# ICE Motivator-MVVM

## Part 1 – Data Binding

We are going to expand the Motivator application to use an MVVM pattern instead of having all of the code and the data hard coded in the cs code behind file.

1. Copy the Motivator project and rename it to Motivator-MVVM. By this, I mean to copy the project folder into a new one named Motivator-MVVM. Renaming projects inside of visual Studio is not an easy thing. It is ok to have them both named Motivator. You might want to go into the Android/iOS settings and change the display name to Motivator-MVVM. Change things like the default namespace could cause the project to not work well, so please avoid this.
2. On the motivator project, right click and add a new class called MainPageViewModel. This class will be used to be out ViewModel for data binding to the main page.
3. Open up the view model file and add 3 properties to the class. Create this with backing variables (propfull + tab+tab) so that we can add some extra code. Add properties for Title, SayingText, and CategoryText. Each property could look like the following code examples (one of the other)

private string title;

public string Title

{

get { return title; }

set { title = value; }

}

private string title;

public string Title

{

get => title;

set => title = value;

}

Complete this for the 3 properties.

|  |  |
| --- | --- |
| Property | Default Value |
| Title | Motivator-MVVM |
| SayingText | Tap a button to get motivated! |
| CategoryText | Eat Vegetables! |

1. Create a constructor for the MainPageViewModel class (ctor+tab+tab) and initialize your properties.
2. In the MainPage.cs file we will be adding a reference to the view model right after the InitializeComponent call. Set the Binding Context to an instance of the new view model class.  
     
    BindingContext = new MainPageViewModel();
3. At this point the view model class is setup as the BindingContext so we can start adding bindings in the XAML code. In the MainPage.xaml file change the various text values to bind to the property from the view model. Here is an example on how to set the title in the XAML. Make the same changes for the sayingText as well as the text on the button.

Text="{Binding Title}"

Run the program and you should look the exact same as it did before we added a view model class.

## Part 2 – Command Binding

At this point we have our data coming back from the view model but we still have the button clicking being handled in the code behind. We are going to create a command and use that for the button instead.

1. Go to the view model class and create a new property named MotivateCommand using the type ICommand (prop+tab+tab). This can be an auto property and does NOT need to have a backing variable, but can if that is what you are most comfortable with.
2. In the constructor, lets initialize the command and we will use a lambda function like this:  
     
    MotivateCommand = new Command(() =>

{

SayingText = "Eat Veggies and you will lose weight!";

});

Notice that we are setting the SayingText property and not the .Text on the control.

1. Back in the MainPage.XAML, remove the setting for the Clicked handler and add a Command setting and data bind it to the command in the view model.  
     
   Command="{Binding MotivateCommand}"

Notice that when you run at this point that the button will not seem to do anything. Add a breakpoint in your command and see that yes it is getting there when you press the button but also that you are setting the saying correctly.

1. Go back to the view model class and then add the INotifyPropertyChanged interface to the class and implement the class using right click on the interface name.
2. What needs to be done is to call the PropertyChanged event to let the GUI know that it needs to update. This will be handled in the setter for the SayingText property. This is done in the setter since then anytime it gets set, it will trigger the GUI to update. The setter will look more like this:

set

{

sayingText = value;

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("SayingText"));

}  
  
The ?.Invoke checks that the variable on the left side is not null before calling it. This could have been done with an if statement as well.

At this point running the app should work like it did before.

1. Make the same properties, data bindings, and commands for the second button that you added to the original project.  
     
   When you are complete the only difference you will see is the title is different.

## Part 3 – Expanding Motivations

From here, the app works like it did before. We are going to enhance it to have each button pick from a list of sayings instead of just a single one.

1. In the view model class, add a private member variable for each category that you have in the app. These should be of type List<string>. In the constructor setup a number of sayings for each such as this example:

healthySayings = new List<string>

{

"Eat Veggies and you will lose weight!",

"Get some exercise",

"Turn on your bluelight filtering on the pc"

};

1. In the MotivateCommand and the other Command that you added, randomly pick one of the sayings to set the SayingText variable instead of a hard coded value.

**Submission: ZIP and Post to the dropbox before the due date**