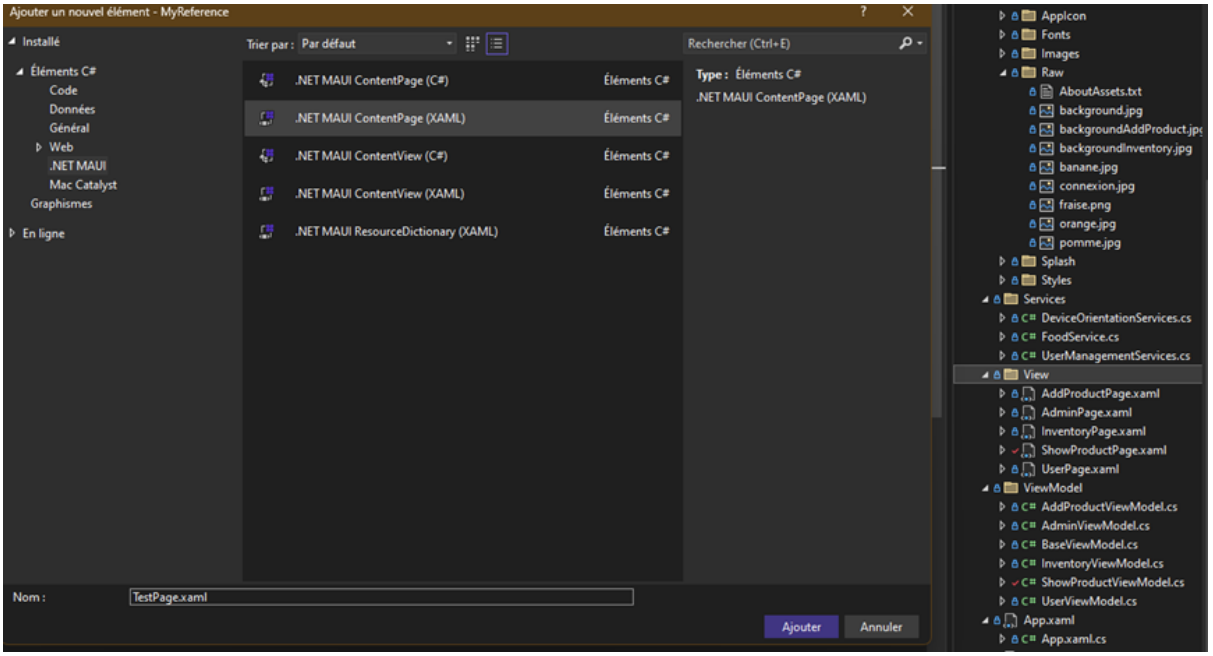
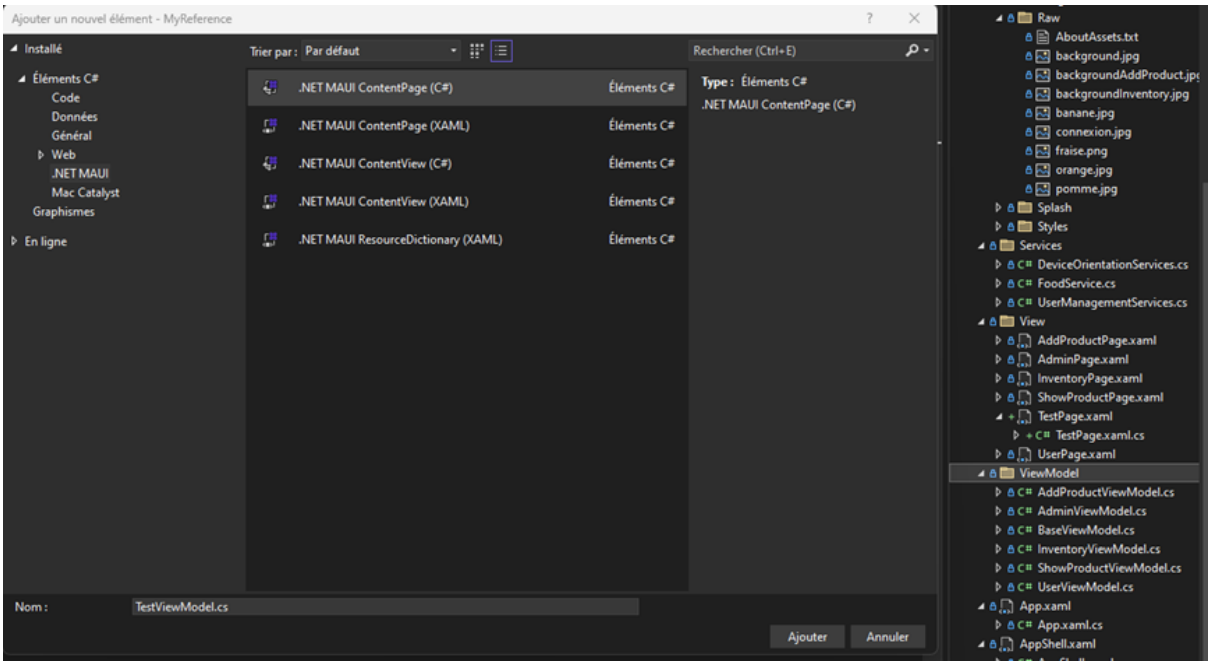
Ajouter une page

