**Xamarin MATERIALS**

1. **Layout**

In Xamarin .NET MAUI, you can use Layout concepts to determine how user interface elements (such as buttons, text, images, etc.) will be arranged and displayed on the screen. There are several different types of Layouts you can use in Xamarin .NET MAUI to design your app's user interface, including: StackLayout, GridLayout, AbsoluteLayout.

* StackLayout

StackLayout is a UI component that allows you to stack child elements either vertically or horizontally. It is one of the simplest layout containers in .NET MAUI and is useful for organizing basic UI elements.

<StackLayout x:Name="layoutName">

<Label Text="Hello, MAUI!" />

<Button Text="Click Me" />

<Entry Placeholder="Type here" />

</StackLayout>

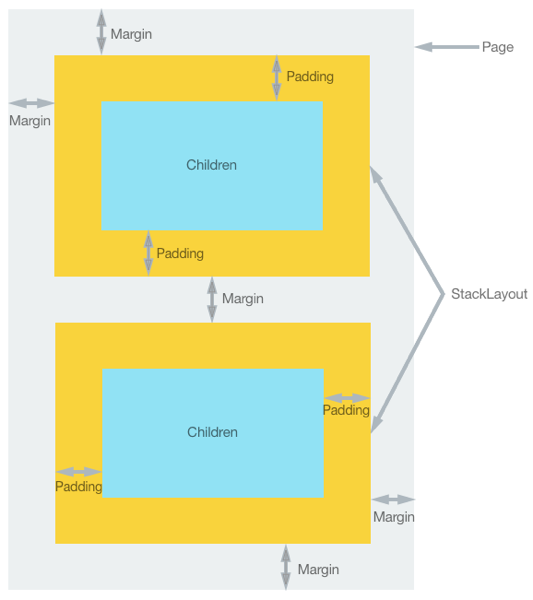
There are two way to order elements in a stack layout:

|  |  |
| --- | --- |
| Screenshot of a vertically oriented StackLayout | Screenshot of a horizontally oriented StackLayout |
| Vertical orientation  The default orientation is Vertical, without specifying the Orientation property  <**StackLayout**>  <!-- Child elements will be stacked vertically -->  </**StackLayout**> | Horizontal orientation  <**StackLayout** Orientation="Horizontal">  <!-- Child elements will be stacked horizontally -->  </**StackLayout**> |

Importance usage

|  |  |
| --- | --- |
| **Property** | **Description** |
| Orientation | Determines the stacking direction ("Horizontal" or "Vertical"). |
| Spacing | The spacing between child elements. |
| HorizontalOptions | How the StackLayout should be placed within the horizontal space of its parent (Start, Center, End, Fill, CenterAndExpand, etc.). |
| VerticalOptions | How the StackLayout should be placed within the vertical space of its parent (Start, Center, End, Fill, CenterAndExpand, etc.). |

* Padding vs Margin



* The [Margin](https://learn.microsoft.com/en-us/dotnet/api/xamarin.forms.view.margin#xamarin-forms-view-margin) property represents the distance between an element and its adjacent elements, and is used to control the element's rendering position, and the rendering position of its neighbors. Margin values can be specified on [layout](https://learn.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/controls/layouts) and [view](https://learn.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/controls/views) classes.
* The [Padding](https://learn.microsoft.com/en-us/dotnet/api/xamarin.forms.layout.padding#xamarin-forms-layout-padding) property represents the distance between an element and its child elements, and is used to separate the control from its own content. Padding values can be specified on [layout](https://learn.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/controls/layouts) classes.

<StackLayout Padding="0,20,0,0">

<Label Text="Xamarin.Forms" Margin="20" />

<Label Text="Xamarin.iOS" Margin="10, 15" />

<Label Text="Xamarin.Android" Margin="0, 20, 15, 5" />

</StackLayout>

The numbers in the list is located with the following position: left, top, right, bottom

Learning material:

<https://learn.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/layouts/stacklayout>

1. Label

A Label is used to display static text. You simply set the Text property to display text content.

<Label Text="Hello, Xamarin Maui!" />

|  |  |
| --- | --- |
| Property | Description |
| Text | The text displayed in the Label. |

Code behide

*void SetLabelText(string text)*

*{*

*// Assign data to the Label from code-behind*

*myLabel.Text = text;*

*}*

1. Button

A Button allows users to perform specific actions. You can set the button's text and handle the event when the button is clicked.

