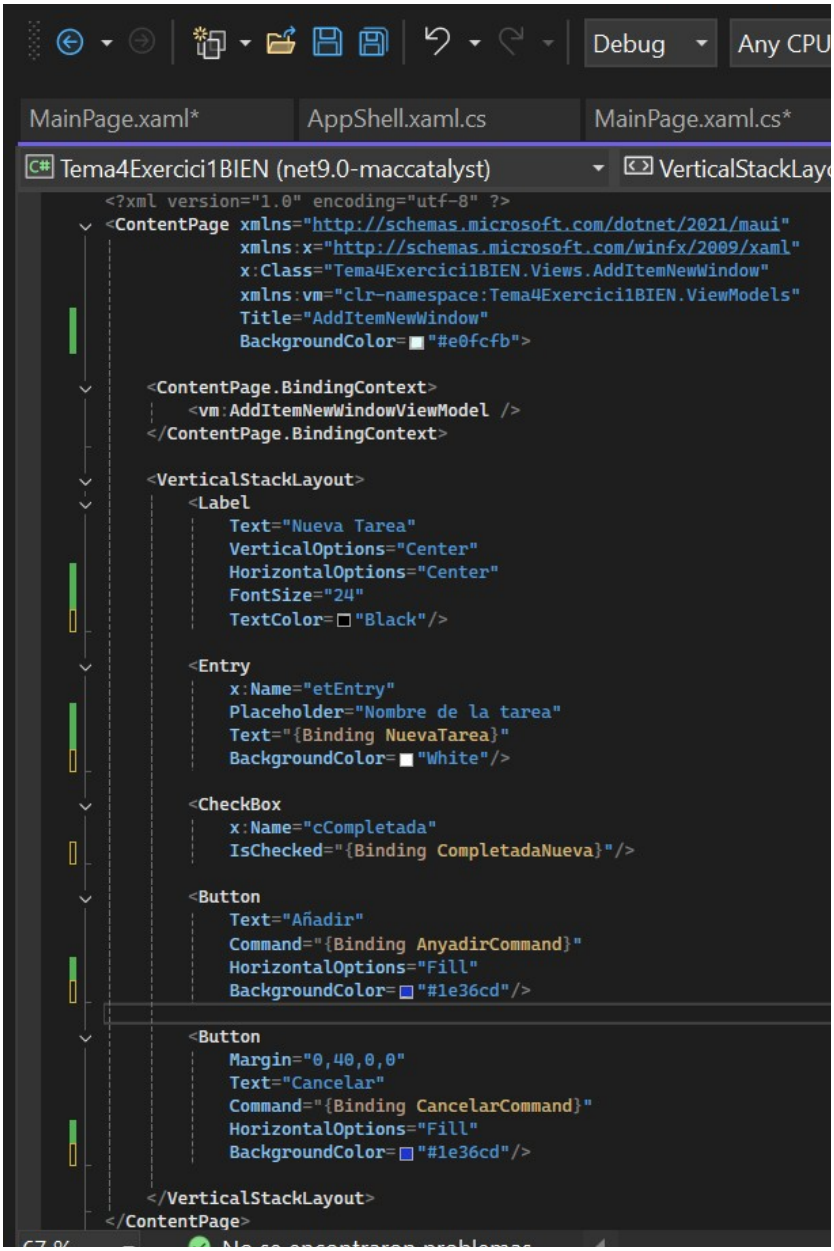

Figura 5: Código visual de AddItemNewWindow: un label, un Entry, un checkbox y dos botones

Código AppShell

En el código del AppShell.cs se ha añadido la otra página llamada AddNewItemView a parte que la de MainPage para así poder navegar entre las páginas y pasar los datos.

