# Chương 1

**Xây dựng ứng dụng bằng Xamarin**

Xamarin đã trở thành đa nền tảng đến mức bây giờ bao gồm ĨOS, Android, Windows, macos, Tizen, WPF, Hololens, GTK và các loại khác. Đó là rất nhiều nền tảng. Xamarin làm điều đó như thế nào?

Xuất thân từ Dự án Mono mã nguồn mở đã mang .NET lên Linux, nền tảng Xa marin là một cổng của .NET cho các hệ điều hành IOS và Android.

Dưới Xamarin.Android là Mono cho Android và bên dưới Xamarin.iOS là MonoTouch. Đây là các ràng buộc c # với API Android và ios gốc để phát triển trên thiết bị di động và máy tính bảng . Điều này mang đến cho chúng ta sức mạnh của giao diện người dùng Android và IOS, thông báo, đồ họa, hoạt hình và các tính năng của điện thoại như vị trí và máy ảnh, tất cả đều sử dụng c # và XAML. Mỗi bản phát hành mới của hệ điều hành Android và ĨOS được khớp với bản phát hành Xamarin mới bao gồm các ràng buộc với API mới của chứng. Xamarin.Forms là một lớp nằm trên các liên kết UI khác, cung cấp thư viện UI đa nền tảng.

Chương này cung cấp một bản giới thiệu về hai cách để xây dựng một ứng dụng bằng Xamarin:

• Xamarin. Forms là một thư viện UI đa nền tảng cho Android, ios và nhiều thứ khác.

• Cách tiếp cận UI dành riêng cho nền tảng (hoặc bản địa) sử dụng Xamarin.Android, Xamarin.iOS.

Chúng ta sẽ nói về khi nào Xamarin.Forms hữu ích và khi một cách tiếp cận cụ thể hơn về nền tảng có thể là better. Sau đó, chúng tôi sẽ nghiên cứu xây dựng giao diện người dùng Xamarin.Forms bằng cách sử dụng các trang, bố cục và chế độ xem. Chứng tôi sẽ tạo ra một giải pháp Xamarin.Forms chứa các dự án được chia sẻ và các dự án cụ thể cho nền tảng. Trong khi thêm các điều khiển Xamarin.Forms vào một dự án, chứng ta sẽ chạm vào các khái niệm UI cơ bản như xử lý hình ảnh và điều khiển định dạng trong bố cục. Hãy bắt đầu bằng cách thảo luận về Xamarin.Forms

**Hiểu về Xamarin.Forms**

Xamarin.Forms là bộ công cụ gồm các lớp UI đa nền tảng được xây dựng trên đỉnh các lớp UI cụ thể nền tảng hơn: Xamarin.Android và Xamarin.iOS. Xamarin.Android và Xamarin.iOS cung cấp các lớp được ánh xạ tới SDK UI gốc tương ứng của họ : SDK UIKit và Android Android. Xamarin.Forms cũng liên kết trực tiếp nhiều nền tảng khác. Điều này cung cấp một tập họp các thành phần UI đa nền tảng hiển thị trong mỗi ba hệ điều hành gốc này



**XAML so với c #**

Xamarin.Forms cung cấp bộ công cụ đa nền tảng gồm các trang, bố cục và điều khiển và là nơi tuyệt vời để bắt đầu xây dựng ứng dụng một cách nhanh chóng. Có hai cách để tạo người dùng trong các giao diện trong Xamarin.Forms, trong c # bằng cách sử dụng API Xamarin.Forms phong phú hoặc sử dụng Ngôn ngữ đánh dấu mở rộng (XAML), ngôn ngữ đánh dấu khai báo do Microsoft sử dụng để xác định giao diện người dùng. Cuốn sách trước của tôi, Xamarin Mobile Application Development, đã trình bày về phương pháp c #, nhưng cuốn sách này là tất cả về XAML. Bạn có thể tạo chính xác cùng loại UI ừong cả c # và XAML, vì vậy lựa chọn chủ yếu là chủ quan và cá nhân, mặc dù có những cân nhắc về kiến trúc. XAML buộc tách mã Vie w, trong khi cách tiếp cận c # thì không.