<Button x:Name="btName" Text="Click Me" Clicked="OnButtonClicked" />

|  |  |
| --- | --- |
| **Property** | **Description** |
| Text | The text displayed on the button. |
| Clicked | Event that occurs when the button is clicked. You just need to type the name for the event, then right click choose go to definition then the eventlistener will be created for you |

Code behide

*void OnButtonClicked(object sender, EventArgs e)*

*{*

*// Handle the event when the button is clicked*

*// Example: Display an alert*

*DisplayAlert("Alert", "Button clicked!", "OK");*

*}*

1. Entry (TextBox)

An Entry is an input field that allows users to enter text or numbers. You can use the Placeholder property to display a hint to the user.

<Entry x:Name="entryName" Placeholder="Type here" />

|  |  |
| --- | --- |
| **Property** | **Description** |
| Text | The text or number entered by the user. |
| Placeholder | Text displayed when the input field is empty. |
| TextChanged | Event that occurs when the text in the Entry changes. |

Code behide

*void OnTextChanged(object sender, TextChangedEventArgs e)*

*{*

*// Handle the event when text in the Entry changes*

*// Example: Process the entered data*

*string enteredText = e.NewTextValue;*

*// Process the enteredText here*

*}*

1. Image

An Image allows you to display images. You can use the Source property to specify the path to the image and WidthRequest and HeightRequest to define the size.

<Image x:Name="imgName" Source="icon.png" WidthRequest="50" HeightRequest="50" />

|  |  |
| --- | --- |
| **Property** | **Description** |
| Source | Path to the image. icon.png reference to an image in |
| WidthRequest | Width of the image. |
| HeightRequest | Height of the image. |

Code behide

*void SetImageSource(string imagePath)*

*{*

*// Assign the image path from code-behind*

*myImage.Source = imagePath;*

*}*

1. ListView

A ListView allows you to display a scrollable list of data. You need to set the ItemsSource to specify the data source and create a DataTemplate to format how each data item is displayed.

<ListView x:Name="contactListView" ItemTapped="OnContactTapped" RowHeight="100">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<StackLayout Padding="10">

<Label Text="{Binding Name}" FontSize="20" />

<Label Text="{Binding PhoneNumber}" />

<Label Text="{Binding Address}" />

</StackLayout>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

bind data to the listview

Create a class Contact

public class Contact

{

public string Name { get; set; }

public string PhoneNumber { get; set; }

public string Address { get; set; }

}

The binding name should be equal to the properties in the class

Then create a list of Contacts

contacts = new List<Contact>

{

new Contact { Name = "John Doe", PhoneNumber = "123-456-7890", Address = "123 Main St" },

new Contact { Name = "Jane Smith", PhoneNumber = "987-654-3210", Address = "456 Elm St" },

new Contact { Name = "Alice Johnson", PhoneNumber = "555-123-4567", Address = "789 Oak St" }

};

contactListView.ItemsSource = contacts;

Important properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| ItemsSource | Source data |
| RowHeight | Height of each row |
| ItemSelected | This event happened when a row is selected |
| ItemTapped | This event happened when a row is slightly tapped |

1. Picker (Select Box)

A Picker allows you to select an item from a list of options. You need to set the ItemsSource to specify the list of options and the Title to display a title for the select box.

*<Picker Title="Select an item">*

*<Picker.ItemsSource>*

*<x:Array Type="{x:Type x:String}">*

*<x:String>Value 1</x:String>*

*<x:String>Value 2</x:String>*

*<x:String>Value 3</x:String>*

*</x:Array>*

*</Picker.ItemsSource>*

*</Picker>*

|  |  |
| --- | --- |
| **Property** | **Description** |
| Title | The title(caption) of the select box. |
| ItemsSource | The list of options for the select box. |
| SelectedIndex | The index of the selected item. |
| SelectedIndexChanged | Event that occurs when the selected item changes.  You can add this property to the item with a value, for example OnSelected and then righr click and choose go to Definition then Visual studio will create for you |

