TP .Net Maui n° 4 : Utilisation d'une base de données pour gérer des contacts

Objectif : Utiliser une base de données Sqlite dans une application .Net Maui

1 - Création de l'interface

Lancez « Visual Studio » et créez un nouveau projet de type « Application .NET MAUI » (.Net 8).
Lancez l'exécution en sélectionnant « Windows Machine ». Testez l'application.

Allez sous « Windows Machine » et faites « Gestionnaire d'appareils Android », pour lancer votre simulateur Android (Si il n'est pas configuré, faites « Nouveauté », sélectionnez « Pixel 2 (+Store), x86, Android 9 - API28 » et laissez les autres paramètres par défaut, faites ensuite « Créer ».)

Ajoutez un répertoire « Views » et un répertoire « Models » à votre projet.
Ajoutez en ressources les images fournies (Moodle).
Modifiez la couleur Primary à #005067 et la couleur Secondary à #E84D0D (Couleurs officielles charte de 3iL).

Dans « Views », ajoutez une nouvelle page « .NET MAUI ContentPage (XAML) » que vous nommez « vListe.xaml ». Toujours dans « Views », ajoutez une 2° page « .NET MAUI ContentPage (XAML) » que vous nommez « vContact.xaml ».

Ouvrez « AppShell.xaml » et modifiez le raccourci « `local` » pour qu'il pointe sur « `TPMauiApp4.Views` ».
Remplacez dans la balise « `ShellContent` » le « `DataTemplate` » pour qu'il pointe sur la page « `vListe` ».

Ouvrez « vListe.xaml », nommez votre page « Contacts », mettez les propriétés « `Spacing="20" Margin="10"` » au « `VerticalStackLayout` ». Ajoutez-y 1 Label centré horizontalement (`HorizontalOptions`), nommez le « `lbContacts` » avec les propriétés « `FontSize="Large"` » et « `TextColor="{StaticResource Primary}"` ».
Ajoutez avant votre « `VerticalStackLayout` », une ToolBar (`<ContentPage.ToolbarItems>`) avec 2 Items nommés « Ajouter » et « Supprimer » et interceptez leur méthode respective.

```
<ToolbarItem Text="Ajouter" Clicked="Add_Clicked"
             IconImageSource="{FontImage Glyph='A', Color=White, Size=22}" />
```

Testez votre application et vérifiez le bon comportement de vos 2 fenêtres.

2 - Implémentation de la base de données

Ouvrez le gestionnaire de package NuGet (Menu Projet) pour rechercher le package « sqlite-net-pcl » et l'ajouter la dernière version à votre projet. Attention à ne pas confondre avec le Package « SQLite.Net-PCL » (en majuscule avec un point à la place du 1° tiret) qui est déprécié. Ce Package a besoin de plusieurs dépendances, acceptez tout en cliquant sur « OK ».

Dans « Models », ajoutez une nouvelle classe « `cContact` » et passez là publique. Dans cette classe déclarez 3 variables publiques nommées « Id » de type entier, « Email » et « Nom » de type « `string` » avec « get/set » (`public int Id { get; set; }`). Incluez l'espane de nom « `SQLite` ».
Au-dessus de la propriété « Id » ajoutez la ligne : `[PrimaryKey, AutoIncrement]`

Que définit cette ligne ? : _____

Sous cette classe dans le même fichier, ajoutez une classe nommée « `cListeContact` ».
Dans celle-ci, définissez la variable privée :

```
private readonly SQLiteConnection _database;
```

Créez un constructeur publique qui admet une chaîne de caractères en paramètre nommée « `dbName` ». Dans ce constructeur
- récupérez en « var » le chemin renvoyé par :
  `Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), dbName);`
- initialisez la variable « `_database` » en appelant « `SQLiteConnection` » en « new » avec le chemin obtenu.
- appelez la méthode « `CreateTable` » de votre variable da la sorte : `CreateTable<cContact>()`

Que fait cette dernière ligne ? : _____

Ajoutez maintenant la méthode : `public List<cContact> GetList()`
qui retourne la liste des contacts présents dans la table : `_database.Table<cContact>().ToList();`

Ajoutez maintenant une méthode « `CreateContact` » qui accepte en paramètre un objet de la classe « cContact » qui insert ce contact dans la table (méthode `Insert` de votre variable base) et qui retourne son résultat.

