

EXERCICI LISTA TAREAS



MARIO FERNÁNDEZ
DIN

2ºDAM

EXERCICI LISTA TAREAS

	1
1. Introducción	3
2. Objetivos	3
3. Implementaciones	3
4. Planificación	3
6. Retos y Soluciones	8
7. Conclusión	8
8. Fuentes	8

Mario

1. Introducción

En este documento, te cuento cómo desarrollé una aplicación en .NET MAUI que gestiona una lista de tareas. La app tiene dos páginas principales que te permiten ver y gestionar las tareas, y añadir nuevas tareas. El objetivo es que sea fácil y funcional para los usuarios, asegurando que la información esté siempre clara y al alcance de un clic.

2. Objetivos

- Crear una interfaz con dos páginas que permitan gestionar una lista de tareas. Los usuarios podrán:
- Ver una lista de tareas.
- Marcar las tareas como completadas.
- Eliminar tareas existentes.
- Añadir nuevas tareas.

3. Implementaciones

- Repasé conceptos clave como la navegación entre páginas en .NET MAUI, el uso de bindings en XAML para actualizar datos dinámicamente, la habilitación de botones basada en condiciones y la implementación de la lógica de negocio en los ViewModels.

4. Planificación



5. Resultados Detallados

MainPage.xaml:

Muestra una lista de tareas utilizando un `CollectionView`.

Cada tarea tiene una casilla de verificación para marcarla como completada y un botón para eliminarla.

Incluye un botón para navegar a la página de añadir tarea.

```
12 <VerticalStackLayout Padding="20" Spacing="10">
13   <Label Text="Lista de tareas" FontSize="24" HorizontalOptions="Center" />
14   <Button Text="Añadir tarea" Command="{Binding AddItemCommandNewWindow}" />
15
16   <CollectionView ItemsSource="{Binding Items}">
17     <CollectionView.ItemTemplate>
18       <DataTemplate>
19         <StackLayout Orientation="Horizontal" Padding="10">
20           <CheckBox IsChecked="{Binding IsCompleted}" />
21           <Label VerticalOptions="Center">
22             <Label.FormattedText>
23               <FormattedString>
24                 <Span Text="{Binding Title}" />
25               </FormattedString>
26             </Label.FormattedText>
27             <Label.Triggers>
28               <DataTrigger TargetType="Label" Binding="{Binding IsCompleted}" Value="True">
29                 <Setter Property="TextDecorations" Value="Strikethrough" />
30               </DataTrigger>
31             </Label.Triggers>
32           </Label>
33           <Button Text="Borrar"
34                 TextColor="White"
35                 BackgroundColor="Red"
36                 Command="{Binding Source={RelativeSource AncestorType={x:Type vm:MainPageViewModel}}, Path=DeleteItemCommand}"
37                 CommandParameter="{Binding .}"
38                 Margin="10,0,0,0" />
39         </StackLayout>
40       </DataTemplate>
41     </CollectionView.ItemTemplate>
42   </CollectionView>
43 </VerticalStackLayout>
44 </ContentPage>
```

```
1 namespace ListaMario.Views
2 {
3     5 referencias
4     public partial class MainPage : ContentPage
5     {
6         0 referencias
7         public MainPage()
8         {
9             InitializeComponent();
10        }
11    }
12 }
```

AddItemNewWindow.xaml y cs:

Proporciona un campo de entrada para el título de la nueva tarea y una casilla de verificación para su estado de completado.

Incluye botones para añadir la tarea y para cancelar la operación.

