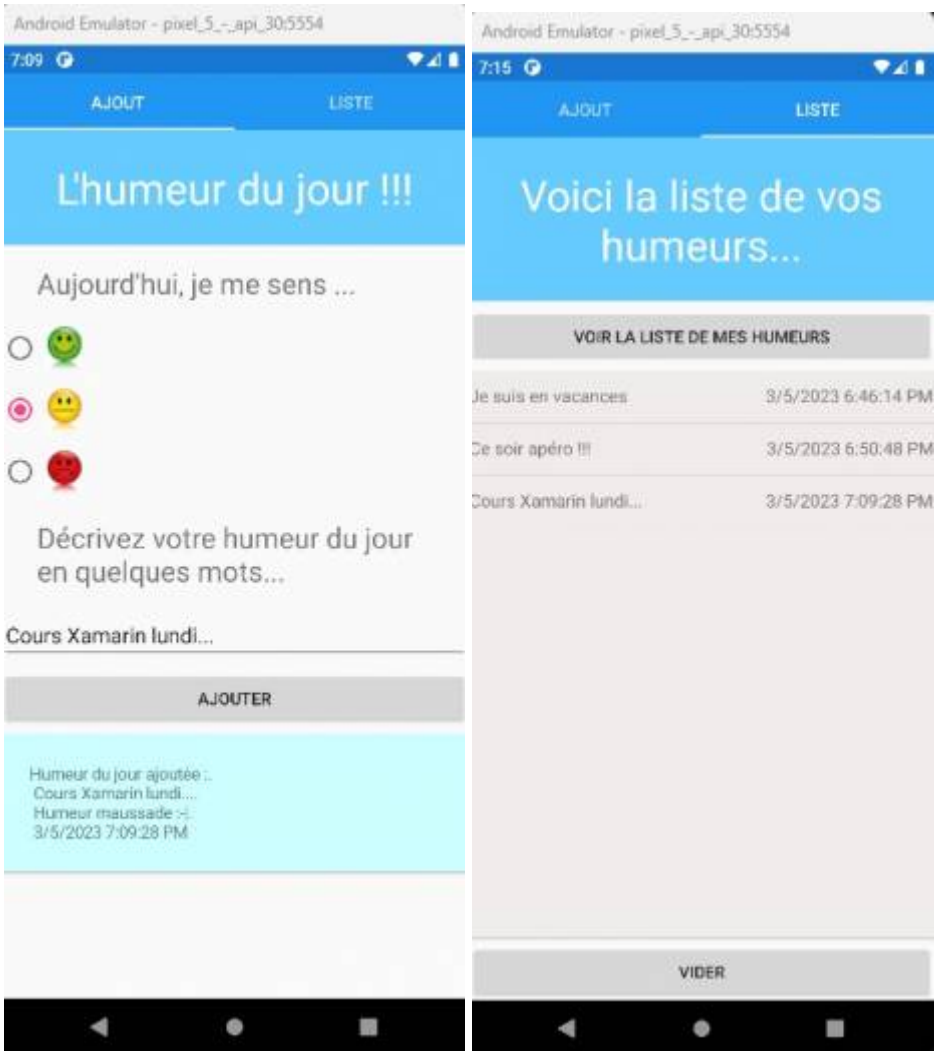
Xamarin : Développer une application mobile Xamarin.Forms avec C# - Base de données locale SQLite

PARTIE 1 : Prérequis et environnement de travail



1. Qu'est-ce que Xamarin

La documentation Microsoft pour Xamarin : https://learn.microsoft.com/fr-fr/xamarin/

Xamarin est une plateforme open source qui permet de générer des applications pour iOS, Android et Windows avec les technologies .NET.

- Xamarin permet de faire du développement natif pour Android, iOS et Windows en C#,

- accès aux API spécifiques à chaque plateforme et donc à toutes les fonctionnalités qu'elles offrent,
- partager le code via une librairie commune à tous les projets. Xamarin Forms est une couche d'abstraction qui permet de développer les interfaces utilisateurs pour les partager à travers toutes les plateformes.

Xamarin permet aux développeurs de partager en moyenne 90 % de leur application entre les plateformes. Ce modèle permet aux développeurs d'écrire toute leur logique métier dans un seul langage tout en obtenant une apparence et une convivialité natives sur chaque plateforme.

