Chapter 15: Gestures

Section 15.1: Make an Image tappable by adding a TapGestureRecognizer

There are a couple of default recognizers available in Xamarin.Forms, one of them is the `TapGestureRecognizer`.

You can add them to virtually any visual element. Have a look at a simple implementation which binds to an `Image`. Here is how to do it in code.

```
var tappedCommand = new Command(() =>
{
    //handle the tap
});

var tapGestureRecognizer = new TapGestureRecognizer { Command = tappedCommand };
image.GestureRecognizers.Add(tapGestureRecognizer);
```

Or in XAML:

```
<Image Source="tapped.jpg">
    <Image.GestureRecognizers>
        <TapGestureRecognizer
               Command="{Binding TappedCommand}"
               NumberOfTapsRequired="2" />
  </Image.GestureRecognizers>
</Image>
```

Here the command is set by using data binding. As you can see you can also set the `NumberOfTapsRequired` to enable it for more taps before it takes action. The default value is 1 tap.

Other gestures are Pinch and Pan.

Section 15.2: Gesture Event

When we put the control of Label, the Label does not provide any event. <Label x:Name="lblSignUp Text="Don't have account?"/> as shown the Label only display purpose only.

When the user want to replace Button with Label, then we give the event for Label. As shown below:

XAML

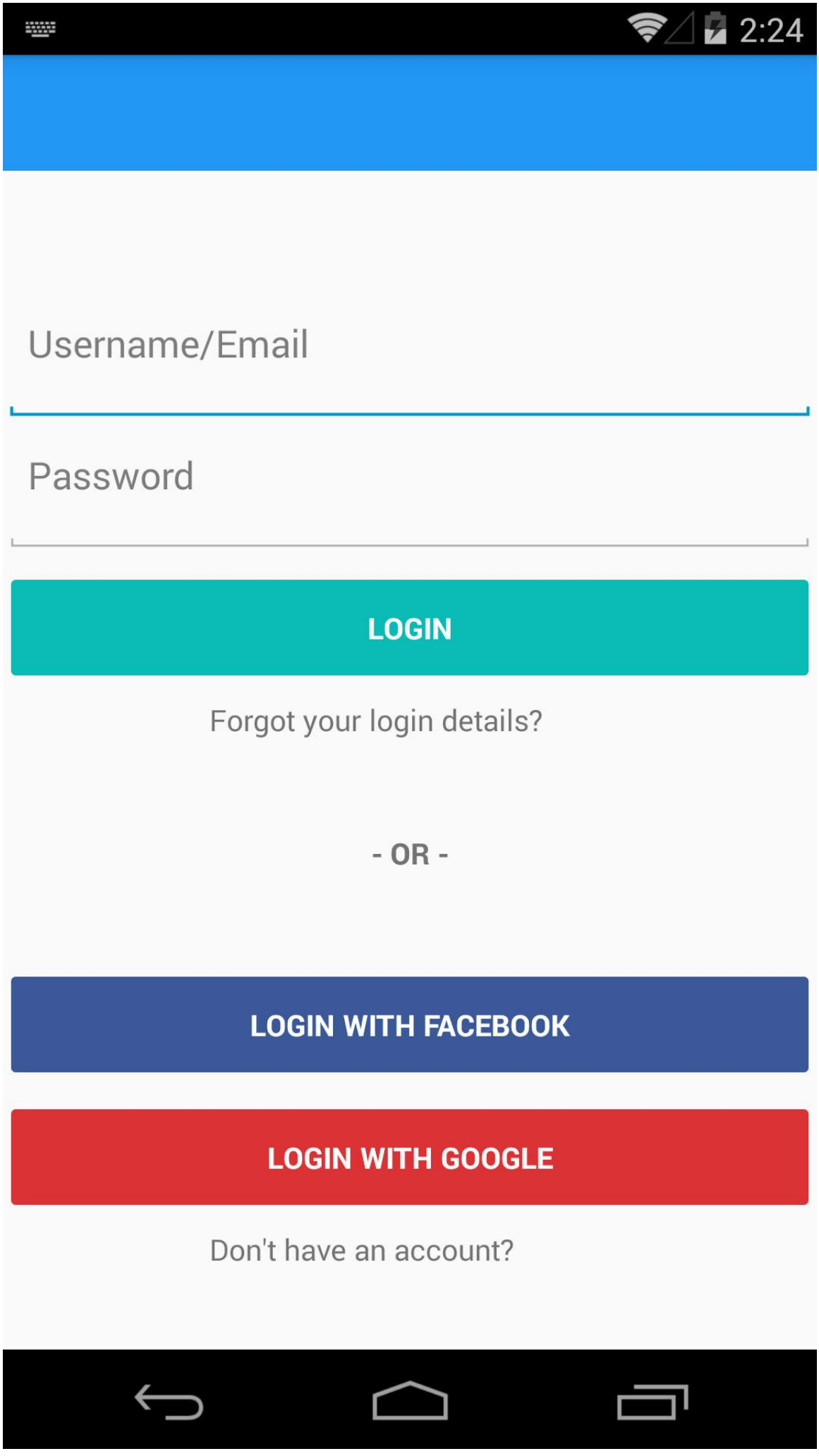
```
<Label x:Name="lblSignUp" Text="Don't have an account?" Grid.Row="8" Grid.Column="1"
Grid.ColumnSpan="2">
  <Label.GestureRecognizers>
    <TapGestureRecognizer
  Tapped="lblSignUp_Tapped"/>
  </Label.GestureRecognizers>
```