Code behide

*void OnPickerSelectedIndexChanged(object sender, EventArgs e)*

*{*

*var selectedValue = myPicker.SelectedItem.ToString();*

*// Handle the selectedValue here*

*}*

* *Populate an empty Picker through code behide*

*XAML file*

<StackLayout>

<Picker x:Name="myPicker"

Title = "Select a monkey"

/>

<Button Text="Show Selected Value" Clicked="OnButtonClicked" />

</StackLayout>

*Code behide*

public MainPage()

{

InitializeComponent();

List<string> options = new List<string>

{

"Option 1",

"Option 2",

"Option 3"

};

myPicker.ItemsSource = options;

}

void OnButtonClicked(object sender, EventArgs e)

{

string selectedValue = myPicker.SelectedItem.ToString(); //get selected data

DisplayAlert("Selected Value", $"The selected value is: {selectedValue}", "OK");

}

1. AlertDialog

AlertDialog is a basic dialog for displaying a message or alert to the user

bool result = await DisplayAlert("Confirmation", "Do you want to continue?", "Yes", "No");

if (result)

{

// User clicked "Yes," handle the affirmative action

// ...

}

else

{

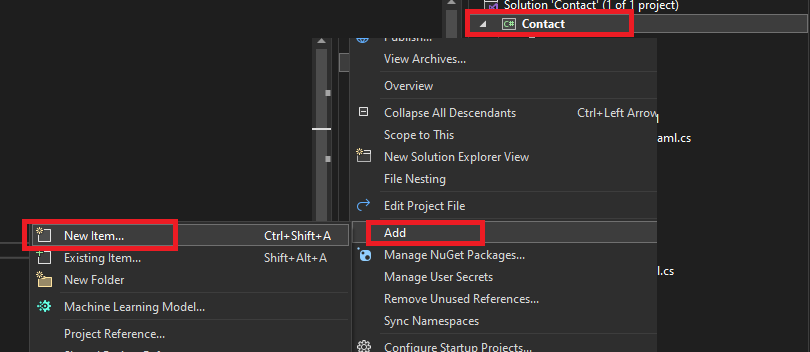
// User clicked "No," handle the negative action or do nothing

// ...