Jason Smith, kỹ sư phần mềm chính trong nhóm Xamarin.Forms tại Microsoft, đã giải thích theo cách này, chứng tôi xây dựng Xamarin. Hình thức mã đầu tiên. Điều đó có nghĩa là tất cả các tính năng được tạo trước tiên để hoạt động bằng c #, sau đó chúng sẽ triển khai chứng cho XAML.

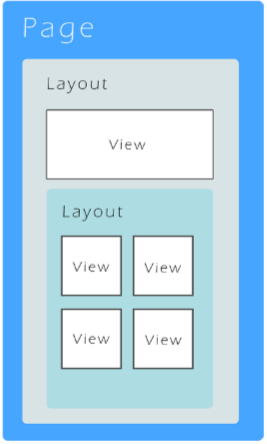
Các phần tử Xamarin.Forms được xây dựng bằng các lớp Trang , Bố cục và Xem . API này cung cấp một loạt các mẫu UI di động đa nền tảng tích hợp. Bắt đầu với các đối tượng Trang cấp cao nhất, nó cung cấp các trang menu quen thuộc như NavigationPage cho các menu khoan phân cấp, TabbedPage cho các menu tab, Mas terDetailPage để tạo ngăn kéo điều hướng, CarouselPage để cuộn các hang hình ảnh và ContentPage, một lớp cơ sở cho tạo các trang tùy chỉnh. Các bố cục trải rộng các định dạng tiêu chuẩn mà chúng tôi sử dụng trên các nền tảng khác nhau bao gồm StackLayout, AbsoluteLayout, RelativeL ayout, Grid, ScrollView và ContentView, lớp bố cục cơ sở. Được sử dụng trong các bố cục đó là hàng tá các điều khiển hoặc chế độ xem quen thuộc, chẳng hạn như ListView , Nút, DatePicker và TableView . Nhiều trong số các khung nhìn này có các tùy chọn ràng buộc dữ liệu tích hợp.

**Mẹo** đồng nghĩa của Variou cho màn hình ui di động tồn tại, chẳng hạn như lượt xem và trang và chúng được sử dụng thay thế cho nhau, một khung nhìn có thể có nghĩa là một màn hình nhưng cũng có thể đề cập đến một điều khiển trong các bối cảnh nhất định.

Xamarin.Forms bao gồm các lớp độc lập với nền tảng được liên kết với các đối tác cụ thể nền tảng riêng của chúng. Điều này có nghĩa là chứng tôi có thể phát triển UI gốc, cơ bản cho cả ba nền tảng mà hầu như không có kiến thức về UI và ĨOS. Hãy vui mừng nhưng hãy cẩn thận! Pur ists cảnh báo rằng cố gắng xây dựng ứng dụng cho các nền tảng này mà không hiểu về API gốc là một cam kết liều lĩnh. Hãy chú ý đến tinh thần của họ. Chúng ta phải quan tâm đến nền tảng Android và ĨOS, sự phát triển, tính năng, id iosyncrasies và bản phát hành của chúng. Chúng ta cũng có thể đắm mình trong sự tiện lợi và thiên tài của sự trừu tượng đa nền tảng tuyệt vời đó là Xamarin.Forms!

**Sử dụng giao diện người dùng Xamarin.Forms**

Các trang, bố cục và khung nhìn tạo nên cốt lõi của Xamarin.Forms UI .Các trang là vùng chứa chính và mỗi màn hình được điền bởi một lóp Trang duy nhất. Một trang có thể chứa các biến thể của lớp Bố cục , sau đó có thể giữ các bố cục khác, được sử dụng để đặt và định cỡ nội dung của chúng. Mục đích của các trang và bố cục là để chứa và trình bày các khung nhìn, đó là các điều khiển được kế thừa từ Chế độ xem lớp .