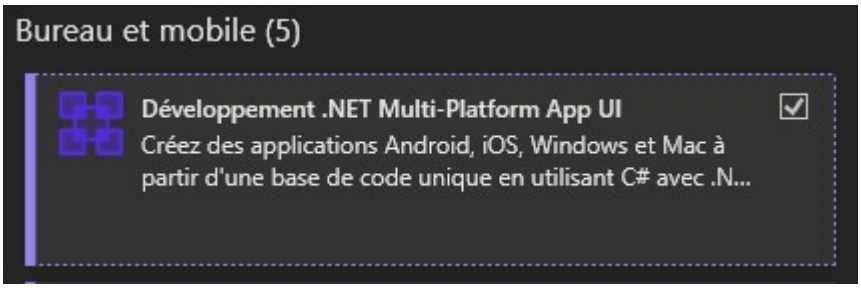
Les applications Xamarin peuvent être écrites sur PC ou Mac et être compilées dans des paquets d'application natifs, par exemple un fichier .apk sur Android ou un fichier .ipa sur iOS.

Xamarin.Forms

- Xamarin.Forms est une infrastructure d'interface utilisateur open source.
- Xamarin.Forms permet aux développeurs de créer des applications Xamarin.iOS, Xamarin.Android et Windows à partir d'une seule base de code partagée.
- Xamarin.Forms permet aux développeurs de créer des interfaces utilisateur en XAML en C#. Ces interfaces utilisateur sont affichées en tant que contrôles natifs performants sur chaque plateforme.

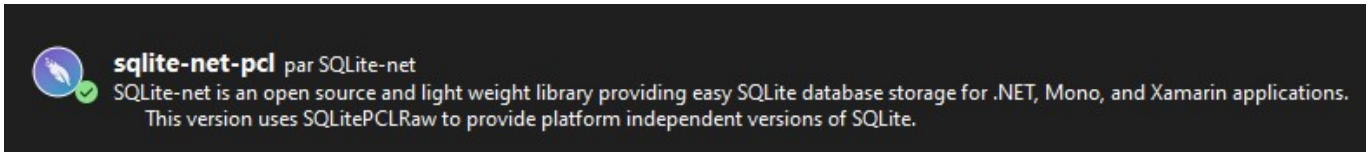
2. Installation de Xamarin

Ajout de Xamarin à Visual Studio :



3. Installer packages NuGet pour gérer la base SQLite

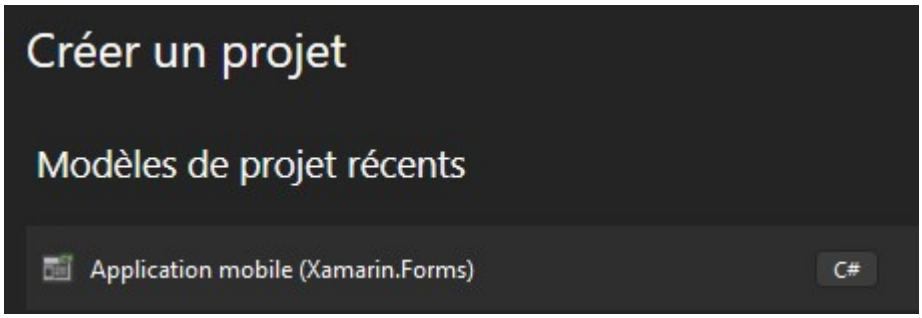
- Dans le menu : Projet → Gérer les packages NuGet... → Rechercher `sqlite-net-pcl`
- Installer sqlite-net-pcl