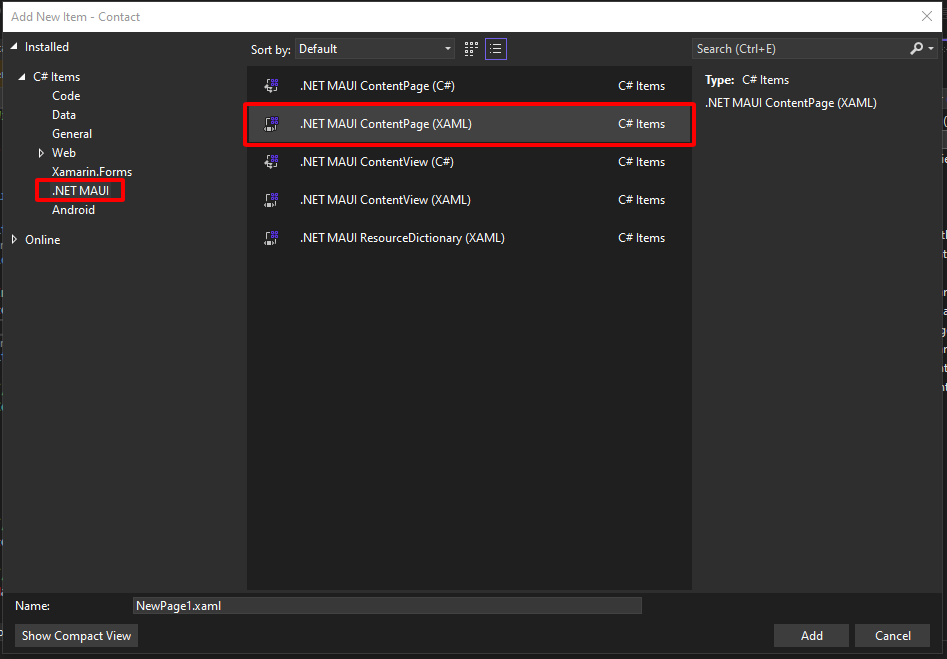
}

1. Page navigation

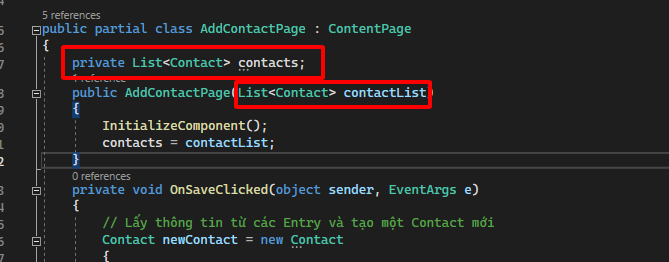
Step1: Create a new page



Right click on the project and choose Add\New Item



Step2: add variables or parameter to the constructor of the new page if you want to send data



Step3: use Navigation to call the page

To navigate to one page use

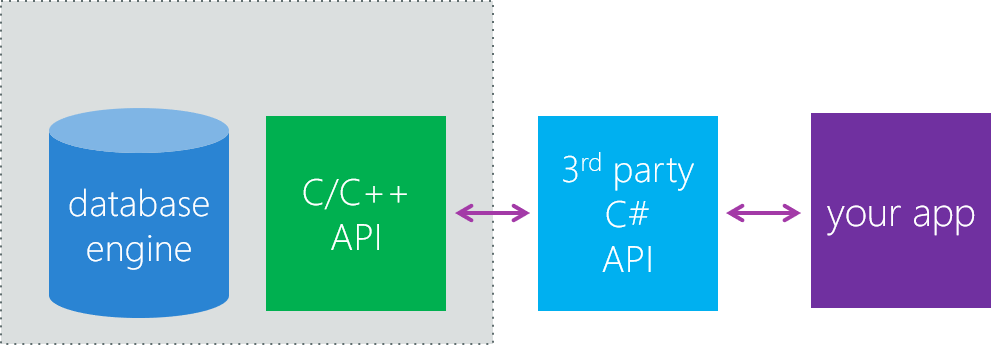
await Navigation.PushAsync(new TargetPage());

You can add some parameter to the constructor of the page so that you can send data

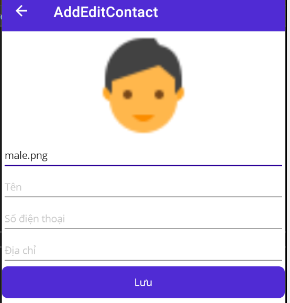
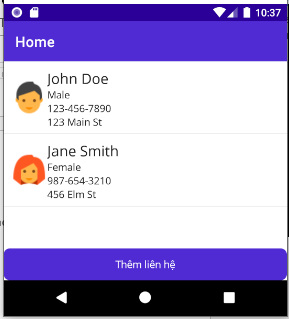
Go back to the page that call

Navigation.PopAsync();