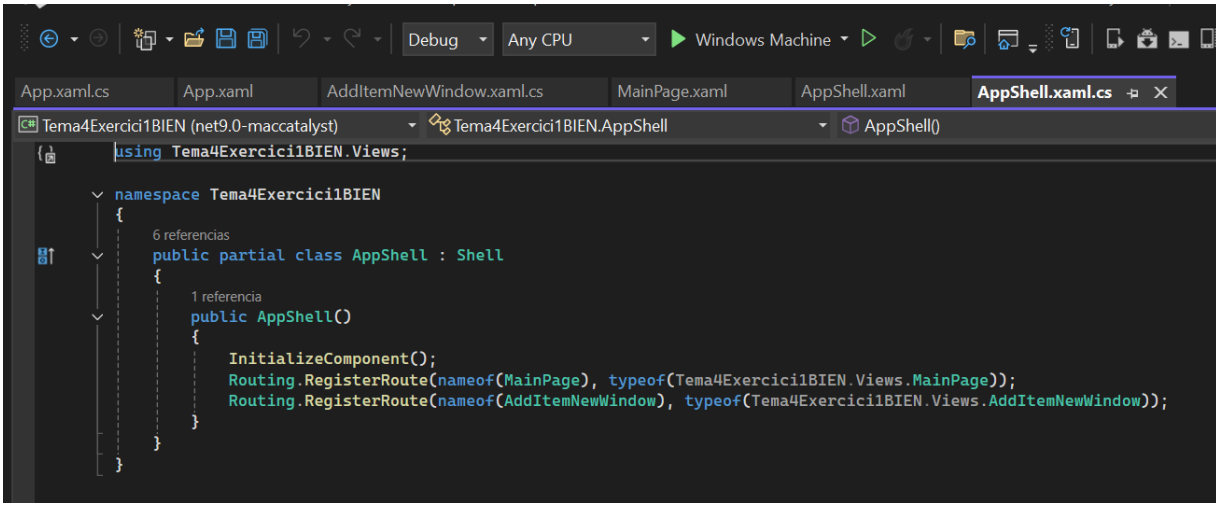

Figura 6: Código Appshel para el Routing de las páginas

Código Interno

-TodoITem.cs

En la clase TodoItem guarda una tarea teniendo como datos el nombre y un boolean para indicar si está completada o no. El constructor inicializa la tarea con los parámetros que recibe.

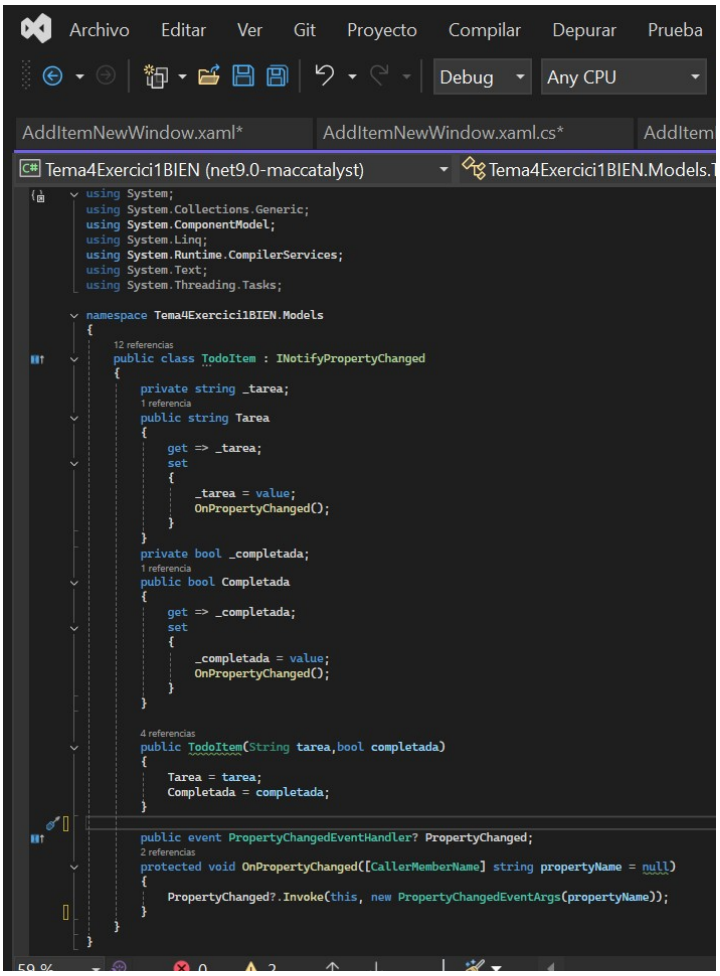

Figura 7: Código de TodoItem: inicialización de tarea con el nombre y el booleano