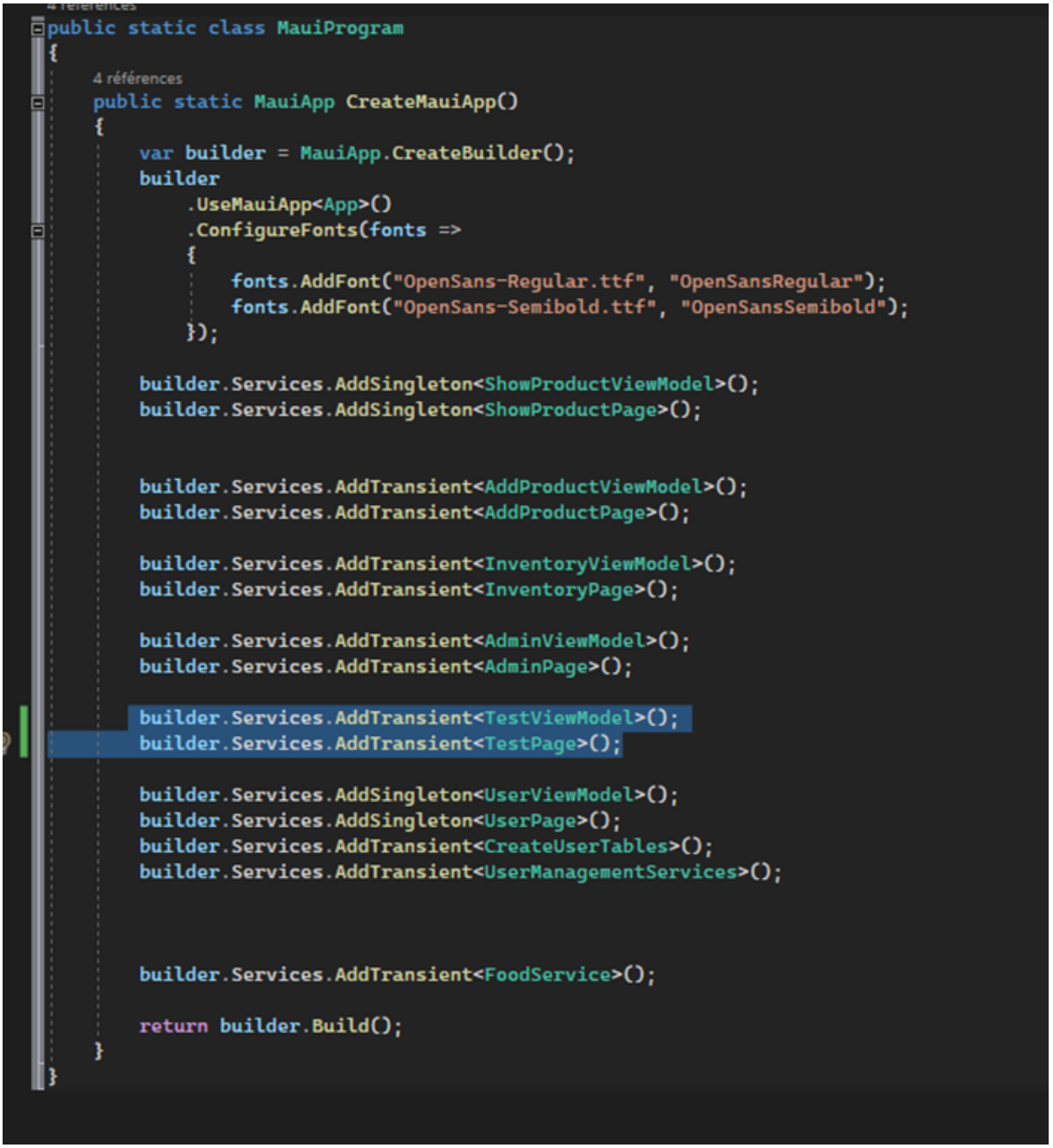
- View > nouvelle élément > MAUI > content page xaml > copier le content page racePage sur celle-ci et remplacer par le nom de la page même