Dans la méthode de votre bouton « Ajouter » de votre fenêtre « vListe », créez un objet « cContact » avec un nom et un @mail et appelez la méthode précédente pour l'ajouter.
Ajoutez une variable globale de type « cListeContact » : `public readonly cListeContact _ListeContact;`
Dans le constructeur de la fenêtre, ajoutez l'initialisation de cette variable :

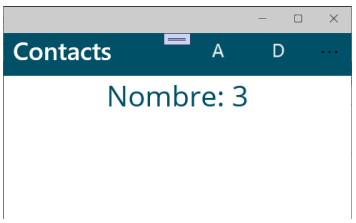
```
_ListeContact = new cListeContact("...");
```

(… a remplacé par le nom que vous voulez que porte votre base de données)
Ajoutez ensuite la méthode :

```
protected override void OnAppearing()
{
    base.OnAppearing();
}
```

Dans celle-ci affichez le nombre de lignes existant dans la base votre Label. Il suffit d'appeler la méthode « `Count` » de la collection renvoyée par « `GetList` ».
Enfin, en fin de méthode de votre bouton « Ajouter » appelez la méthode « `OnAppearing` ».

Vous pouvez maintenant tester l'accès à la base de données. Votre bouton doit renvoyer le nombre de contacts insérés dans la base avec le bouton « Ajouter ».



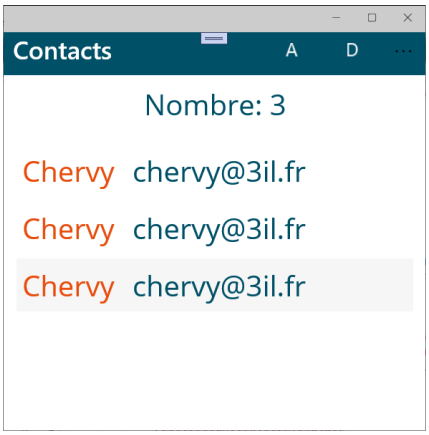
3 - Affichage et gestion des données

- Affichage des éléments de la base :

Etant donné que la base est liée à notre classe, il va être très facile d'afficher les éléments.
Dans votre fenêtre « vListe », ajoutez en dessous du Label une « `CollectionView` » avec :
- le nom `cvListeContact`
- la propriété : `SelectionMode="Single"`
- la balise `<CollectionView.ItemTemplate>` englobant une balise `<DataTemplate>` englobant elle-même un `<VerticalStackLayout>` qui englobe un `<HorizontalStackLayout>`

Dans celui-ci (`HorizontalStackLayout`) :
- mettez les propriétés « `Spacing="20" Margin="10"` »
- ajoutez 2 Labels avec les propriétés « `FontSize="Large" TextColor="{StaticResource Secondary}"` » (`Primary` pour le 2°) et la propriété « `Text="{Binding Nom}"` » pour le premier et « `Email` » pour le 2°.

Pour le chargement des données, il suffit dans «`OnAppearing`» de passer la collection de contacts renvoyée par « `GetList` » (de votre variable base) comme source de votre collection « `cvListeContact` » en utilisant « `ItemsSource` ».
Vous devez voir s'afficher l'élément ajoutés plusieurs fois au début.



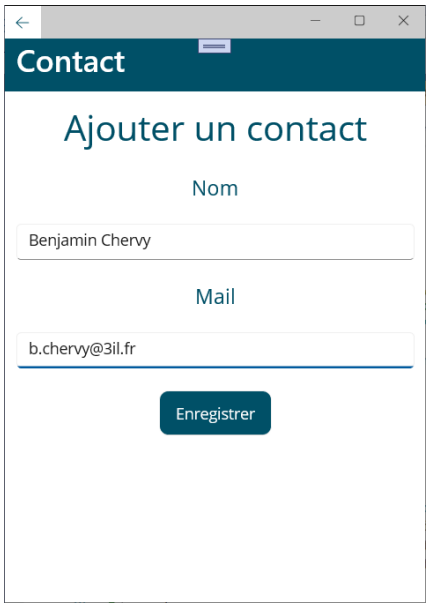
- Ajout d'un nouveau contact :

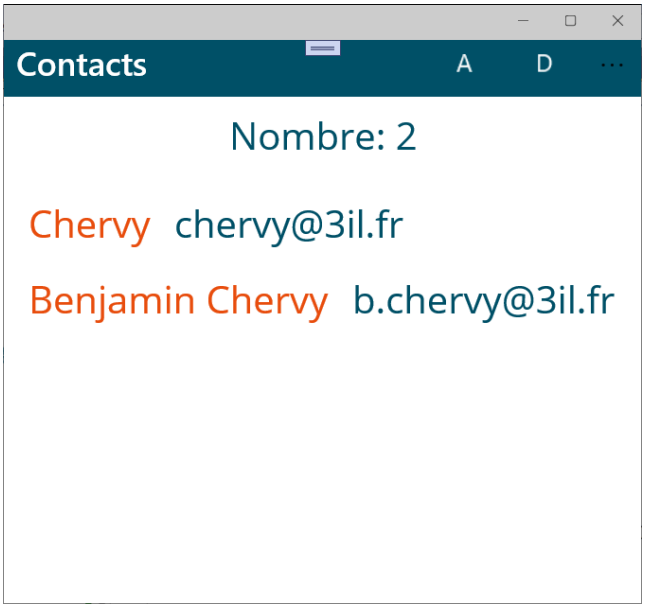
Ouvrez « vContact.xaml », modifiez le titre et ajoutez dans le « `VerticalStackLayout` » :
- 1 Label affichant « `Ajouter un contact` »
- 1 Label affichant « `Nom` » suivi d'un contrôle « `Editor` » que vous nommerez
- 1 Label affichant « `Mail` » suivi d'un contrôle « `Editor` » que vous nommerez
- 1 bouton « Enregistrer » avec sa méthode implémentée
Arrangez les espaces et les couleurs.