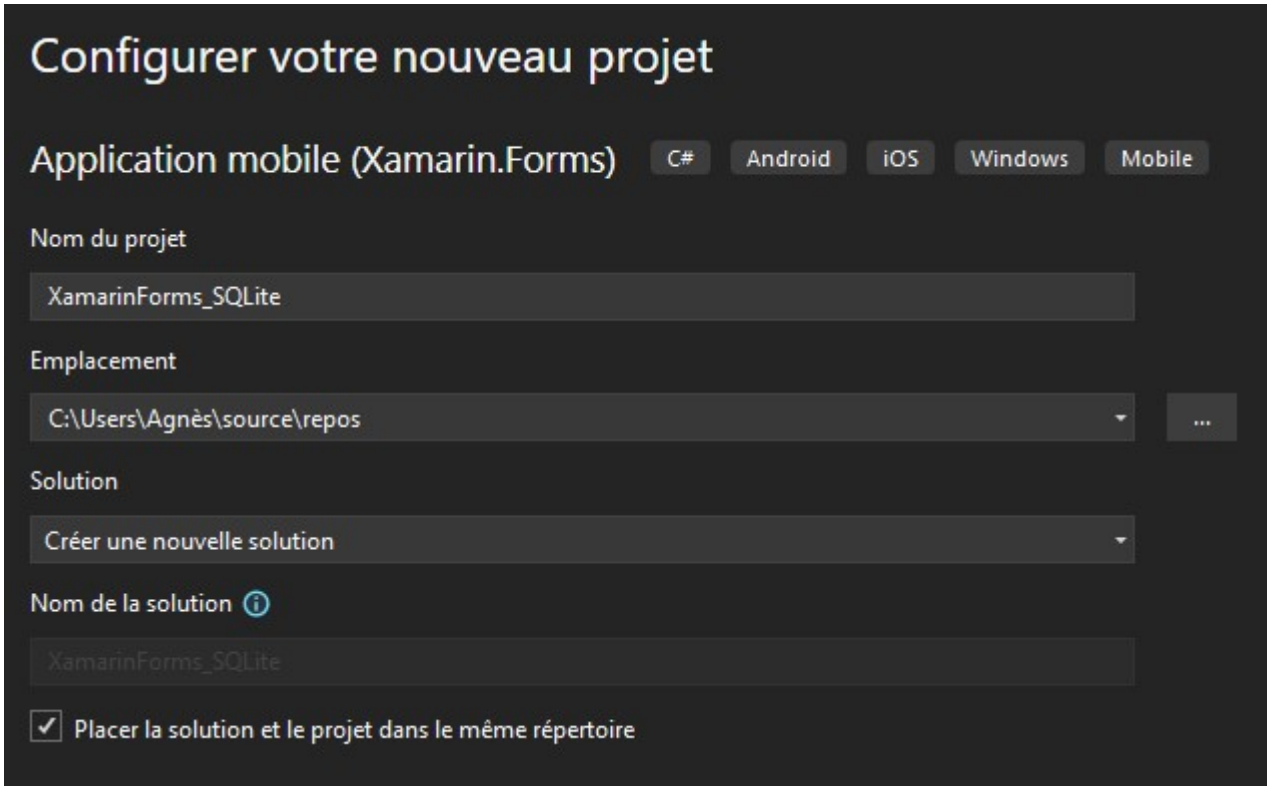


PARTIE 2 : L'application

ETAPE 1

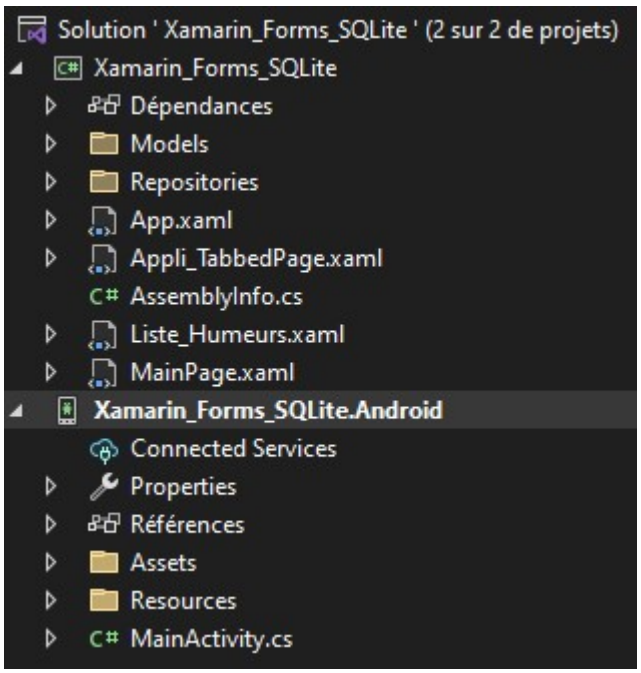
1. Créer un projet Xamarin.Forms





On peut voir :

- Le projet commun `Xamarin_Forms_SQLite` - Code C# commun à toutes les plateformes (Androiq, iOS, Windows)
- Le projet Android `Xamarin_Forms_SQLite.Android`



2. Créer de la base de données SQLite

Création d'une variable globale à l'application gérée par la classe `App.xaml.cs` pour définir l'emplacement de notre base de données SQLite.