**Page**

Lớp Trang là thùng chứa chính của mỗi màn hình chính trong ứng dụng

Page là lớp cơ sở để tạo các lớp UI cấp cao nhất . Đây là các trang chính:

• ContentPage

• MasterDetailPage

• NavigationPage

• TabbedPage

• CarouselPage

Ngoài vai trò là nơi chứa các bố cục và chế độ xem, các trang còn cung cấp một menu phong phú các màn hình có sẵn với chức năng hữu ích bao gồm điều hướng và phản ứng cử chỉ.

**Layout**

Lượt xem được đặt và kích thước bởi lớp Container, Bố cục . Bố cục có nhiều loại với các tính năng khác nhau để định dạng chế độ xem của chúng. Các thùng chứa này cho phép các khung nhìn được định dạng chính xác, lỏng lẻo, tuyệt đối với hệ tọa độ hoặc liên quan đến nhau. Bố cục là mô mềm của trang, sụn giữ các khía cạnh chắc chắn, có thể nhìn thấy của trang (Views). Dưới đây là các bố cục chính:

• StackLayout

• FlexLayout

• Grid

• AbsoluteLayout

• RelativeLayout

• ScrollView

• Frame

• ContentView

Các thuộc tính Nội dung và / hoặc Children của bố cục chứa các bố cục và chế độ xem khác. Căn chỉnh ngang(Horizontal) và dọc(vertical) được thiết lập bởi các **HorizontalOptions** và **VerticalOptions**. Hàng, cột và ô trong bố cục có thể được đệm bằng không gian, có kích thước để mở rộng để lấp đầy không gian có sẵn hoặc thu nhỏ để phù hợp với nội dung của chúng.

**Mẹo** bố trí Xamarin.Forms có nguồn gốc từ lớp View, vì vậy mọi thứ có trong một trang thực sự là một dạng xem.

**View**

Lượt xem là các điều khiển, các yếu tố hiển thị và tương tác trên một trang. Các phạm vi này từ các chế độ xem cơ bản như nút, nhãn và hộp văn bản đến các chế độ xem nâng cao hơn như danh sách và điều hướng. Chế độ xem chứa các thuộc tính **content**, **font**, **color**, and **alignment** của chúng. **Horizontal** and **vertical** được thiết lập bởi các thuộc tính s **HorizontalOptions** and **VerticalOptions**. Giống như bố cục, các khung nhìn có thể được đệm bằng không gian, có kích thước để mở rộng để lấp đầy không gian có sẵn hoặc thu nhỏ để phù họp với nội dung của chúng. Đây là các khung nhìn chính được nhóm theo chức năng:

• Basic—fundamental views

• Label

• Image

• Button

• BoxView

• List—make a scrollable, selectable list

• ListView

• SearchBar

• Text entry—user entry of text strings using a keyboard

• Entry

• Editor

• Selection—user choice of a wide range of fields

• Picker

• DatePicker

• TimePicker

• Stepper

• Slider

• Switch

• User feedback—notify the user of app processing status

• ActivityIndicator

• ProgressBar

• Others

• Map

• WebView

**Mẹo** Cẩn thận không nhầm lẫn lớp Xamarin.Forms View với màn hình có nghĩa là màn hình hoặc lớp trình bày.

**Creating a Xamarin.Forms Solution**

Create a Xamarin.Forms solution. In Visual Studio, create a New Project and select project type Visual C# ➤ Cross-Platform ➤ Mobile App (Xamarin.Forms). In Visual Studio for Mac, create a New Solution and select project type Multi-platform ➤ App ➤ Xamarin.Forms ➤ Blank Forms App. Name it FormsExample.