-MainPage.cs

El ObservableCollectionView Lista almacena objetos tipo TodoItem, para realizar esto recibe mediante el QueryProperty una Tarea desde la otra página y se inicializa en una variable Tarea, al recibir esta tarea hará validaciones de si está vacío o si el check es true o false. Una vez comprobado esto creará una nueva tarea de tipo TodoItem y se añadirá en la lista.
En este código se utiliza los ICommand para llamar a las funciones. En este caso al pulsar el de GoToAdd navegará a la página AddItemNewWindow, y la función de Eliminar detectará si pulsa el botón de borrar la tarea y la eliminará de la lista.
Todo esto utilizando INotifyPropertyChanged para que se actualice automáticamente.

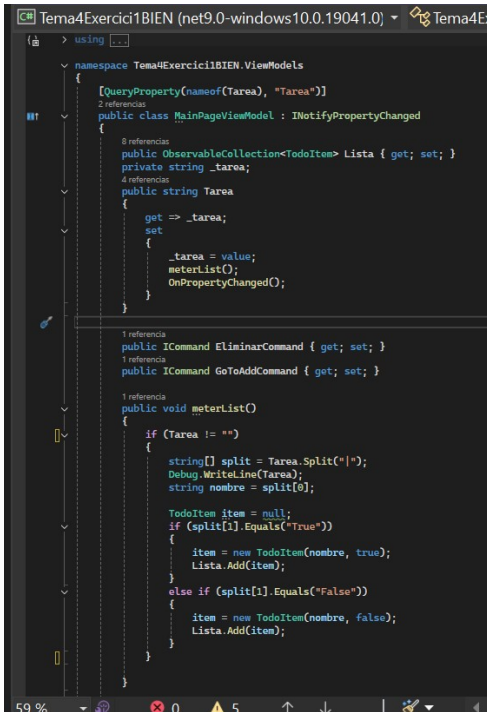

Figura 8: Primera parte del código de MainPageViewModel: ObservableCollection, QueryProperty, ICommand

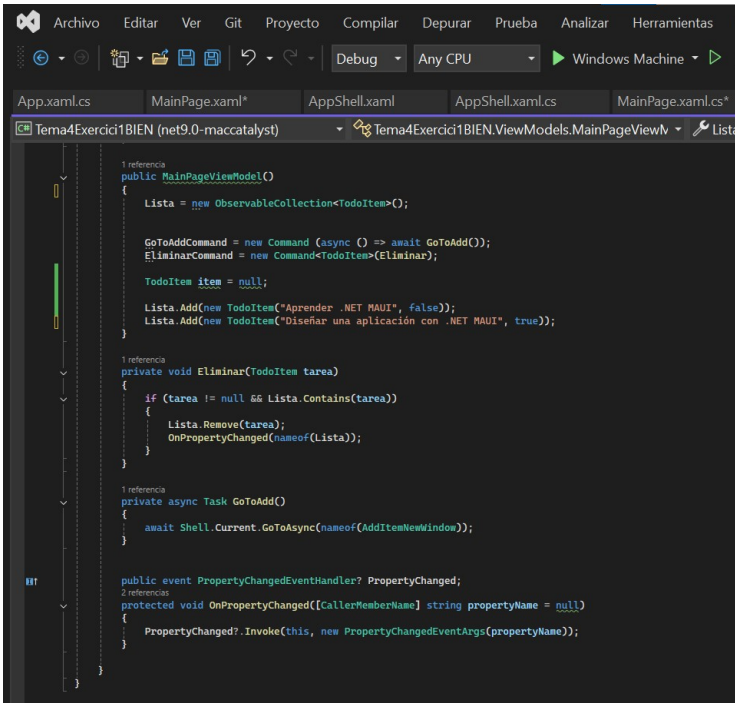

Figura 9: Segunda parte del código de MainPageViewModel: Constructor, dos funciones para navegar de página y eliminar y el INotifyPropertyChanged