Dans le fichier `App.xaml.cs` :

- string dbPath : nom de la variable contenant l'emplacement de notre base SQLite
- Path.Combine() : méthode permettant d'obtenir un chemin d'accès (Espace de noms : *using System.IO;*)
- FileSystem.AppDataDirectory : propriété qui obtient l'emplacement où les données d'application peuvent être stockées (Espace de noms : *using Xamarin. Essentiel;*)
- database.db3 : nom du fichier de la base de données SQLite

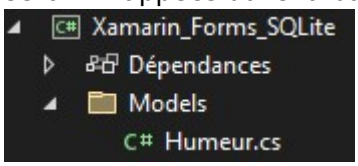
```
public partial class App : Application
{
    protected string dbPath = Path.Combine(FileSystem.AppDataDirectory, "database.db3");
```

3. Description des classes "métier" : Mapping avec la base de données SQLite

Classe Humeur

Création de la classe *Humeur* qui sera mappée dans la base de données SQLite.

Dans le projet commun Xamarin.Forms, ajouter un dossier `Models` qui contiendra les classes qui seront mappées dans la base de données SQLite.



La classe Humeur *[Table("Humeur")]* sera composée :

- `Id` de type int *[PrimaryKey, AutoIncrement]*
- `Commentaire` de type string *[MaxLength(50)]*
- `Note` de type int
- `DateAjout` de type DateTime

```
namespace Xamarin_Forms_SQLite.Models
{
    [Table("Humeur")]
    9 références
    public class Humeur
    {
        [PrimaryKey, AutoIncrement]
        0 références
        public int Id { get; set; }
```

Remarques :

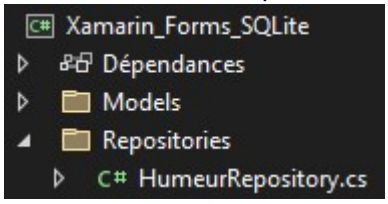
- La classe `Humeur` est publique : *public class Humeur{}*
- `Note` correspond à 1 si “Bonne humeur” et 0 si “Mauvaise humeur”
- On n'oublie pas les accesseurs { get; set; }

4. Création des classes d'accès (CRUD) à la base de données SQLite

Classe HumeurRepository

Création de la classe *HumeurRepository* qui contiendra les méthodes permettant les opérations (CRUD) de la classe *Humeur*.

Dans le projet commun Xamarin.Forms, ajouter un dossier `Repositories` qui contiendra les classes avec les méthodes permettant les opérations de manipulation de nos classes dans le dossier `Models`.



Variable globale de connexion SQLite

```
protected SQLiteAsyncConnection _connection;
```

On va utiliser une connexion en mode asynchrone.

- SQLite.SQLiteAsyncConnection(string dbPath) : construit une nouvelle SQLiteAsyncConnection et ouvre une base de données SQLite spécifiée par dbPath.
- using SQLite;

Remarques :

- La classe `HumeurRepository` est publique : *public class HumeurRepository{}*
- Ajout du constructeur de la classe HumeurRepository
- Ajout d'une méthode : public async Task AjouterHumeurAsync(...) pour ajouter un objet Humeur humeur dans la base de données
- Ajout d'une méthode : public async Task<List<Humeur» ListeHumeursAsync() pour récupérer une liste d'objets Humeur
- Ajout d'une méthode : public Task SupprimerAllHumeurs<T>() pour vider la base de données

Constructeur HumeurRepository

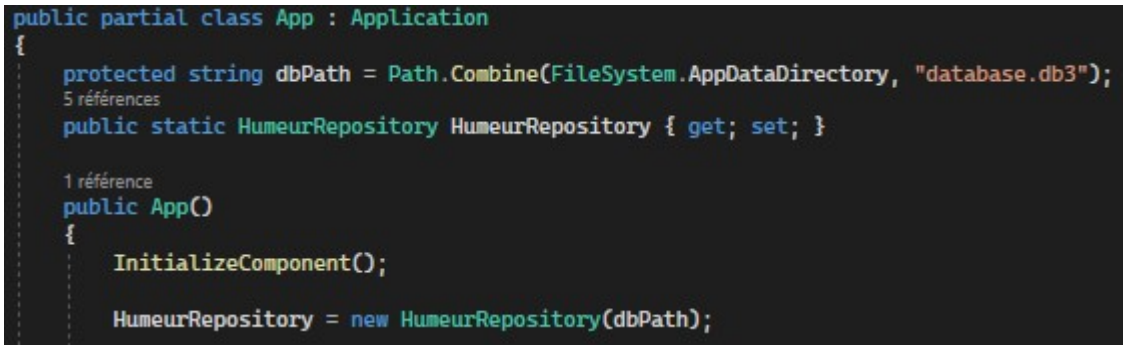
```
//En paramètre la variable dbPath contenant l'emplacement de la BD
public HumeurRepository(string dbPath){}
```

