Project work 2

The task I chose : *As an UNDAC Team Member, I want to raise an alert to the operational team leader so that my input into an operation is visible*

Descriptive Summary of the Issue

The primary focus of this portfolio entry is to enhance an application used by UNDAC (United Nations Disaster Assessment and Coordination) team members. The goal is to enable team members to raise alerts, making it easier for them to communicate with the operational team leader. This enhancement involves the addition of a new feature, including the creation of an "Alert" model in the SQLite database, as well as a method for triggering these alerts. This modification aims to strengthen the operational efficiency of UNDAC team members in emergency situations.

Snippets from My Code with Commentary

I firstly created a class `Alert.cs` in which we find different elements: the alert ID, the message, the sender, the recipient, the time of sending.

```
public class Alert
{
    [PrimaryKey, AutoIncrement]
    public int Id { get; set; } // Identifiant unique de l'alerte
    public string Message { get; set; } // Contenu du message d'alerte
    public DateTime Timestamp { get; set; } // Horodatage de la cr�ation de
l'alerte
    public string Sender { get; set; } // L'exp�diteur de l'alerte
    public string Recipient { get; set; } // Le destinataire pr�vu de l'alerte
}
```

I, of course, initialized the creation of the table in the database with this line of code: `await Database.CreateTableAsync<Alert>();`.

I add it in the `Init()` function

```
public async Task Init()
        {

            if (Database is not null)
                return;

            Database = new SQLiteAsyncConnection(DatabasePath, Constants.Flags);
            // add your table here for example await
Database.CreateTableAsync<Organisation>();
            await Database.CreateTableAsync<Alert>();

        }
```

We then create and implement the page for sending messages. I add the xaml page `RaiseAlert.xaml` and its code behind `RaiseAlert.xaml.cs`.

RaiseAlert.xaml :

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Undac.RaiseAlert"
             Title="AlertPage">


    <StackLayout>

        <Entry x:Name="MessageEntry" Placeholder="Enter the alert message" />

        <Entry x:Name="SenderEntry" Placeholder="Your name" />

        <Entry x:Name="RecipientEntry" Placeholder="Recipient's name" />

        <Button Text="Trigger the alert" Clicked="RaiseAlertClicked" />


    </StackLayout>
</ContentPage>
```

- I use the `<Entry>` tag that allows users to interact with the application to provide information. But it's also more convenient for performing checks, which is the case for us.

- I have also created a `<button>` to send the alert to the relevant person.

RaiseAlert.xaml.cs :

```
using Undac.Data;

namespace Undac;

public partial class RaiseAlert : ContentPage
{
    public RaiseAlert()
    {
        InitializeComponent();
    }

    private async Task RaiseAlertClickedAsync(object a, EventArgs e)
    {
        string message = MessageEntry.Text;
        string sender = SenderEntry.Text;
        string recipient = RecipientEntry.Text;


        await UndacDatabase.RaiseAlert(message, sender, recipient);
    }

}
```

1. The asynchronous method RaiseAlertClickedAsync(object a, EventArgs e) is called when the user clicks on the ***'trigger the alert'*** button to trigger an alert. This method retrieves the content entered in the text fields of the user interface.

   Specifically:

   - `string message = MessageEntry.Text`; retrieves the text entered in the '*MessageEntry*' text field.
   - `string sender = SenderEntry.Text`; retrieves the text entered in the '*SenderEntry*' text field.
   - `string recipient = RecipientEntry.Text`; retrieves the text entered in the '*RecipientEntry*' text field.

2. After retrieving this information, the `await UndacDatabase.RaiseAlert(message, sender, recipient)` method is called to trigger an alert. This line of code utilizes the UndacDatabase class with the static method `RaiseAlert`, which I've added, to handle sending the alert. The message, sender, and recipient values are passed to this method to success the operation.

The `RaiseAlert()` method :

```
    internal static async Task RaiseAlert(string message, string sender, string 
recipient)
        {
            var alert = new Alert
            {
                Message = message,
                Sender = sender,
                Recipient = recipient,
                Timestamp = DateTime.Now
            };

            await Database.InsertAsync(alert);
        }
```

The `RaiseAlert()` method is used to record an alert in a database. It creates an instance of the `Alert` class with the alert's information (message, sender, recipient, timestamp) and inserts it into the database asynchronously using `InsertAsync`.

However, the last part I did doesn't work because I still can't connect to the database.

Conclusion

Comparing my practice to that of my colleagues, I noticed that code documentation and comment writing were areas where I could improve. Clear documentation is essential for ensuring that all team members understand the code and can contribute to it. I also realized that collaboration and communication are skills just as important as programming itself.