MAUI CRUD avec MVVM Toolkit

Ce document présente la mise en œuvre d'une application CRUD (Create, Read, Update, Delete) simple pour gérer des cartes (avec titre et contenu textuel) en utilisant .NET MAUI et le Community Toolkit MVVM. Les données sont stockées uniquement en mémoire RAM.

Sommaire

0. Structure du projet
1. Structure du Modèle
2. Service de Gestion des Cartes
3. ViewModel
4. Interface Utilisateur
5. Configuration de l'Application

0. Structure du projet

```
CardsApp/
│
├── Models/
│   └── Card.cs                 # Modèle de données pour une carte
│
├── Services/
│   └── CardService.cs          # Service de gestion des cartes en mémoire (CRUD)
│
├── ViewModels/
│   └── CardsViewModel.cs       # ViewModel avec les commandes CRUD
│
├── Views/
│   └── CardsPage.xaml          # Interface utilisateur avec formulaire et liste
│
├── Converters/
│   └── InvertedBoolConverter.cs # Pour gérer l'affichage conditionnel des boutons
│
└── MauiProgram.cs              # Enregistrement des services
```

1. Structure du Modèle

Notre modèle `Card` est simple, représentant une carte avec un titre et un contenu textuel.

```
using System;

namespace CardsApp.Models
```

```
{
    public class Card
    {
        public string Id { get; set; } = Guid.NewGuid().ToString();
        public string Title { get; set; }
        public string Content { get; set; }
        public DateTime CreatedAt { get; set; } = DateTime.Now;
    }
}
```

> Avec la base de données, l'ID pourra facilement passer en int (et auto incrémenté) L'utilisation de string avec Guid évite de devoir gérer les ids à la main...

Chaque carte possède :

- Un identifiant unique généré automatiquement
- Un titre
- Un contenu textuel
- Une date de création

2. Service de Gestion des Cartes

Nous créons un service qui gère les opérations CRUD sur les cartes, stockées uniquement en mémoire RAM.

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using CardsApp.Models;

namespace CardsApp.Services
{
    public static CardService instance = new CardService();
    public class CardService
    {
        // Collection en mémoire RAM pour stocker les cartes
        private List<Card> _cards = new List<Card>();

        // Récupère toutes les cartes
        public List<Card> GetCards()
        {
            return _cards.ToList();
        }

        // Récupère une carte par son ID
        public Card GetCard(string id)
        {
            return _cards.FirstOrDefault(c => c.Id == id);
        }

        // Ajoute une nouvelle carte
        public void AddCard(Card card)
        {
            _cards.Add(card);
```

```
        }

        // Met à jour une carte existante
        public bool UpdateCard(Card card)
        {
            var existingCard = _cards.FirstOrDefault(c => c.Id == card.Id);
            if (existingCard == null)
                return false;

            var index = _cards.IndexOf(existingCard);
            _cards[index] = card;
            return true;
        }

        // Supprime une carte
        public bool DeleteCard(string id)
        {
            var card = _cards.FirstOrDefault(c => c.Id == id);
            if (card == null)
                return false;

            return _cards.Remove(card);
        }
    }
}
```

> **Note** : Ce service utilise une simple liste en mémoire, sans persistance. Dans une application réelle, vous pourriez remplacer ce service par une implémentation utilisant une base de données locale.

3. ViewModel

Le ViewModel utilise le Community Toolkit MVVM pour simplifier l'implémentation du pattern MVVM. Nous utilisons les attributs de source génération comme `[ObservableProperty]` et `[RelayCommand]` pour réduire le code boilerplate.