- _connection= new SQLiteAsyncConnection(dbPath); : création de l'objet _connexion
- _connection.CreateTableAsync<Humeur>(); : création de la table avec la connexion

Accès aux méthodes de la classe HumeurRepository

Dans la classe App.xaml.cs :

```
public static HumeurRepository HumeurRepository { get; set; }
```



```
public partial class App : Application
{
    protected string dbPath = Path.Combine(FileSystem.AppDataDirectory, "database.db3");
    5 références
    public static HumeurRepository HumeurRepository { get; set; }

    1 référence
    public App()
    {
        InitializeComponent();

        HumeurRepository = new HumeurRepository(dbPath);
```

5. Création des méthodes de la classe HumeurRepository

Définition de la méthode *AjouterHumeurAsync* de la classe *HumeurRepository*

Méthode AjouterHumeurAsync

```
public async Task AjouterHumeurAsync(-- paramètres --){}
```

- La programmation asynchrone est basée sur les objets Task et Task<T>, qui modélisent les opérations asynchrones. Ces objets sont exposés à l'aide des mots clés async et await.
- Le mot clé async définit une méthode comme asynchrone, ce qui vous permet d'utiliser le mot clé await dans le corps de la méthode.
- Quand le mot clé await est utilisé, il suspend la méthode d'appel et cède le contrôle à son appelant jusqu'à ce que la tâche awaited soit terminée.
- Très intéressant pour ne pas bloquer l'affichage dans le cas où le chargement des données est long (nombre d'enregistrements importants)

On va utiliser un try-catch :

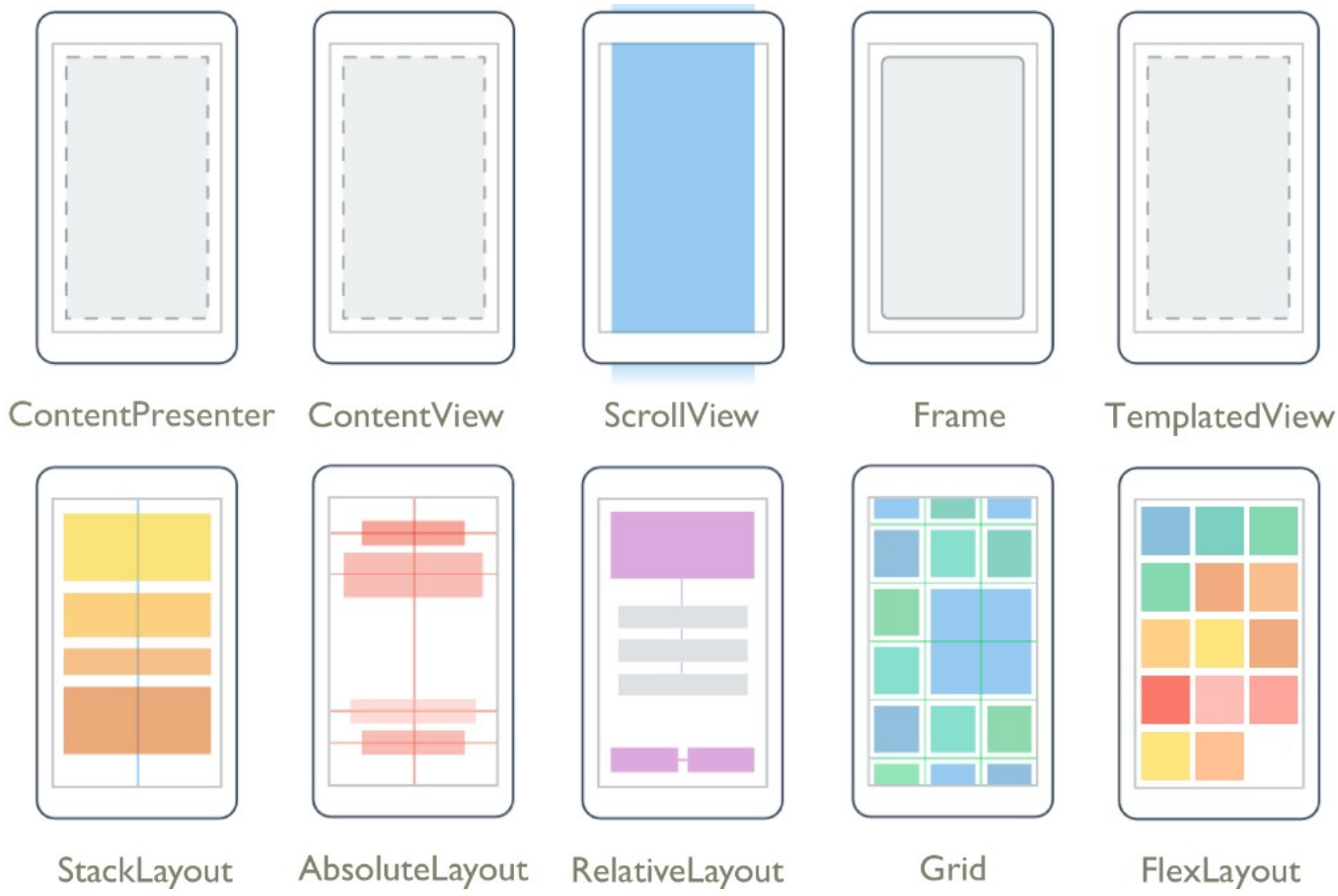
```
try
{
   nbHumeurs = await _connection.InsertAsync(new Humeur { Commentaire =
commentaire, Note = note, DateAjout = dateAjout });
   // Gestion d'un message à afficher
   Message = $"Humeur du jour ajoutée : {commentaire}.\n {monHumeur}.\n
{dateAjout}";
}
catch (Exception e)
{
   Message = $"Impossible d'ajouter l'humeur : {commentaire}.\n Erreur :
```

```
{e.Message}";
}
```

