FED MAUI Lab 02 MauiTodo App

Formål

To gain the experience with data binding.

*Create MauiTodo App to practice and demonstrate an understanding of how data binding works in .NET MAUI. (*A simple .NET MauiTodo App for creating a list of tasks.)

Forudsætninger

You have read chapter 3 of MAUI in Action.

Delopgave 1: Todo List

Create a new .NET Maui project called *MauiTodo* in Visual Studio. Use a package SQLite-net. *You need to* **install** *sqlite-net-pcl and sqlite-net-cipher NuGet packages in Visual Studio.*

Add a folder called *Models* and you need a *TodoItem* class with properties that represents the to-do items. (*TodoItem* class and Database class are implemented in files *TodoItem.cs, Database.cs* and Download from Brightspace)

In the XAML (MainPage.xaml) file

Delete the pre-made *ScrollView* and everything inside it.

- This should leave only with the <ContentPage...> </ContentPage> tags.

Inside <ContentPage...> </ContentPage> tags:

Replace with a New Layout, *Grid* and child elements (*Label, Entry, Date picker, Button, ScrollView*):

1. *Add* an opening and closing Grid tag with *five* rows.

```
<Grid RowDefinitions="1*, 1*, 1*, 1*, 8*"
      MaximumWidthRequest="400"
      Padding="20">

</Grid>
```

Note: Label, Entry, Date picker, Button, ScrollView layout will be here between Grid opening and close tags.

Inside Grid:

2. *Add* a Label with the page title as "Maui Todo" : Use a `Label` with Text property

```
<Label ...
       Text="Maui Todo"
       .../>
```

3. *Add* a text Entry where the user can enter the title for new to-do items and *Give* a name, *TodoTitleEntry*

```
<Entry ...
       ...
       x:Name="TodoTitleEntry" />
```

4. *Add* a Date picker for the due date and Give a name, DueDatePicker.

```
<DatePicker ...
            ...
            x:Name="DueDatepicker" />
```

5. *Add* a Button to confirm adding new to-do items, and Clicked event with event handler name e.g. *Button_Clicked*.

```
<Button ..
        Text="Add"
            ...
        Clicked="Button_Clicked" />
```

6. *Add* a ScrollView on the fifth row (**row 4**) to extend/scroll with to-do items when it extend beyond the page. Note: you need a Label with a name *TodoLabel* inside <ScrollView ..> ... </ScrollView>.

```
<ScrollView Grid.Row="4">
    <Label ...
           x:Name="TodosLabel" />
</ScrollView>
```

In Code-behind (MainPage.xaml.cs file)

Delete the method `OnCounterClicked` (Replace with the new logics.)

Add code to the code-behind to wire up functionality (Refer the lecture slides):

1. **Add** an instance of the database, and the *values of to-dos* in the database to hold a user's new to-do item.

```
string _todoListData = string.Empty; //Values of the to-do items
readonly Database _database; //Stores an instance of the database class
```

2. Update the class *constructor*, ***Inside the Constructor***,
   Create an instance of the database class and Assign it to the database field.
   *(**Important**: **Do NOT delete*** `InitializeComponent();`*)*:

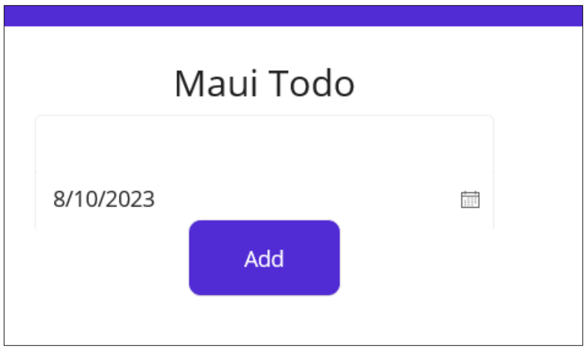
```
_database = new Database(); //create an instance of the database class & assign it to the _database field.
_ = Initialize(); //Uses the discard variable to call our Initialize method
```