```
using System.Collections.ObjectModel;
using CardsApp.Models;
using CardsApp.Services;
using CommunityToolkit.Mvvm.ComponentModel;
using CommunityToolkit.Mvvm.Input;

namespace CardsApp.ViewModels
{
    public partial class CardsViewModel : ObservableObject
    {
        private readonly CardService _cardService = CardService.Instance;

        [ObservableProperty]
        private ObservableCollection<Card> _cards;

        [ObservableProperty]
        private Card _selectedCard;

        [ObservableProperty]
        private string _cardTitle;
```

```
[ObservableProperty]
private string _cardContent;

[ObservableProperty]
private bool _isEditing;

public CardsViewModel()
{
    Cards = new ObservableCollection<Card>();
    LoadCards();
}

// Charge les cartes depuis le service
private void LoadCards()
{
    var cards = _cardService.GetCards();

    Cards.Clear();
    foreach (var card in cards)
    {
        Cards.Add(card);
    }
}

// Ajoute une nouvelle carte
[RelayCommand]
public void AddCard()
{
    if (string.IsNullOrWhiteSpace(CardTitle))
        return;

    var newCard = new Card
    {
        Title = CardTitle,
        Content = CardContent
    };

    _cardService.AddCard(newCard);
    Cards.Add(newCard);

    // Réinitialiser les champs
    CardTitle = string.Empty;
    CardContent = string.Empty;
}

// Prépare l'édition d'une carte
[RelayCommand]
public void PrepareEdit(Card card)
{
    if (card == null)
        return;

    SelectedCard = card;
    CardTitle = card.Title;
    CardContent = card.Content;
    IsEditing = true;
}

// Met à jour une carte existante
[RelayCommand]
```

```
        public void UpdateCard()
        {
            if (SelectedCard == null || string.IsNullOrWhiteSpace(CardTitle))
                return;

            SelectedCard.Title = CardTitle;
            SelectedCard.Content = CardContent;

            _cardService.UpdateCard(SelectedCard);

            // Rafraîchir la liste (ou juste mettre à jour l'élément spécifique)
            LoadCards();

            // Réinitialiser l'état d'édition
            CancelEdit();
        }

        // Supprime une carte
        [RelayCommand]
        public void DeleteCard(Card card)
        {
            if (card == null)
                return;

            _cardService.DeleteCard(card.Id);
            Cards.Remove(card);
        }

        // Annule l'édition en cours
        [RelayCommand]
        public void CancelEdit()
        {
            IsEditing = false;
            CardTitle = string.Empty;
            CardContent = string.Empty;
            SelectedCard = null;
        }
    }
}
```

4. Interface Utilisateur

L'interface utilisateur est définie en XAML. Nous utilisons une référence nommée `vm` pour le binding.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:viewmodels="clr-namespace:CardsApp.ViewModels"
             xmlns:models="clr-namespace:CardsApp.Models"
             x:Class="CardsApp.Views.CardsPage"
             Title="Gestion des Cartes">

    <!-- Déclaration du ViewModel en tant que ressource nommée "vm" pour le binding par référence -->
    <ContentPage.BindingContext>
        <viewmodels:CardsViewModel x:Name="vm"/>
    </ContentPage.BindingContext>
```

```
<Grid RowDefinitions="Auto,*,Auto" Padding="15" RowSpacing="10">
    <!-- Formulaire d'ajout/édition de carte -->
    <VerticalStackLayout Grid.Row="0" Spacing="10">
        <Entry Placeholder="Titre de la carte"
               Text="{Binding Source={x:Reference vm}, Path=CardTitle}"/>
        <Editor Placeholder="Contenu de la carte"
                AutoSize="TextChanges"
                Text="{Binding Source={x:Reference vm}, Path=CardContent}"
                HeightRequest="100"/>

        <Grid ColumnDefinitions="*,*" ColumnSpacing="10">
            <!-- Boutons conditionnels selon le mode (ajout ou édition) -->
            <Button Grid.Column="0"
                    Text="Ajouter"
                    Command="{Binding Source={x:Reference vm}, Path=AddCardCommand}"
                    IsVisible="{Binding Source={x:Reference vm}, Path=IsEditing, Converter={StaticResource

            <Button Grid.Column="0"
                    Text="Mettre à jour"
                    Command="{Binding Source={x:Reference vm}, Path=UpdateCardCommand}"
                    IsVisible="{Binding Source={x:Reference vm}, Path=IsEditing}"/>

            <Button Grid.Column="1"
                    Text="Annuler"
                    Command="{Binding Source={x:Reference vm}, Path=CancelEditCommand}"
                    IsVisible="{Binding Source={x:Reference vm}, Path=IsEditing}"/>
        </Grid>
    </VerticalStackLayout>

    <!-- Liste des cartes -->
    <CollectionView Grid.Row="1"
                    ItemsSource="{Binding Source={x:Reference vm}, Path=Cards}">
        <CollectionView.EmptyView>
            <VerticalStackLayout HorizontalOptions="Center" VerticalOptions="Center">
                <Label Text="Aucune carte disponible"/>
                <Label Text="Ajoutez une carte en utilisant le formulaire ci-dessus"/>
            </VerticalStackLayout>
        </CollectionView.EmptyView>

        <CollectionView.ItemTemplate>
            <DataTemplate x:DataType="models:Card">
                <SwipeView>
                    <SwipeView.RightItems>
                        <SwipeItems>
                            <SwipeItem Text="Éditer"
                                       Command="{Binding Source={x:Reference vm}, Path=PrepareEditCommand}"
                                       CommandParameter="{Binding .}"
                                       BackgroundColor="Orange"/>
                            <SwipeItem Text="Supprimer"
                                       Command="{Binding Source={x:Reference vm}, Path=DeleteCardCommand}"
                                       CommandParameter="{Binding .}"
                                       BackgroundColor="Red"/>
                        </SwipeItems>
                    </SwipeView.RightItems>

                    <Frame Margin="0,5" Padding="10">
                        <VerticalStackLayout>
                            <Label Text="{Binding Title}" FontSize="Medium" FontAttributes="Bold"/>
                            <Label Text="{Binding Content}" Margin="0,5,0,0"/>
                            <Label Text="{Binding CreatedAt, StringFormat='Créé le {0:dd/MM/yyyy à HH:mm}'}
```

```
                                    FontSize="Small"
                                    TextColor="Gray"
                                    Margin="0,5,0,0"/>
                            </VerticalStackLayout>
                        </Frame>
                    </SwipeView>
                </DataTemplate>
            </CollectionView.ItemTemplate>
        </CollectionView>
    </Grid>
</ContentPage>
```