C#

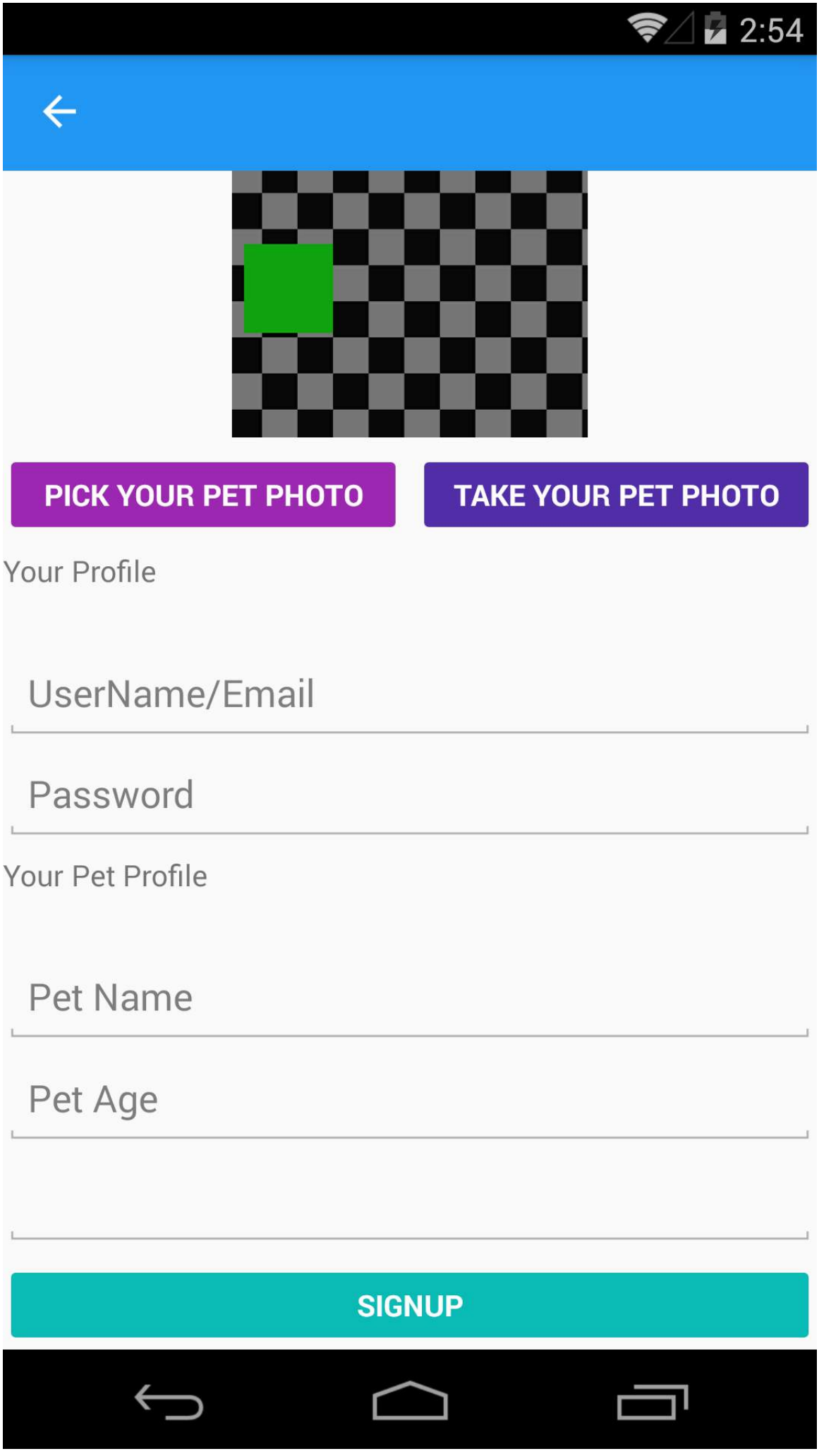
```
var lblSignUp_Tapped = new TapGestureRecognizer();
lblSignUp_Tapped.Tapped += (s,e) =>
{
//
//  Do your work here.
```

```
//
};
lblSignUp.GestureRecognizers.Add(lblSignUp_Tapped);
```

The Screen Below shown the Label Event. Screen 1 : The Label "Don't have an account?" as shown in Bottom .