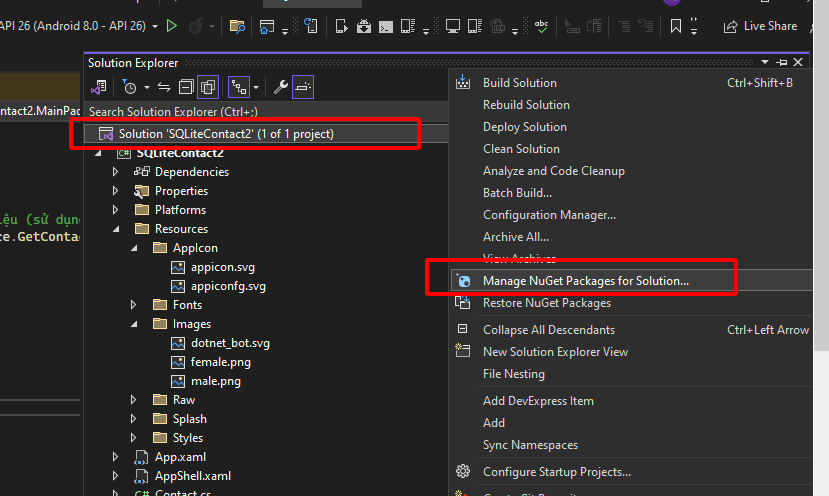
1. SQLite



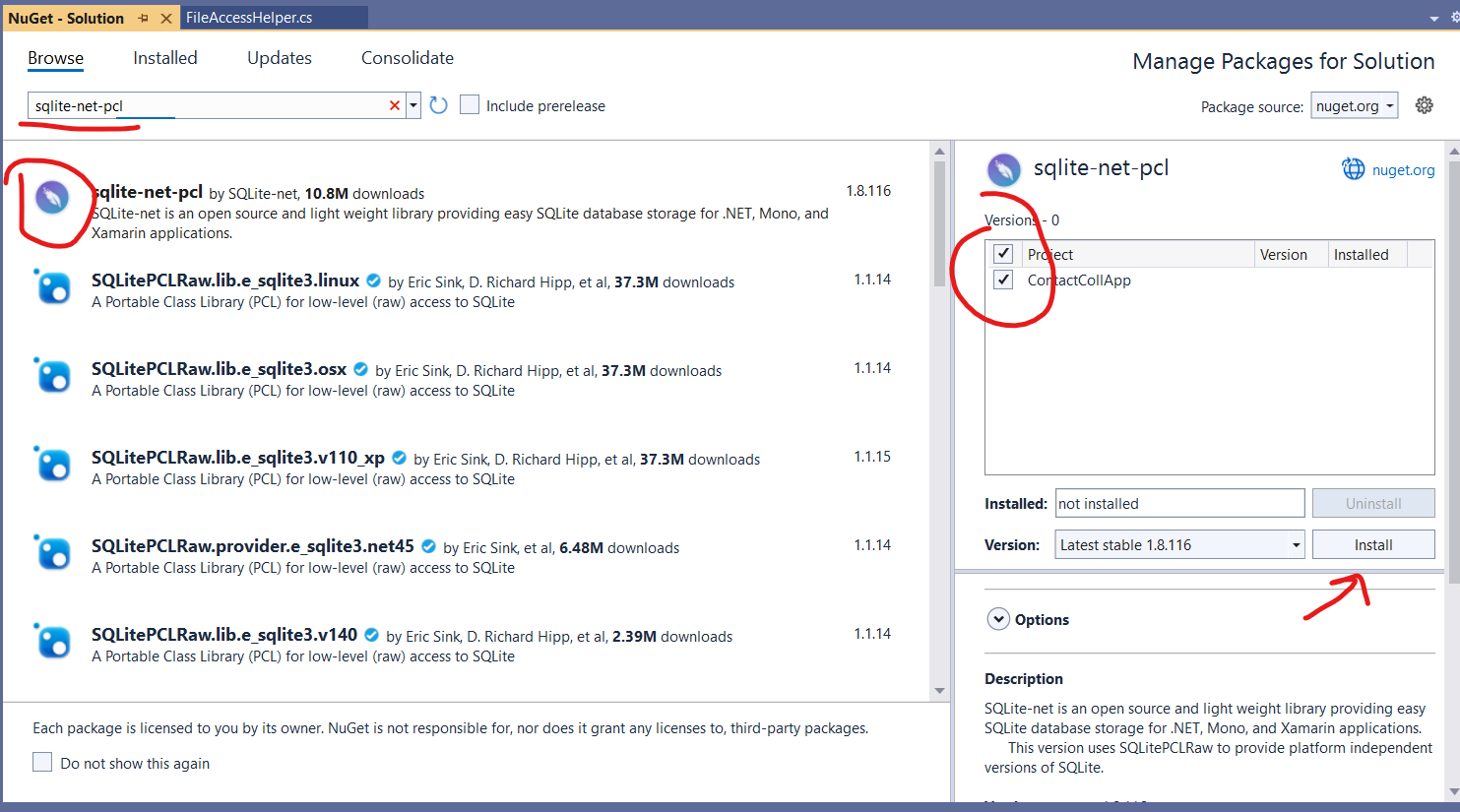
In this part, We will make an application to manage contact as an example for using sqlite