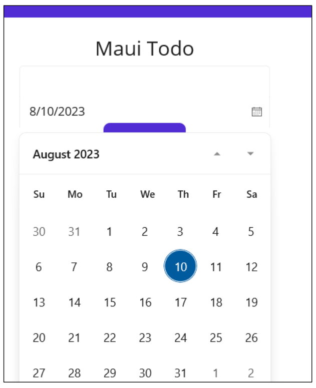
3. **Add** a new Task method to `Initialize` our page on load (Use Async/Await): *Refer to the lecture slide.*

4. **Add** an event handler method (`Button_Clicked` event handler) to respond to button clicks and add a new to-do to the database (use Async/Await): *Refer to the lecture slide.*

Run the App, the app can:

When you click the Add button, an event handler gets the values from the Entry and DatePicker and uses them to create a new to-do item with those values for the title and due date, respectively. It then adds them to the list of to-do items on the screen.





Delopgave 2: Bindings

Apply Data binding in the MauiTodo App.

In the XAML (MainPage.xaml) file

1. Replace *ScrollView* with this *CollectionView*. Use *CollectionView, ItemTemplate, DataTemplate* to apply Data binding in XAML. (Refer the lecture slides.)
   - Name the *CollectionView* with e.g. TodosCollection so we can refer to it in code.

     ```
     <CollectionView ...
         x:Name="TodosCollection">
     ```

   - Inside the *CollectionView*, Add an element to define the ItemTemplate property of the *CollectionView*.

     ```
     <CollectionView.ItemTemplate>
     ```

   - Use a DataTemplate to defines how each item in the collection to be presented and is assigned to the ItemTemplate property of the CoolectionView by being nested as a direct child.

     ```
     <DataTemplate>
     ```

2. For extra layouts, you can add a Grid, a Checkbox, two Labels:

   Grid can have:

   - Two rows: one with a height to adjust its contents, and the other with for a height of e.g. 50.
   - Two columns: one 2/7 of the width, and the other 5/7 of the width.

   Checkbox cto the first column, first row for the item layout.

   Two Labels:

   - Add one Label to the second column, first row and *Bind the* Text *property of the Label to the Title property of the item*.
   - Add the other Label to the second column, second row and *Bind the* Text *property of the Label to the* Due *property of the item* and as the Due property is a DateTime, supplies a formatting rule.

In the code-behind

1. *Add* an ObservableCollection of TodoItems at the very top of the class (before the private member definitions).

   Note: Remember add the required using statement at the top of the file.

2. In the constructor, *set* the ItemsSource property of the CollectionView to the ObservableCollection:
3. *Delete* the private *_todoListData* string and every line that references it to get rid of the code that are not using anymore.
4. *Add* a new logic *in two places*: to handle initializing the collection and updating it when a user adds a new to-do item.
   - Add the to-do item in the loop to the ObservableCollection in the foreach loop in the Initialiase method.
   - Add the same logic in the Button_Clicked method in if statement.

     ```
     Todos.Add(todo);
     ```

Delopgave 3: Bindings Refactoring with ItemsSource in XAML

Refactor the MauiTodo app to do all the binding in the XAML to look at how to set the *ItemsSource* for our CollectionView in XAML.

1. Open the *MainPage.xaml.cs* code-behind file, and in the constructor, **remove** the line we added in the previous section to set the items source:

   ~~`TodosCollection.ItemsSource = Todos;`~~

2. Instead, **set up** this binding in the XAML. Open the MainPage.xaml file:
3. In the <ContentPage...> opening tag,
   Add a name (so that it can be referenced) and a binding context.
4. Then, in the CollectionView,
   Bind the ItemsSource property to the *Todos ObservableCollection* in the binding context.

Note: Run the App, the output result will be same, but the code is refactored effectively.