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
3             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4             xmlns:vm="clr-namespace:ListaMario.Views.Models"
5             x:Class="ListaMario.Views.AddItemNewWindow"
6             Title="Nueva tarea">
7
8     <ContentPage.BindingContext>
9         <vm:AddItemNewWindowViewModel />
10    </ContentPage.BindingContext>
11
12    <VerticalStackLayout Padding="20" Spacing="10">
13        <Label Text="Nueva tarea" FontSize="24" HorizontalOptions="Center" />
14        <Entry Placeholder="Nombre de la tarea" Text="{Binding NewTaskTitle}" />
15        <CheckBox IsChecked="{Binding IsCompleted}" />
16        <HorizontalStackLayout Spacing="10">
17            <Button Text="Añadir" Command="{Binding AddItemCommand}" />
18            <Button Text="Cancelar" Command="{Binding CancelCommand}" />
19        </HorizontalStackLayout>
20    </VerticalStackLayout>
21 </ContentPage>
```

```
1 namespace ListaMario.Views
2 {
3     5 referencias
4     public partial class AddItemNewWindow : ContentPage
5     {
6         0 referencias
7         public AddItemNewWindow()
8         {
9             InitializeComponent();
10    }
11 }
```

AppShell.xaml:

Define las rutas para las páginas principales (MainPage y AddItemNewWindow).

Configura la navegación utilizando Shell.Navigation.GotoAsync para pasar parámetros entre páginas.

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <Shell xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
3       xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4       xmlns:views="clr-namespace:ListaMario.Views"
5       x:Class="ListaMario.AppShell">
6
7     <ShellContent Title="Lista de tareas" ContentTemplate="{DataTemplate views:MainPage}" Route="MainPage"/>
8     <ShellContent Title="Nueva tarea" ContentTemplate="{DataTemplate views:AddItemNewWindow}" Route="AddItemNewWindow"/>
9 </Shell>
```

TodoItem.cs:

La clase TodoItem es un modelo de datos para representar tareas en la lista de tareas de la aplicación. Implementa la interfaz INotifyPropertyChanged para que cualquier cambio en sus propiedades (Title e IsCompleted) se notifique automáticamente a la interfaz de usuario. Esto permite que la interfaz de usuario se actualice dinámicamente cuando se marcan tareas como completadas o se cambian los títulos de las tareas.

```
C# ListaMario (net8.0-android) ListaMario.Models.TodoItem

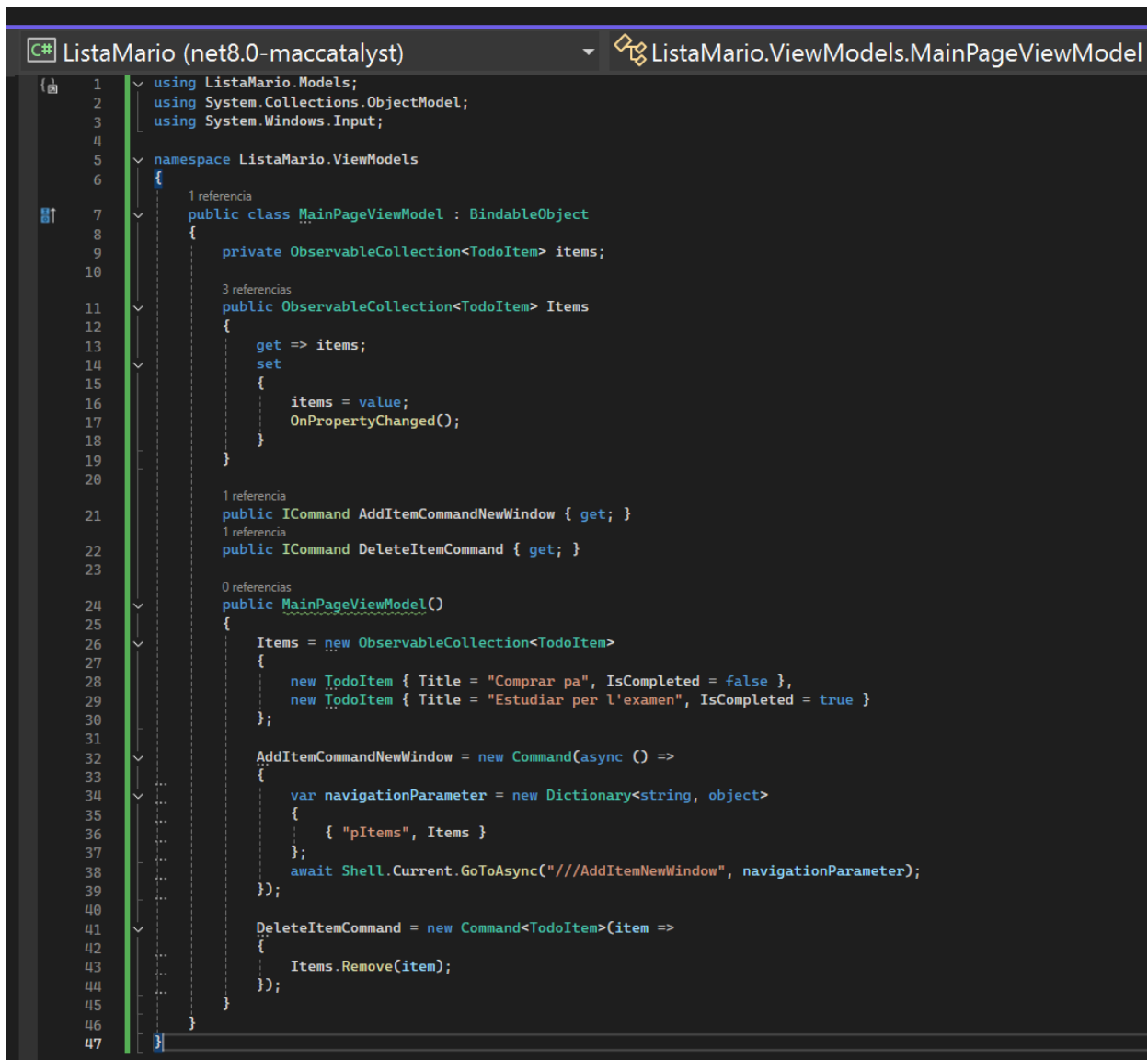
1 using System.ComponentModel;
2
3 namespace ListaMario.Models
4 {
5     9 referencias
6     public class TodoItem : INotifyPropertyChanged
7     {
8         private string title;
9         private bool isCompleted;
10
11         4 referencias
12         public string Title
13         {
14             get => title;
15             set
16             {
17                 if (title != value)
18                 {
19                     title = value;
20                     OnPropertyChanged(nameof(Title));
21                 }
22             }
23
24         4 referencias
25         public bool IsCompleted
26         {
27             get => isCompleted;
28             set
29             {
30                 if (isCompleted != value)
31                 {
32                     isCompleted = value;
33                     OnPropertyChanged(nameof(IsCompleted));
34                 }
35             }
36
37         public event PropertyChangedEventHandler PropertyChanged;
38
39         2 referencias
40         protected void OnPropertyChanged(string propertyName)
41         {
42             PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
43         }
44     }
45 }
```

MainPageViewModel.cs:

Define una colección observable de TodoItem que se vincula a la vista principal.

Contiene comandos para añadir una nueva tarea y eliminar una tarea existente.

Implementa la lógica para navegar a la página de añadir tarea, pasando la lista de tareas como parámetro.



```
1 using ListaMario.Models;
2 using System.Collections.ObjectModel;
3 using System.Windows.Input;
4
5 namespace ListaMario.ViewModels
6 {
7     public class MainPageViewModel : BindableObject
8     {
9         private ObservableCollection<TodoItem> items;
10
11         public ObservableCollection<TodoItem> Items
12         {
13             get => items;
14             set
15             {
16                 items = value;
17                 OnPropertyChanged();
18             }
19         }
20
21         public ICommand AddItemCommandNewWindow { get; }
22         public ICommand DeleteItemCommand { get; }
23
24         public MainPageViewModel()
25         {
26             Items = new ObservableCollection<TodoItem>
27             {
28                 new TodoItem { Title = "Comprar pa", IsCompleted = false },
29                 new TodoItem { Title = "Estudiar per l'examen", IsCompleted = true }
30             };
31
32             AddItemCommandNewWindow = new Command(async () =>
33             {
34                 var navigationParameter = new Dictionary<string, object>
35                 {
36                     { "pItems", Items }
37                 };
38                 await Shell.Current.GoToAsync("///AddItemNewWindow", navigationParameter);
39             });
40
41             DeleteItemCommand = new Command<TodoItem>(item =>
42             {
43                 Items.Remove(item);
44             });
45         }
46     }
47 }
```