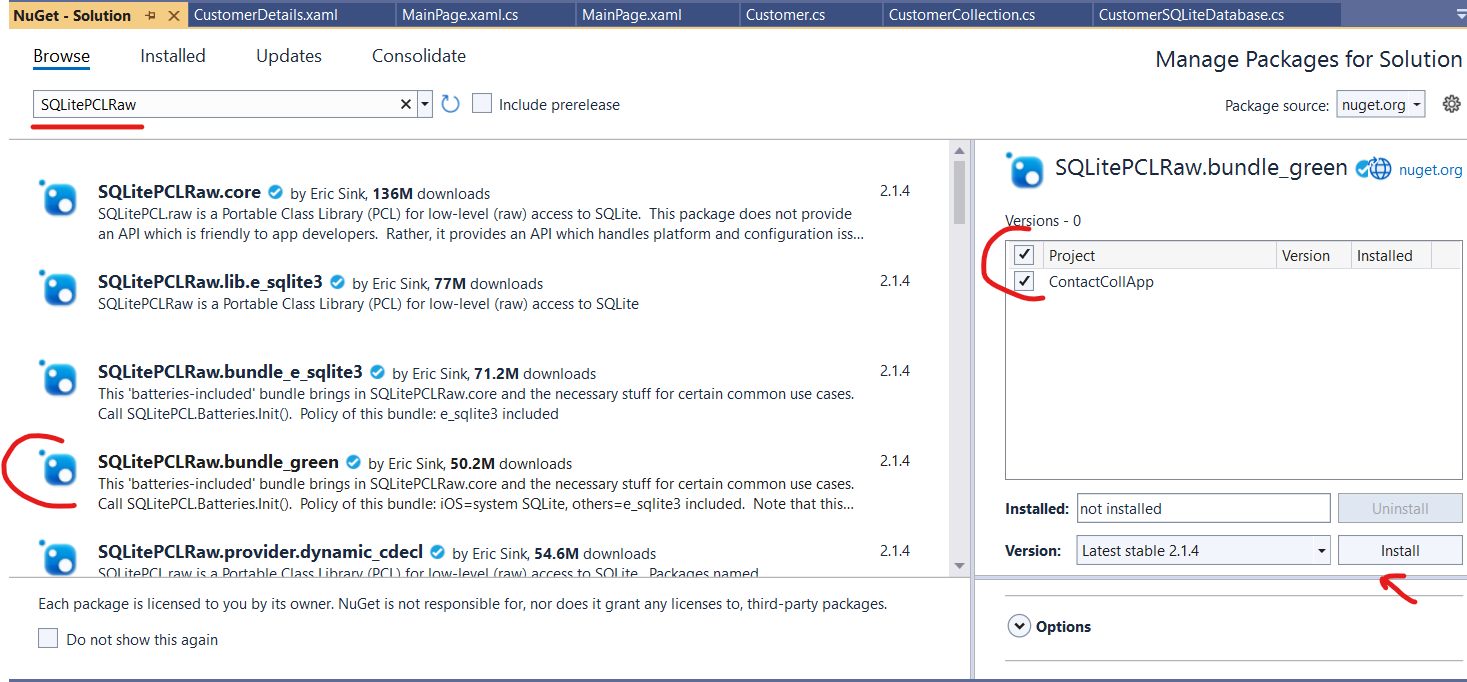
* Install SQLite: To use SQLite. First we need to install 3rd party C# Api through NuGet