**Chương 1 Xây dựng ứng dụng bằng Xamarin**

Điêu này sẽ tạo ra nhiêu dự án: một cho mã Xamarin.Forms và sau đó là các dự án dành riêng cho nên tảng bao gôm Android và ĨOS. Các dự án dành riêng cho nền tảng có sẵn tùy thuộc vào việc bạn sử dụng PC hay Mac, cho dù bạn đang ở trong Visual Studio hay Visual Studio cho Mac và giấy phép bạn sở hữu. Visual Studio cho Mac sẽ cung cấp cho bạn một dự án ĨOS và một dự án Android. Một PC có Visual Studio sẽ tạo ra ba dự án: một .NET Standard cho Xamarin.Forms, một Android và một IOS.

**Tip** Xamarin miễn phí với giấy phép Visual studio và phiên bản Visual studio **Community** miễn phí.

**Dự án Xamarin.Forms**

Khi sử dụng Visual Studio, dự án Xamarin.Forms chứa App.cs ,định nghĩa và trả về trang chính của ứng dụng. The Application đối tượng phục vụ như là lớp cơ sở của App và cung cấp các Mainpage bất động sản cũng như các sự kiện vòng đời OnStart, OnSleep , và OnResume .

Listing 1-1. App.cs in a New Xamarin.Forms XAML Project

public partial class App : Application

{

public App ()

{

InitializeComponent();

MainPage = new MainPage();

}

protected override void OnStart ()

{

// Handle when your app starts

}

protected override void OnSleep ()

{

// Handle when your app sleeps

}

protected override void OnResume ()

{

// Handle when your app resumes

}

}

Mỗi nền tảng có một lớp bao bọc lấy lớp ứng dụng được chia sẻ và hiển thị nó làm triển khai gốc. Mã mặc định đặt thuộc tính MainPage trong hàm tạo của nó trong trường họp này là một đối tượng ContentPage có tên MainPage. Chúng ta sẽ sớm thay thế MainPage bằng lớp ContentPage của riêng chúng ta và điều khiển vị trí trên nó bằng XAML.

**Mẹo** một thuộc tính Application.Current tham chiếu đến đối tượng ứng dụng hiện tại ở bất kỳ đâu trong ứng dụng của bạn.

Các OnStart, OnSleep , và OnResume phưorng pháp ghi đè tạo ra cho chúng tôi được sử dụng để quản lý các ứng dụng của chúng tôi khi nó được ngừng hoạt động và khởi chạy lại.

**Các phương thức vòng đời ứng dụng: OnStart, OnS ngủ và OnResume**

Khi người dừng nhấp vào nút Quay lại hoặc Trang chủ (hoặc Trình chuyển đổi ứng dụng ) trên thiết bị của họ, một ứng dụng sẽ di chuyển vào nền. Khi họ chọn lại ứng dụng một lần nữa, nó sẽ tiếp tục và di chuyển trở lại nền trước. Sự khởi đầu của một ứng dụng, sự phát triển của ứng dụng từ tiền cảnh sang trạng thái nền sau đó trở lại nền trước một lần nữa, cho đến khi chấm dứt, được gọi là vòng đời của ứng dụng . Lớp ứng dụng bao gồm ba phương thức virtual để xử lý các sự kiện trong vòng đời:

• OnStart -Called khi ứng dụng được đầu tiên bắt đầu. Hữu ích để tải các giá trị vào bộ nhớ mà ứng dụng cần.

• OnStart—Called Tập tin mỗi khi ứng dụng được chuyển vào nền.

Hữu ích cho việc dọn dẹp và bắt đầu các cuộc gọi nền.

• OnResume—Called Gặp phải khi ứng dụng được tiếp tục lại sau khi ở chế độ nền. Hữu ích cho việc tải lại các giá trị vào bộ nhớ và frả về từ các luồng nền.