AddItemNewWindowViewModel.cs:

Define propiedades para el título de la nueva tarea y su estado de completado. Contiene comandos para añadir la nueva tarea a la lista y cancelar la operación. Implementa la lógica para navegar de regreso a la página principal.

```
C# ListaMario (net8.0-maccatalyst) ListaMario.ViewModels.AddItemN
4
5 namespace ListaMario.ViewModels
6 {
7     [QueryProperty(nameof(Items), "pItems")]
8     public class AddItemNewWindowViewModel : BindableObject
9     {
10         private ObservableCollection<TodoItem> items;
11         private string newTaskTitle;
12         private bool isCompleted;
13
14         2 referencias
15         public ObservableCollection<TodoItem> Items
16         {
17             get => items;
18             set
19             {
20                 items = value;
21                 OnPropertyChanged();
22             }
23         }
24
25         2 referencias
26         public string NewTaskTitle
27         {
28             get => newTaskTitle;
29             set
30             {
31                 newTaskTitle = value;
32                 OnPropertyChanged();
33             }
34         }
35
36         1 referencia
37         public bool IsCompleted
38         {
39             get => isCompleted;
40             set
41             {
42                 isCompleted = value;
43                 OnPropertyChanged();
44             }
45         }
46
47         1 referencia
48         public ICommand AddItemCommand { get; }
49
50         1 referencia
51         public ICommand CancelCommand { get; }
52
53         0 referencias
54         public AddItemNewWindowViewModel()
55         {
56             AddItemCommand = new Command(async () =>
57             {
58                 if (!string.IsNullOrEmpty(NewTaskTitle))
59                 {
60                     var newItem = new TodoItem { Title = NewTaskTitle, IsCompleted = IsCompleted };
61                     Items.Add(newItem);
62                     await Shell.Current.GoToAsync("///MainPage");
63                 }
64                 else
65                 {
66                     // Opcional: Mostrar un missatge d'error si el títol de la tasca està buit
67                     await Application.Current.MainPage.DisplayAlert("Error", "El títol de la tasca no pot estar buit.", "OK");
68                 }
69             });
70
71             CancelCommand = new Command(async () =>
72             {
73                 await Shell.Current.GoToAsync("///MainPage");
74             });
75         }
76     }
77 }
```


6. Retos y Soluciones

1. Fallaba el paso de parámetros entre páginas:
 - Solución: Me aseguré de que los nombres en QueryProperty coincidieran con los parámetros en la URL.
2. No se actualizaban los valores vinculados en la interfaz:
 - Solución: Añadí OnPropertyChanged() a las propiedades del código detrás.
3. El cálculo del precio no funcionaba correctamente:
 - Solución: Verifiqué las condiciones de la lógica en el método CalcularPrecio.
4. El botón de navegación no redirigía a la página correcta:
 - Solución: Revisé y corregí la estructura de las rutas en GoToAsync.
5. El diseño no se adaptaba bien a diferentes pantallas:
 - Solución: Añadí un ScrollView para manejar contenido dinámico.

7. Conclusión

Durante el proyecto, me topé con varios problemas como pasar datos entre páginas, que los cálculos no salieran bien o que el diseño no se viera bien en todas las pantallas. Pero ajustando rutas, arreglando la lógica y usando cosas como ScrollView, conseguí que todo funcionara como quería.

8. Fuentes

- Documentación de .NET MAUI: <https://docs.microsoft.com/en-us/dotnet/maui/>
- Guía de XAML: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/>
- Métodos asincrónicos en C#: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/>
- IA ChatGPT para informarme de algunos aspectos que no sabía que hacían.