2:24
Username/Email
Password
LOGIN
Forgot your login details?
- OR -
LOGIN WITH FACEBOOK
LOGIN WITH GOOGLE
Don't have an account?

When the User click the Label "Don't have an account?", it will Navigate to Sign Up Screen.



For more details:

[https://developer.xamarin.com/guides/xamarin-forms/user-interface/gestures/tap/][1]

Section 15.3: Zoom an Image with the Pinch gesture

In order to make an `Image` (or any other visual element) zoomable we have to add a `PinchGestureRecognizer` to it. Here is how to do it in code:

```
var pinchGesture = new PinchGestureRecognizer();
pinchGesture.PinchUpdated += (s, e) => {
// Handle the pinch
};

image.GestureRecognizers.Add(pinchGesture);
```

But it can also be done from XAML:

```
<Image Source="waterfront.jpg">
  <Image.GestureRecognizers>
    <PinchGestureRecognizer PinchUpdated="OnPinchUpdated" />
  </Image.GestureRecognizers>
</Image>
```

In the accompanied event handler you should provide the code to zoom your image. Of course other uses can be implement as well.

```
void OnPinchUpdated (object sender, PinchGestureUpdatedEventArgs e)
{
    // ... code here
}
```

Other gestures are Tap and Pan.

Section 15.4: Show all of the zoomed Image content with the PanGestureRecognizer

When you have a zoomed `Image` (or other content) you may want to drag around the `Image` to show all of its content in the zoomed in state.

This can be achieved by implementing the PanGestureRecognizer. From code this looks like so:

```
var panGesture = new PanGestureRecognizer();
panGesture.PanUpdated += (s, e) => {
  // Handle the pan
};

image.GestureRecognizers.Add(panGesture);
```

This can also be done from XAML:

```
<Image Source="MonoMonkey.jpg">
  <Image.GestureRecognizers>
    <PanGestureRecognizer PanUpdated="OnPanUpdated" />
  </Image.GestureRecognizers>
</Image>
```

In the code-behind event you can now handle the panning accordingly. Use this method signature to handle it:

```
void OnPanUpdated (object sender, PanUpdatedEventArgs e)
{
  // Handle the pan
}
```

Section 15.5: Tap Gesture

With the Tap Gesture, you can make any UI-Element clickable (Images, Buttons, StackLayouts, ...):

(1) In code, using the event:

```
var tapGestureRecognizer = new TapGestureRecognizer();
tapGestureRecognizer.Tapped += (s, e) => {
    // handle the tap
};
image.GestureRecognizers.Add(tapGestureRecognizer);
```

(2) In code, using `ICommand` (with MVVM-Pattern, for example):

```
var tapGestureRecognizer = new TapGestureRecognizer();
tapGestureRecognizer.SetBinding (TapGestureRecognizer.CommandProperty, "TapCommand");
image.GestureRecognizers.Add(tapGestureRecognizer);
```

(3) Or in Xaml (with event and `ICommand`, only one is needed):

```
<Image Source="tapped.jpg">
    <Image.GestureRecognizers>
        <TapGestureRecognizer Tapped="OnTapGestureRecognizerTapped" Command="{Binding TapCommand"}
/>
  </Image.GestureRecognizers>
</Image>
```

Section 15.6: Place a pin where the user touched the screen with MR.Gestures

Xamarins built in gesture recognizers provide only very basic touch handling. E.g. there is no way to get the position of a touching finger. MR.Gestures is a component which adds 14 different touch handling events. The position of the touching fingers is part of the `EventArgs` passed to all MR.Gestures events.

If you want to place a pin anywhere on the screen, the easiest way is to use an `MR.Gestures.AbsoluteLayout` which handles the `Tapping` event.

```
<mr:AbsoluteLayout x:Name="MainLayout" Tapping="OnTapping">
    ...
</mr:AbsoluteLayout>
```

As you can see the `Tapping="OnTapping"` also feels more like .NET than Xamarins syntax with the nested `GestureRecognizers`. That syntax was copied from iOS and it smells a bit for .NET developers.

In your code behind you could add the `OnTapping` handler like this:

```
private void OnTapping(object sender, MR.Gestures.TapEventArgs e)
{
    if (e.Touches?.Length > 0)
    {
        Point touch = e.Touches[0];
```

```
        var image = new Image() { Source = "pin" };
        MainLayout.Children.Add(image, touch);
    }
}
```

Instead of the `Tapping` event, you could also use the `TappingCommand` and bind to your ViewModel, but that would complicate things in this simple example.

More samples for MR.Gestures can be found in the GestureSample app on GitHub and on the MR.Gestures website. These also show how to use all the other touch events with event handlers, commands, MVVM, ...