OnSleep cũng được sử dụng để chấm dứt ứng dụng thông thường (không phải là sự cố). Bất cứ khi nào một ứng dụng chuyển sang trạng thái nền, nó phải được giả định rằng nó có thể không bao giờ trờ lại từ trạng thái đó.

**Mẹo** sử dụng từ điển Properties để duy trì đĩa trong các phương thức này khi ứng dụng được chạy nền.

**Xây dựng trang bằng ContentPage**

Các property Mainpage trong App.cs (Bảng liệt kê 1-1 ) được gán các trang mặc định trong Xamarin. Forms: MainPage . XAML cho MainPage được hiển thị trong Liệt kê 1-2 . Nó chứa một bố cục được gọi là StackLayout và một chế độ xem hoặc điều khiển được gọi là Label.

Liệt kê 1-2. MainPage.xaml

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:local="clr-namespace:MyApp"

x:Class=" MyApp.MainPage">

<StackLayout>

<!-- Place new controls here -->

<Label Text="Welcome to Xamarin.Forms!"

HorizontalOptions="Center"

VerticalOptions="CenterAndExpand" />

</StackLayout>

</ContentPage>

Mã C# của MainPage phía sau rất đơn giản, như bạn có thể thấy trong Liệt kê 1-3 . Lớp này xuất phát từ ContentPage và có một phương thức **InitializeComponent** trong hàm tạo của nó để hiển thị XAML đi kèm.

Listing 1-3. MainPage.xaml.cs in a New Xamarin.Forms XAML Project

public partial class MainPage : ContentPage

{

public MainPage()

{

InitializeComponent();

}

}

ContentPage có các thuộc tính ảnh hưởng đến sự xuất hiện của trang. Các thuộc tính Padding sẽ tạo không gian xung quanh bên lề của trang để cải thiện khả năng đọc và thiết kế. BackgroundImage có thể chứa một hình ảnh được hiển thị trên nền của trang.

Một số thành viên của ContentPage rất hữu ích cho việc điều hướng và quản lý. Các tài sản có chứa văn bản, và các biểu tượng tài sản chứa một hình ảnh sẽ được hiển thị ở phía trên cùng của trang khi NavigationPage được thực hiện. Các phương thức vòng đời OnAppear và OnDisappear có thể được ghi đè để xử lý việc khởi tạo và hoàn thiện ContentPage . Các ToolBarltems là hữu ích cho việc tạo ra một trình đơn thả xuống.

**Xamarin.Android**

Dự án Android chứa một tệp khởi động có tên MainActivity.cs , đinh nghĩa một lớp hoạt động được kế thừa từ Xamarin.Forms.Platform.Android. FormsApplicationActivity như đã thấy trong Liệt kê 1-4 .

**Listing 1-4**. MainActivity.cs in the FormsExample.Droid Project

namespace FormsExample.Droid

{

[Activity(Label = "FormsExample", Icon = "@drawable/icon",

MainLauncher = true, ConfigurationChanges = ConfigChanges.

ScreenSize | ConfigChanges.Orientation)]

public class MainActivity : global::Xamarin.Forms.Platform.

Android.FormsApplicationActivity

{

protected override void OnCreate(Bundle bundle)

{

base.OnCreate(bundle);

global::Xamarin.Forms.Forms.Init(this, bundle);

LoadApplication(new App());

}

}

}

Trong phương thức OnCreate , Xamarin.Forms được khởi tạo và LoadApplication đặt ứng dụng làm ứng dụng hiện tại

**Xamarin.iOS**

Dự án IOS chứa một tệp khởi động có tên là AppDelegate (Liệt kê 1-5 ) kế thừa từ Xamarin.Forms.Platform.iOS.FormsApplicationDelegate

**Listing 1-5.** AppDelegate.cs in the FormsExample.iOS Project

namespace FormsExample.iOS

{

[Register("AppDelegate")]

public partial class AppDelegate : global::Xamarin.Forms.