5. L'interface avec XAML

XAML : eXtensible Application Markup Language permet aux développeurs de définir des interfaces utilisateur dans applications Xamarin.Forms à l'aide de balisage plutôt que de code.

- Les dispositions sont utilisées pour composer des contrôles d'interface utilisateur dans des structures visuelles.
- Un Layout contient généralement une logique pour définir la position et la taille des éléments enfants dans les applications Xamarin.Forms.
- Dispositions avec contenu unique : ContentView, Frame, ScrollView, TemplatedView, ContentPresenter
- Dispositions avec plusieurs enfants : StackLayout, Grid, AbsoluteLayout, RelativeLayout, FlexLayout
- Voir documentation Xamarin.Forms : https://learn.microsoft.com/fr-fr/xamarin/xamarin-forms/user-interface/



- Réaliser l'interface utilisateur de la page principale.
- Créer l'événement *Clicked* du bouton pour l'ajout d'une humeur
- Créer la méthode associée à l'événement *HumeurButton_Clicked*

MainPage.xml

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Xamarin_Forms_SQLite.MainPage">
    <StackLayout>
        <Label Text="L'humeur du jour !!!" HorizontalTextAlignment="Center"
/>
        <Label Text="Aujourd'hui, je suis de ..." />
        <RadioButton x:Name="bonneHumeurRadio" Content="bonne humeur"
IsChecked="True"/>
        <RadioButton x:Name="mauvaiseHumeurRadio" Content="mauvaise
humeur"/>
        <Label Text="Décrivez votre humeur du jour en quelques mots..." />
        <Entry x:Name="humeurEntry" />
        <Button x:Name="humeurButton" Text="Ajouter" />
        <Label x:Name="message" />
    </StackLayout>
</ContentPage>
```

Le bouton Ajouter

On ajoute l'attribut Clicked="HumeurButton_Clicked" dans le fichier xaml pour gérer l'événement (gestionnaire d'événements).