- ViewModel > nouvelle element > classe c#



- MAUIProgramme > ajouter la page et le model



```
public static class MauiProgram
{
    4 références
    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");
            });

        builder.Services.AddSingleton<ShowProductViewModel>();
        builder.Services.AddSingleton<ShowProductPage>();


        builder.Services.AddTransient<AddProductViewModel>();
        builder.Services.AddTransient<AddProductPage>();

        builder.Services.AddTransient<InventoryViewModel>();
        builder.Services.AddTransient<InventoryPage>();

        builder.Services.AddTransient<AdminViewModel>();
        builder.Services.AddTransient<AdminPage>();

        builder.Services.AddTransient<TestViewModel>();
        builder.Services.AddTransient<TestPage>();

        builder.Services.AddSingleton<UserViewModel>();
        builder.Services.AddSingleton<UserPage>();
        builder.Services.AddTransient<CreateUserTables>();
        builder.Services.AddTransient<UserManagementServices>();



        builder.Services.AddTransient<FoodService>();

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

- AppShell.xaml.cs > ajouter le router de la page

```
public partial class AppShell : Shell
{
    1 référence
    public AppShell()
    {
        InitializeComponent();

        Routing.RegisterRoute(nameof(AddProductPage), typeof(AddProductPage));
        Routing.RegisterRoute(nameof(InventoryPage), typeof(InventoryPage));
        Routing.RegisterRoute(nameof(ShowProductPage), typeof(ShowProductPage));
        Routing.RegisterRoute(nameof(AdminPage), typeof(AdminPage));
        Routing.RegisterRoute(nameof(TestPage), typeof(TestPage));