Platform.iOS.FormsApplicationDelegate

{

public override bool FinishedLaunching(UIApplication app,

NSDictionary options)

{

global::Xamarin.Forms.Forms.Init();

LoadApplication(new App());

return base.FinishedLaunching(app, options);

}

}

}

Xamarin.Forms được khởi tạo trong phương thức Init() và LoadApplication đặt ứng dụng làm trang hiện tại.

Tất cả các công cụ khởi tạo dành riêng cho nền tảng của chúng tôi, Android MainActivity và IOS AppDelegate, đều nhận được trang bắt đầu từ lóp ứng dụng Xamarin.Forms , theo mặc định, trả về trang thử nghiệm.

**Core Library**

Core Library là một dự án trong giải pháp Xamarin.Forms dành cho doanh nghiệp và / hoặc lớp truy cập dữ liệu của một ứng dụng nên phần lớn là độc lập. Mặc dù không được tạo rõ ràng như là một phần của các mẫu giải pháp Xamarin.Forms, một dự án. Core Library là tiêu chuẩn thực hành. Tạo một cái cho mình và thêm nó vào giải pháp của bạn. Điều này có thể chứa các mô hình dữ liệu, tệp hoặc tài nguyên được chia sẻ, truy cập dữ liệu, logic nghiệp vụ hoặc tham chiếu đến PCLs. Đây là nơi dành cho mã không phải giao diện người dùng trung gian hoặc back-end độc lập với nền tảng. Nó được tham chiếu bởi bất kỳ hoặc tất cả các dự án khác cùng giải pháp. Sử dụng nó để tối ưu hóa việc tái sử dụng mã và để tách rời UI projects khỏi lớp truy cập dữ liệu và logic. Lưu ý Core Library là một kiến trúc giải pháp tiên tiến. Nếu bạn mới bắt đầu với Xamarin.Forms, hãy xem xét việc truy cập dữ liệu, logic nghiệp vụ và mã được chia sẻ trong dự án Xamarin.Forms.

**Đặt trang chính của ứng dụng**

Đầu tiên chúng tôi tạo một trang tùy chỉnh trong dự án Xamarin.Forms và đặt nó thành trang chính của ứng dụng. Thêm một tệp mới vào project của bạn và chọn Content Page. Điều này sẽ tạo ra một lớp được kế thừa từ ContentPage .

Gọi nó là ContentPageExample . Cả XAML và mã C# phía sau tệp sẽ được tạo. Đây là tệp XAML , ContentPageExample.xaml :

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="FormsExample.ContentPageExample">

<ContentPage.Content>

<StackLayout>

<Label Text="Welcome to Xamarin.Forms!"

VerticalOptions="CenterAndExpand"

HorizontalOptions="CenterAndExpand" />

</StackLayout>

</ContentPage.Content>

</ContentPage>

Here is the C# code behind called ContentPageExample.cs:

namespace FormsExample

{

public partial class ContentPageExample : ContentPage

{

public ContentPageExample ()

{

InitializeComponent ();

}

}

}

Sau đó quay lại Xamarin.Forms App.cs , chúng tôi cập nhật hàm tạo ứng dụng để đặt phiên bản của lớp ContentPageExample mới của chúng tôi làm MainPage :

namespace FormsExample

{

public class App : Application

{

public App()

{

MainPage = new ContentPageExample();

}

Bây giờ chúng tôi đã có lớp trang tùy chỉnh sẵn sàng và có thể tải lên tệp XAML ContentPageExample của chúng tôi bằng các điều khiển.

**Thêm Xamarin.Forms Views**

View là thuật ngữ để kiểm soát trong Xamarin.Forms, đơn vị xây dựng UI nhỏ nhất. Hầu hết các khung nhìn kế thừa từ lớp View và cung cấp các chức năng UI cơ bản, chẳng hạn như label hoặc button.