```
<Button x:Name="humeurButton" Text="Ajouter"
Clicked="HumeurButton_Clicked"/>
```

L'événement HumeurButton_Clicked

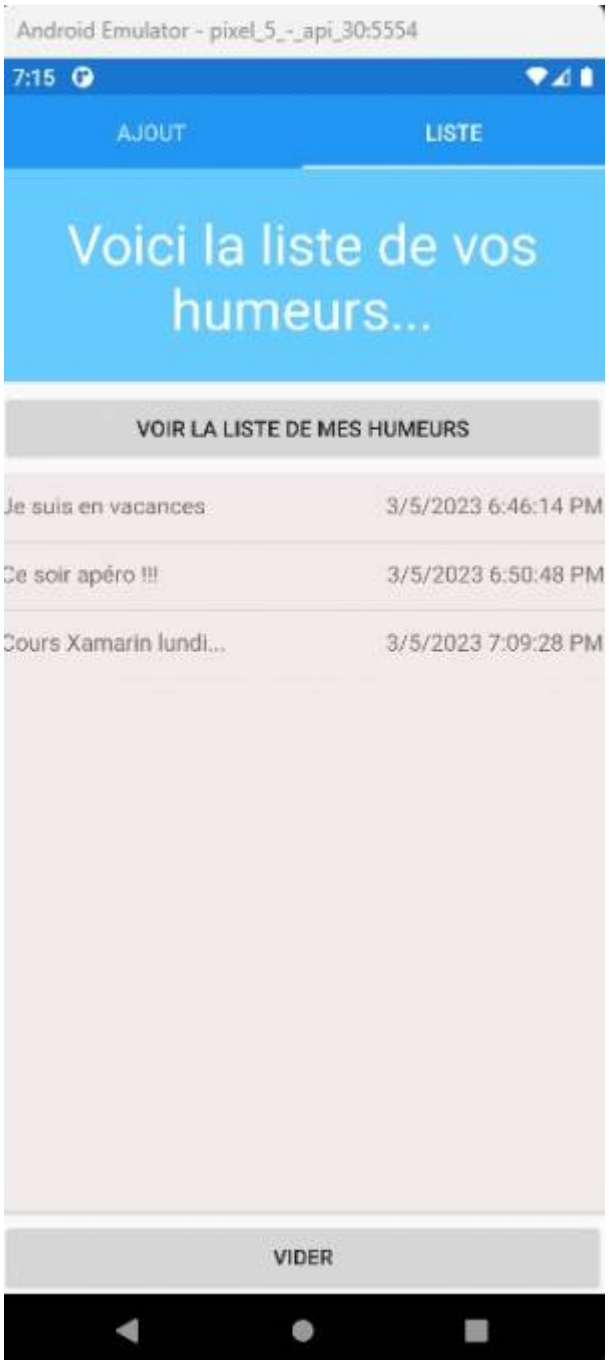
Dans le fichier MainPage.xaml.cs, on déclare la méthode :

```
private async void HumeurButton_Clicked(object sender, EventArgs e)
{
   // Appel de la méthode AjouterHumeurAsync de la classe HumeurRepository
   await App.HumeurRepository.AjouterHumeurAsync(-- paramètres --);
}
```

Await et async permettent d'indiquer au compilateur:

- la méthode pour lesquelles l'exécution sera asynchrone en utilisant async : `AjouterHumeurAsync`
- le code où on va attendre la fin de l'exécution d'une tâche en utilisant await : `await App.HumeurRepository.AjouterHumeurAsync`

ETAPE 2



1. Interface : Création du menu à Onglets

- Réaliser l'interface utilisateur pour le menu de l'application : menu à Onglets
- Créer l'interface XAML associée

Ajouter un nouvel élément au projet → Page à onglets : Appli_TabbedPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:vues="clr-namespace:Xamarin_Forms_SQLite"
            x:Class="Xamarin_Forms_SQLite.Appli_TabbedPage">
  <!--Pages can be added as references or inline-->
    <TabbedPage.Children>
        <vues:MainPage Title="AJOUT" />
        <vues:Liste_Humeurs Title="LISTE" />
    </TabbedPage.Children>
</TabbedPage>
```

2. Interface : Création de la page de gestion de la liste des humeurs

- Réaliser l'interface utilisateur de gestion de la liste des humeurs
- Créer l'interface XAML associée

Ajouter un nouvel élément au projet → Page : Liste_Humeurs.xaml

Liste_Humeurs.xaml

On va utiliser l'élément <ListView> (peut-être voir <CollectionView> ???)

```
<ListView
   SelectionMode="Single">
   <ListView.ItemTemplate>
      <DataTemplate>
         <ViewCell>
         </ViewCell>
      </DataTemplate>
   </ListView.ItemTemplate>
</ListView
```

Afin de remplir la liste, on va utiliser la liaison des données avec un objet : {Binding propriété}
Exemple : <Label Text="{Binding Commentaire}" />

3. Le bouton Voir la Liste des humeurs : Affichage de la ListView

Ajout de la méthode ListeHumeursAsync dans la classe HumeurRepository

```
public async Task<List<Humeur>> ListeHumeursAsync(){}
```

- On va retourner une liste d'humeurs : List<Humeur>
- On va utiliser la méthode ToListAsync() à travers la connexion : _connection.Table<Classe>()
- Ne pas oublier le try-catch

Le bouton Voir la liste de mes humeurs

L'événement ListeHumeurButton_Clicked

On remplit la <ListView> grâce à sa propriété ItemsSource qui va remplir les zones de liaison créées avec le Binding.