Dans cet exemple, nous utilisons un `SwipeView` pour permettre à l'utilisateur de faire glisser les cartes vers la gauche pour révéler des options d'édition et de suppression.

> **Note** : Il faut également définir un convertisseur pour inverser les valeurs booléennes :

```
using System.Globalization;
using Microsoft.Maui.Controls;

namespace CardsApp.Converters
{
    public class InvertedBoolConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            return value is bool boolValue ? !boolValue : value;
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            return value is bool boolValue ? !boolValue : value;
        }
    }
}
```

5. Configuration de l'Application

Dans le fichier `MauiProgram.cs`, on enregistre le convertisseur.

```
using Microsoft.Extensions.Logging;
using CardsApp.Services;
using CardsApp.ViewModels;
using CardsApp.Views;

namespace CardsApp;

public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder();
```

```
        builder
            .UseMauiApp<App>()
            .ConfigureFonts(fonts =>
            {
                fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
                fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
            });

        // Enregistrer les convertisseurs comme ressources globales
        builder.Services.AddSingleton<ResourceDictionary>(provider => new ResourceDictionary
        {
            { "InvertedBoolConverter", new Converters.InvertedBoolConverter() }
        });

#if DEBUG
        builder.Logging.AddDebug();
#endif

        return builder.Build();
    }
}
```

Conclusion

Cet exemple montre comment implémenter une application CRUD simple pour gérer des cartes en utilisant .NET MAUI et le Community Toolkit MVVM. Les points clés de cette implémentation sont :

1. **Stockage en mémoire RAM** : Les données sont stockées uniquement en mémoire, sans persistance.
2. **Pattern MVVM simplifié** : Utilisation des générateurs de source du Community Toolkit MVVM pour réduire le code boilerplate.
3. **Binding par référence** : La référence au ViewModel est établie via `Binding Source={x:Reference vm}` .
4. **Interface intuitive** : Utilisation de SwipeView pour des interactions naturelles sur mobile.

Cette implémentation peut être facilement étendue pour ajouter la persistance des données à l'aide de SQLite ou d'autres mécanismes de stockage disponibles dans MAUI...