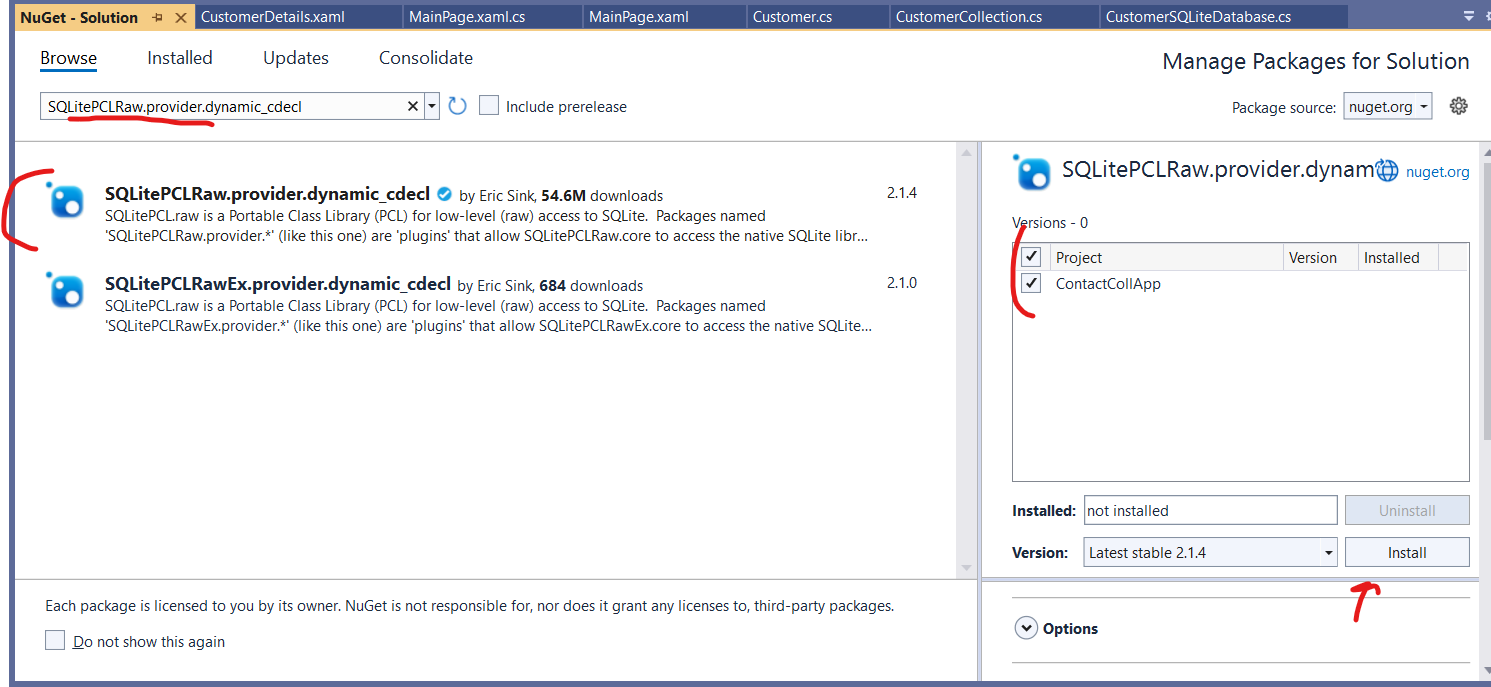
* Install sqlite-net-pcl from NuGet Package Manager: Search sqlite-net-pcl



* Search SQLitePCLRaw.bundle\_green



* Search SQLitePCLRaw.provider.dynamic\_cdecl



* Start making the application
* Step1: Create a class to represent your contact data

using SQLite;

public class Contact

{

[PrimaryKey, AutoIncrement]//notify that this record is primary key and auto increase

public int Id { get; set; }

public string Name { get; set; }

public string PhoneNumber { get; set; }

public string Address { get; set; }

public string Gender { get; set; }

[Ignore] // Đánh dấu trường này để không lưu trong cơ sở dữ liệu

public string ImageSource { get; set; }

}

Remember to put using SQlite to your class;

* Step2:Create class DatabaseService

public class DatabaseService

{

SQLiteConnection database; // variable to store the connection to database

public DatabaseService(string dbPath)

{

database = new SQLiteConnection(dbPath); // initialize the database

//Create table

database.CreateTable<Contact>();

if (database.Table<Contact>().Count() == 0)

{

// Add default data

database.Insert(new Contact { Name = "John Doe", PhoneNumber = "123-456-7890", Address = "123 Main St", Gender= "Male" });

database.Insert(new Contact { Name = "Jane Smith", PhoneNumber = "987-654-3210", Address = "456 Elm St", Gender = "Female" });

}

}

public List<Contact> GetContacts()

{

return database.Table<Contact>().ToList();

}

public void AddContact(Contact contact)

{

database.Insert(contact);

}

}

* Step3: Design the main page **MainPage.xaml** to show the list of contact

<?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"

x:Class="SQLiteContact2.MainPage">

<StackLayout>

<ListView x:Name="contactListView" RowHeight="100">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<StackLayout Padding="10" Orientation="Horizontal">