4. Supprimer la liste des humeurs

Ajout de la méthode SupprimerAllHumeurs dans la classe HumeurRepository

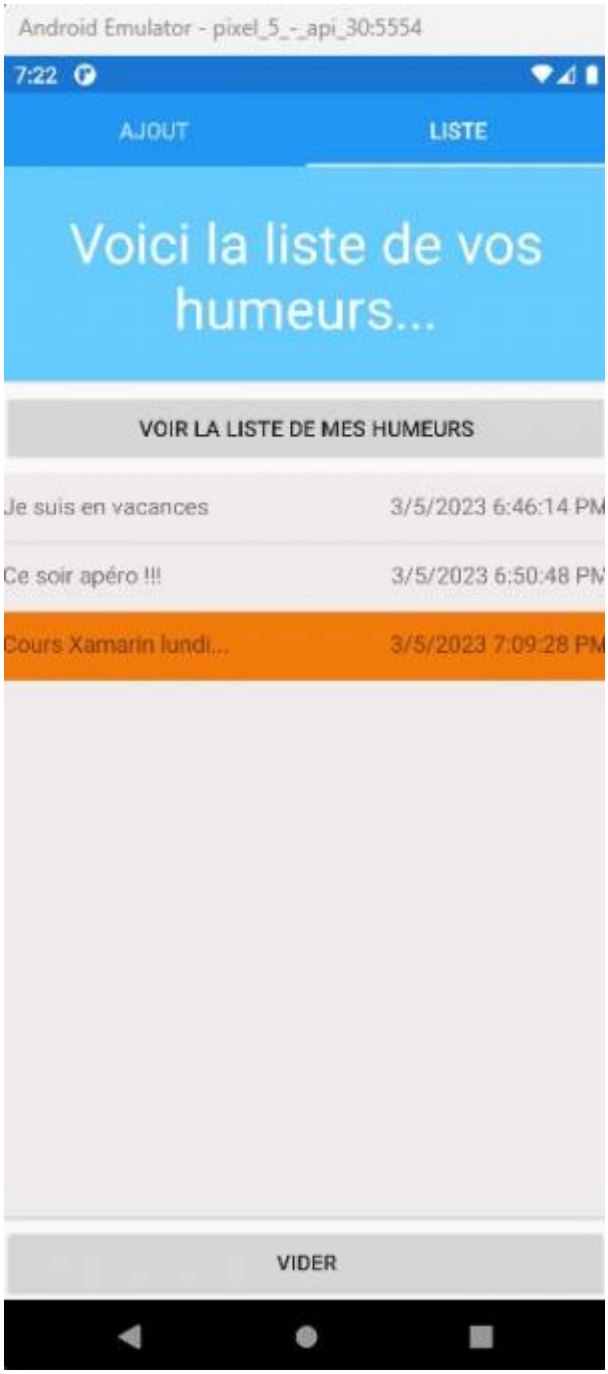
```
public Task SupprimerAllHumeurs<T>(){}
```

- On va utiliser la méthode DeleteAllAsync<Classe>() à travers la connexion

Le bouton Vider

L'événement ViderButton_Clicked

5. Supprimer une humeur



Ajout de la méthode SupprimerUneHumeur dans la classe HumeurRepository

- On supprime une humeur avec la méthode DeleteAsync à travers la connexion.

L'événement HumeursListView_ItemTapped

On ajoute l'attribut ItemTapped="HumeursListView_ItemTapped" à la ListView pour gérer l'événement (gestionnaire d'événements).

- Il faudra récupérer l'élément sélectionné dans la ListVew

```
ListView listeHumeurs = (ListView)sender;
Humeur humeur = (Humeur)listeHumeurs.SelectedItem;
```

- DisplayAlert : Présente à l'utilisateur de l'application une boîte de dialogue d'alerte