-AddItemNewWindow.cs

En este código se guardan los datos de una nueva tarea escrita en el Entry y en el check para enviarlo al MainPage y se guarde en la lista.
En este código se utiliza los ICommand para llamar a las funciones. En este caso al pulsar el de Cancelar volverá a la página MainPage con la tarea vacía ya que no se ha añadido nada, en cambio la función de Anyadir enviará en un string la información de la nueva tarea que se ha puesto en la página.
Todo esto utilizando INotifyPropertyChanged para que se actualice automáticamente.

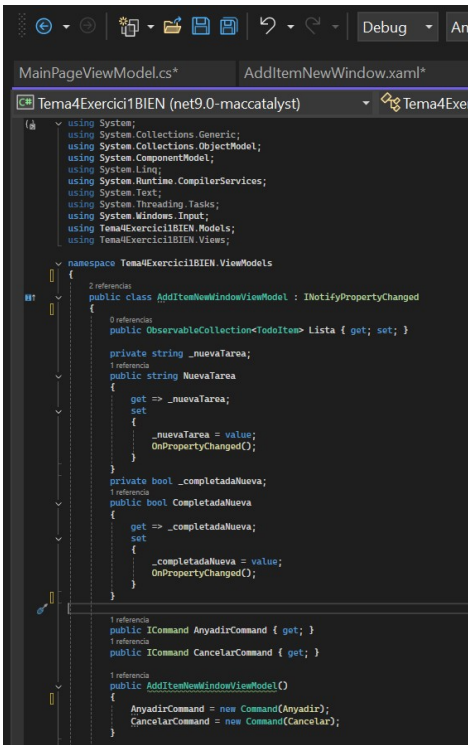

Figura 11: Primera parte del código de AddNewItemViewModel: Creación de nuevos datos de la tarea, ICommand

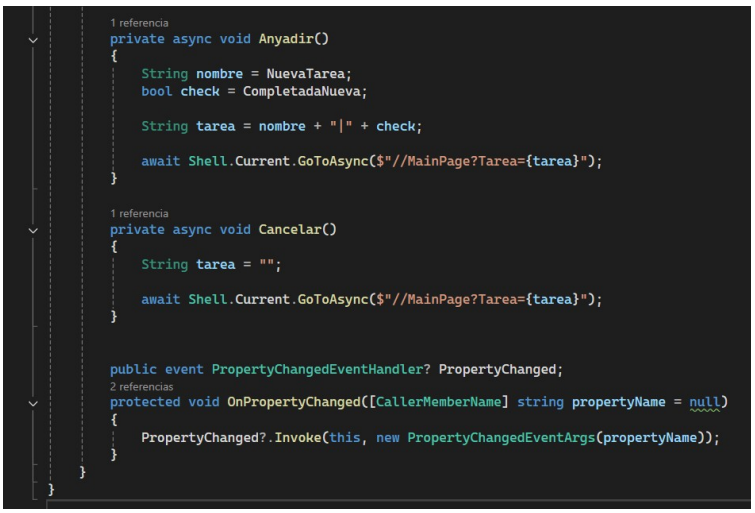

Figura 10: Segunda parte del código de AddNewItemViewModel: Dos funciones para navegar de página, una para no enviar nada y la otra para envíar los nuevos datos, INotifyPropertyChanged

Problemas encontrados y sugerencias

Ha habido distintos errores al igual que siempre, después de ir probando, hacer otras veces el proyecto y ayudas externas para conseguir información se ha conseguido. Se ha echado en falta algo más de información.

Conclusión

Buena actividad para aprender a navegar entre vistas de forma eficiente, mejorando la organización del código, utilizando ViewModels.

Bibliografía/webgrafía

·Pdfs del profesor de mi profesor.

Video del funcionamiento

Utilizar enlace de la leyenda abajo:

Dibujo 1: https://youtu.be/MPOl0E_CTwE