<Image Source="{Binding ImageSource}" WidthRequest="50" HeightRequest="50" />

<StackLayout >

<Label Text="{Binding Name}" FontSize="20" />

<Label Text="{Binding Gender}" />

<Label Text="{Binding PhoneNumber}" />

<Label Text="{Binding Address}" />

</StackLayout>

</StackLayout>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

<Button Text="Thêm liên hệ" Clicked="OnAddContactClicked" />

</StackLayout>

</ContentPage>

Then add some code to the code behide of the page **MainPage.xaml.cs**

public partial class MainPage : ContentPage

{

DatabaseService databaseService;

public MainPage()

{

InitializeComponent();

// Store database in the application folder

string dbPath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), "contacts.db");

// Initialize the DatabaseService

databaseService = new DatabaseService(dbPath);

}

protected override void OnAppearing()

{

base.OnAppearing();

// Get the list of contact from database

List<Contact> contacts = databaseService.GetContacts();

foreach (var c in contacts)

{

if (c.Gender== "Male")

{

c.ImageSource = "male.png";

}

else

{

c.ImageSource = "female.png";

}

}

// Pass the list of contact to the lisview

contactListView.ItemsSource = contacts;

}

// When user click Addcontact then navigagte to the AddEditContactPage

private void OnAddContactClicked(object sender, EventArgs e)

{

Navigation.PushAsync(new AddEditContact(databaseService));

}

}

* Step4: Design the **AddEditContact.xaml**

<?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"

x:Class="SQLiteContact2.AddEditContact"

Title="AddEditContact">

<StackLayout>

<Entry Placeholder="Tên" x:Name="nameEntry" />

<Entry Placeholder="Số điện thoại" x:Name="phoneEntry" />

<Entry Placeholder="Địa chỉ" x:Name="addressEntry" />

<Button Text="Lưu" Clicked="OnSaveClicked" />

</StackLayout>

</ContentPage>

Then add some code to the code behide of the page **AddEditContact.xaml.cs**

namespace SQLiteContact2;

public partial class AddEditContact : ContentPage

{

DatabaseService databaseService;

public AddEditContact(DatabaseService dbService)

{

InitializeComponent();

databaseService = dbService;

}

private void OnSaveClicked(object sender, EventArgs e)

{

Contact newContact = new Contact

{

Name = nameEntry.Text,

PhoneNumber = phoneEntry.Text,

Address = addressEntry.Text

};

databaseService.AddContact(newContact);

Navigation.PopAsync();

}

}

* Table of database method

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| SQLiteConnection(string databasePath) | Initializes a connection to the SQLite database, providing the database path. If the database doesn't exist, it will be automatically created.  *SQLiteConnection connection = new SQLiteConnection("myDatabase.db");* |
| Close() | Closes the connection to the SQLite database.  *connection.Close();* |
| CreateTable<T>() | Creates a table in the database based on the type T if the table is not existed.  *connection.CreateTable<MyTable>();* |
| DropTable<T>() | Drops a table from the database based on type T.  *connection.DropTable<MyTable>();* |
| Insert(item) | Inserts an item into the database.  *connection.Insert(new MyItem());* |
| Update(item) | Updates an item in the database.  *connection.Update(existingItem);* |
| Delete(item) | Deletes an item from the database.  *connection.Delete(existingItem);* |
| Table<T>() | Retrieves all data from a table of type T and returns a LINQ query.  *var query = connection.Table<MyTable>();* |
| Execute(string query) | Executes a custom SQL query on the database.  *connection.Execute("DELETE FROM MyTable WHERE Id = ?", id);* |
| Query<T>(string query, params object[] args) | Executes a custom SQL query and returns the result as a list of objects of type T.  *var results = connection.Query<MyTable>("SELECT \* FROM MyTable WHERE Name = ?", name);* |
| ExecuteScalar<TResult>(string query, params object[] args) | Executes a custom SQL query and returns a result value.  *int count = connection.ExecuteScalar<int>("SELECT COUNT(\*) FROM MyTable");* |
| ExecuteScalar<T>(string query, params object[] args) | Executes a custom SQL query and returns the result as an object of type T.  *string name = connection.ExecuteScalar<string>("SELECT Name FROM MyTable WHERE Id = ?", id);* |
| BeginTransaction() | Starts an SQLite transaction.  *connection.BeginTransaction();* |
| Commit() | Completes and commits an SQLite transaction.  *connection.Commit();* |
| Rollback() | Rolls back an SQLite transaction.  *connection.Rollback();* |
|  |  |

1. ss