Mẹo tất cả các giải pháp mã ví dụ có thể được tìm thấy dưới tiêu đề của cuốn sách này trên

<https://www.apress.com/us/book/9781484240298> trong tab Source Code/

Downloads tab hoặc trên github tại <https://github.com/danhermes/xamarin-xaml-book-examples>

Hãy bắt đầu đơn giản và đặt một số View vào ContentPageExample.xaml.

**Label View**

Nhãn hiển thị văn bản đơn hoặc đa dòng. Dưới đây là một số ví dụ:

<Label Text="Label" FontSize="40" HorizontalOptions="Center" />

<Label FontSize="20" HorizontalOptions="CenterAndExpand">

<Label.Text>

This control is great for

displaying one or more

lines of text.

</Label.Text>

</Label>

Văn bản nhiều dòng xảy ra ngầm khi đủ văn bản được sử dụng mà nó kết thúc tốt đẹp, hoặc rõ ràng với các ngắt dòng.

Một Label xem có hai loại liên kết, xem lý giải và văn bản biện minh. Toàn bộ chế độ xem được chứng minh trong một bố cục bằng các thuộc tính HorizonalOptions và verticalOptions được chỉ định bằng cách sử dụng LayoutOptions . Label text được chứng minh trong một nhãn sử dụng Label’s HorizontalTextAlignment và VerticallTextAlignment properties.

HorizontalTextAlignment = "End"

Bảng liệt kê TextAlocation được gán cho các thuộc tính căn chỉnh này có ba giá trị: Start, Center, and End.

Tiếp theo, các nhãn phải được chỉ định cho một bố cục cho vị trí trên trang. Trong ví dụ này, chứng tôi sử dụng mặc định căn chỉnh văn bản và không khai báo rõ ràng căn chỉnh văn bản.

**Đặt chế độ xem bằng StackLayout**

Một khung nhìn **Layout** hoạt động như một thùng chứa cho các khung nhìn khác. Cho đến khi một ContentPage có thể chỉ có một bố trí hoặc xem, tất cả các View trên trang của chúng tôi phải được đặt trong một khung chứa được gán cho các ContentPage sở hữu nội dung. Ở đây, chúng tôi sử dụng StackLayout, một lớp con của **Layout** có thể ngăn xếp ngăn xếp con xem dọc trong ContentPageExample.xaml:

<StackLayout HeightRequest="1500">

<Label Text = "Label" FontSize="40" HorizontalOptions="Center" />

<Label FontSize="20" HorizontalOptions="CenterAndExpand">

<Label.Text>

This control is great for

displaying one or more

lines of text.

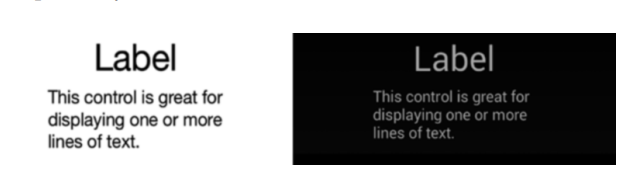
</Label.Text>

</Label>

</StackLayout>

Chúng tôi đặt tất cả các chế độ xem con vào chế độ xem cha mẹ StackLayout và đặt chiều cao được yêu cầu với HeightRequest. HeightRequest đã được đặt lớn hơn trang hiển thị để sau này chúng ta có thể làm cho nó cuộn.

**Lưu ý** Các khung nhìn con StackLayout được đặt mặc định theo chiều dọc trừ khi thứ tự ngang được chỉ định bằng Orientation = "Horizontal".

Biên dịch và chạy mã. Hình 1-7 hiển thị nhãn của chúng tôi trên StackLayout cho IOS và Android, tương ứng:  


*Figure 1-7. Xamarin.Forms Labels on a StackLayout*

Néu bạn đang sử dụng IOS và muốn các dự án Xamarin.Forms của bạn trông giống như các ví dụ trong cuốn sách này có nền đen và văn bản màu trắng hoặc bạn đang sử dụng nền tảng khác và muốn có giao diện IOS nhiều hơn, hãy đặt màu nền và phông chữ màu sắc có thể giúp bạn.