    }
}
```

** Bien vérifier le xaml , xaml.cs et le ViewModel , que tout sois pareil au autre fichier

Modèle

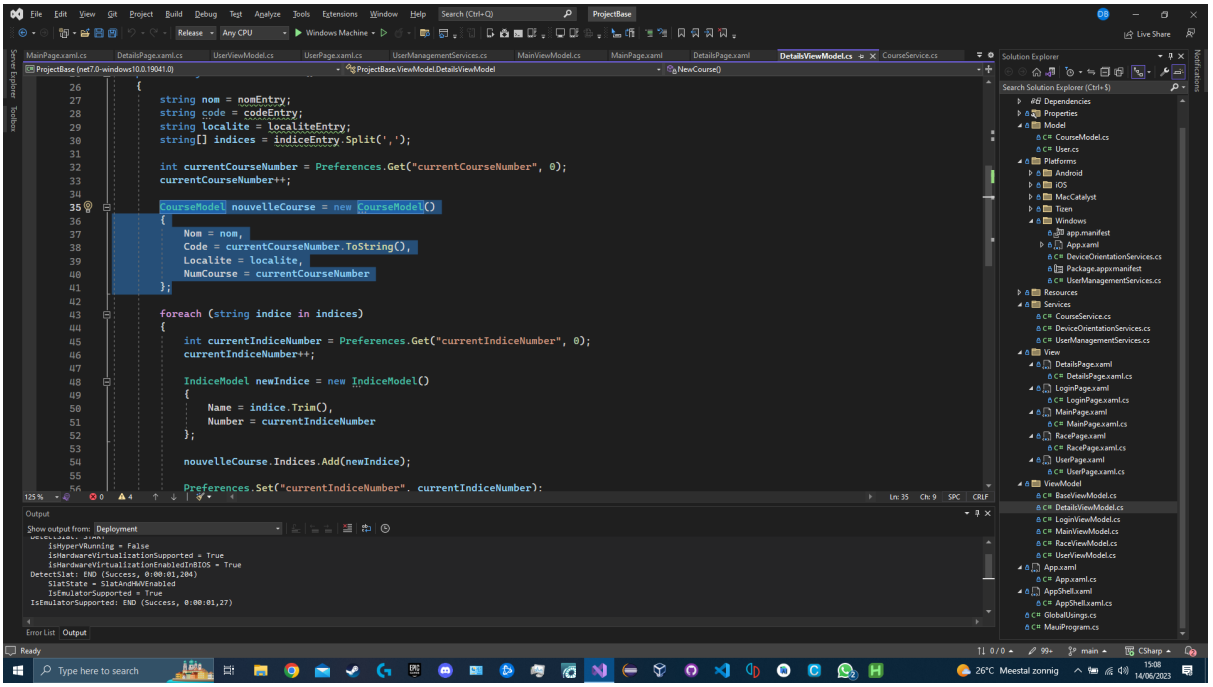
- Model > cree un nvx modèl en se basant sur courseModel

Représentation

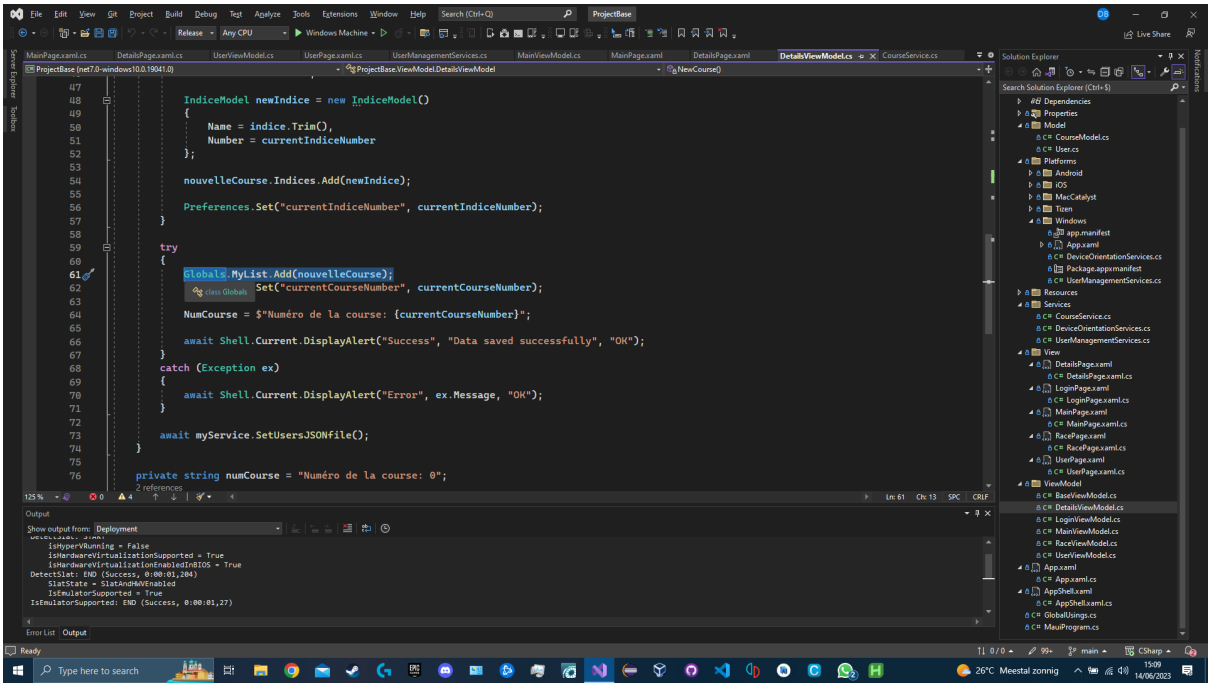
- Copier la CollectionView de MainPage.xaml

Ajout objet

DetailsViewModel.cs tu crée un nouvel Modèle

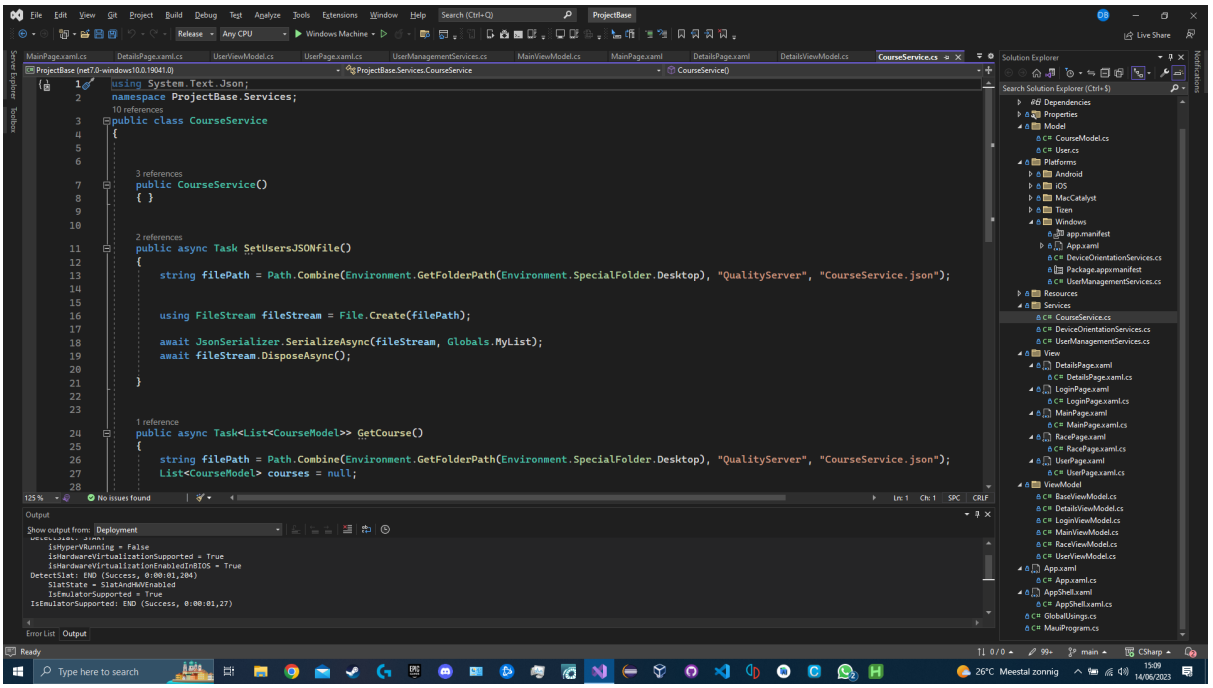


Puis tu l’ajoute dans la liste globale pour l’affichage