Pensez à implémenter la route vers « vContact » dans le constructeur de « vListe ».
```
Routing.RegisterRoute($"{nameof(vContact)}", typeof(vContact));
```

Dans « vContact.xaml.cs », dans la méthode du bouton, vous allez récupérer les données tapées par l'utilisateur, c'est-à-dire le nom et le mail du nouveau contact.
Il va être nécessaire de faire passer ces 2 informations à la fenêtre « vListe » car le code de la fenêtre « vContact » ne voit pas la variable représentant base de données.
Pour cela, il suffit de faire passer un paramètre type chaîne contenant ces informations. Pour cela :
- définissez dans « vListe » une variable de type chaîne avec un « setter »
```
public string Name
{
    set
    {
     }
}
```
- définissez un paramètre de requête qui peut être reçu par « vListe » (au-dessus de la déclaration de la classe de fenêtre) :
```
[QueryProperty(nameof(vListe.Name), nameof(vListe.Name))]
```
- dans la méthode du bouton, appelez « vListe » avec « `GoToAsync` » en lui passant comme paramètre la chaîne de caractères qui sera la concaténation des 2 informations (sVal) :
```
Shell.Current.GoToAsync($"{nameof(vListe)}?{nameof(vListe.Name)}={sVal}");
```
Sachant qu'il vous faut construire cette chaîne en concaténant le nom suivi d'un flag « $$ » et du mail.
- pensez aussi à spécifier la route vers « vListe »

Dans le Setter de « Name », il va falloir séparer le nom et le mail reçu (substring), puis créer un nouvel objet « cContact » pour ensuite appeler « `CreateContact` » de votre base.



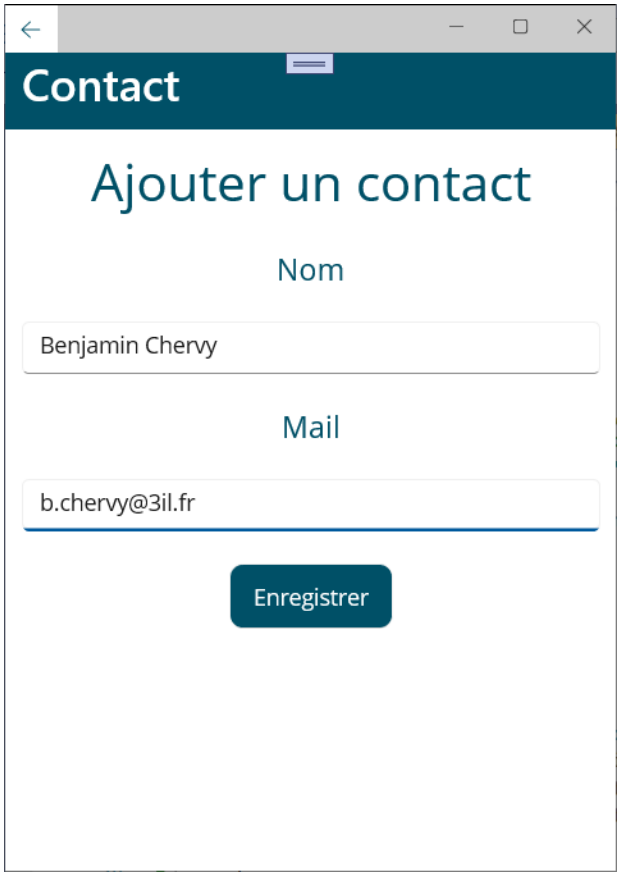


- Suppression d'un contact :

Pour supprimer un contact il suffit :
- de créer une méthode « `DeleteContact` » à la classe « cContact » qui appelle « `Delete` » de la variable base
- de vérifier lors du clic que « `SelectedItem` » de la collection de contact n'est pas nul
- de récupérer ce contact et d'appeler la méthode « `DeleteContact` »

- Modification d'un contact :

Il ne manque plus qu'à ajouter la modification d'un contact. Quelles pistes :
- créer une nouvelle fenêtre dédiée qui recevra le nom et le mail du contact sélectionné
- faites en sorte que « vListe » garde en mémoire de la valeur « Id » pendant l'affichage de la fenêtre de modification pour permettre la mise à jour
- votre variable de base de données présente une méthode « Update »


Contact
Ajouter un contact
Nom
Benjamin Chervy
Mail
b.chervy@3il.fr
Enregistrer