```
        IndiceModel newIndice = new IndiceModel()
        {
            Name = indice.Trim(),
            Number = currentIndiceNumber
        };

        nouvelleCourse.Indices.Add(newIndice);

        Preferences.Set("currentIndiceNumber", currentIndiceNumber);
    }

    try
    {
        Globals.MyList.Add(nouvelleCourse);
        Set("currentCourseNumber", currentCourseNumber);

        NumCourse = $"Numéro de la course: {currentCourseNumber}";

        await Shell.Current.DisplayAlert("Success", "Data saved successfully", "OK");
    }
    catch (Exception ex)
    {
        await Shell.Current.DisplayAlert("Error", ex.Message, "OK");
    }

    await myService.SetUsersJSONfile();
}

private string numCourse = "Numéro de la course: 0";
```

Puis tu vas dans Service > CourseService pour la sauvegarde dans le JSON.



```
using System.Text.Json;
namespace ProjectBase.Services;
public class CourseService
{


    public CourseService()
    { }


    public async Task SetUsersJSONfile()
    {
        string filePath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Desktop), "QualityServer", "CourseService.json");


        using FileStream fileStream = File.Create(filePath);

        await JsonSerializer.SerializeAsync(fileStream, Globals.MyList);
        await fileStream.DisposeAsync();

    }


    public async Task<List<CourseModel>> GetCourse()
    {
        string filePath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Desktop), "QualityServer", "CourseService.json");
        List<CourseModel> courses = null;
```

Nouvelle bd

Il y a des boutons dans la userPage qui appellent des méthodes dans UserViewModel via RelayCommand. Ces méthodes appellent des méthodes dans Platform > Windows UserManagmentService. Puis Services > UserManagmentService > CopierColler toute la page

Comptes utilisateurs

Global Using > Définir (IsConnected = false) Et (isAdmin = false) Ainsi Que le UserSet

```
26 références
public class Globals
{
    public static List<CourseModel> MyList = new List<CourseModel>();
    public static DataSet UserSet = new();
    public static bool isConnected = false;
    public static bool isAdmin = false;
}
```

Quand on appel une methode du mainViewModel on peux verifier si on est admin ou connecter

```
[RelayCommand]
0 références
async Task GoToDetailsPage(string data)
{
    if (!Globals.isConnected)
    {
        await Shell.Current.DisplayAlert("Database", "Not Connected", "OK");
        return;
    }

    await Shell.Current.GoToAsync(nameof(DetailsPage), true, new Dictionary<string, object>
    {
        {"Databc", data }
    });
}
```

Amélioration :

- La page de Login doit se trouver en premier plan
- myService.SetUsersJSONfile(); appelé sans "await"
- "await viewModel.LoginUser(nom, mdp);" => A utiliser en "RelayCommand"
- "Frame" à remplacer par "Border"
- ObservableProperty manipulé avec minuscule.. (NewCourse)
- SetUsersJSONfile => Manipulation de fichiers sans try/catch