**Background Color and Font Color**

Màu nền của trang và màu phông chữ xem có thể được thay đổi bằng cách sử dụng thuộc tính BackgroundColor của ContentPage và thuộc tính TextColor được tìm thấy trên Chế độ Views dựa trên văn bản .

Nếu bạn đang làm việc trên một dự án IOS và muốn công việc của mình giống với các ví dụ về sách có nền màu đen, hãy thêm dòng này vào trang của bạn:

<ContentPage BackgroundColor= "Black"

Nếu bạn muốn nó trông giống IOS cổ điển hơn, thì hãy đặt nó thành Color.White . Màu văn bản sau đó sẽ được đặt tự động thành màu nhạt hơn . Tuy nhiên, bạn có thể điều khiển màu văn bản theo cách thủ công trên các điều khiển văn bản với thuộc tính TextColor .

<Label TextColor= "White"

**Using Fonts**

Định dạng văn bản trên các điều khiển bằng cách sử dụng các properties này :

*FontFamily* : Đặt tên của phông chữ trong fontFamily property; nếu không, phông chữ mặc định của nền tảng sẽ được sử dụng, ví dụ: FontF Family = "Courier" .

*FontSize* : Kích thước và trọng lượng phông chữ được chỉ định trong thuộc tính FontSize bằng cách sử dụng giá trị kép hoặc liệt kê NamedSize . Dưới đây là một ví dụ sử dụng double:

FontSize = "40". Đặt kích thước tương đối bằng cách sử dụng NamedSize giá trị như NamedSize.Large, sử dụng NamedSize members Large, Medium, Small, and Micro ,ví dụ:

FontSize = " Large ".

*FontAttribut*: Các kiểu phông chữ như in đậm và in nghiêng được chỉ định bằng cách sử dụng thuộc tính FontAttribut. Các thuộc tính đơn được đặt như thế này: Tùy chọn FontAttribut = "Bold" là None, Bold, and Italic.

Nhiều thuộc tính được chỉ định bằng cách sử dụng một chuỗi thuộc tính được định dạng một “[font-face],[attributes],[size]”

**Mẹo** các thuộc tính định dạng văn bản này cũng có thể được thiết lập trên toàn bộ các kiểu sử dụng, được trình bày trong Chương 4 .

**Sử dụng phông chữ dành riêng cho nền tảng**

Đảm bảo tên phông chữ của bạn sẽ hoạt động cho tất cả các nền tảng mục tiêu của bạn, hoặc trang của bạn có thể thất bại một cách bí ẩn. Nếu bạn cần các tên phông chữ khác nhau cho mỗi nền tảng, hãy sử dụng thẻ OnPlatform, đặt giá trị theo nền tảng, như sau:

<Label.FontFamily>

<OnPlatform x:TypeArguments="x:String">

<On Platform="iOS">Courier</On>

<On Platform="Android">Droid Sans Mono</On>

</OnPlatform>

</Label.FontFamily>

**Mẹo** một cách khác để khai báo các thẻ On trong OnPlatform liên quan đến tham số Giá trị.

<On Platform="Android" Value="Droid Sans Mono"/>

**Button View**

Các nút Xamarin.Forms có hình chữ nhật và có thể nhấp. Hãy thêm một nút ole đơn giản:

<Button Text = "Make It So" FontSize="Large" HorizontalOptions="Center"

VerticalOptions="Fill" Clicked="ButtonClicked" />

Các Text property bao gồm các văn bản rõ ràng vào nút. HorizontalOptions và verticaioptions (được thảo luận trong phần tiếp theo) xác định sự liên kết và kích thước của điều khiển. Cài đặt phông chữ NamedSize này làm cho phông chữ Lớn.

**Mẹo** Các nút có thể được tùy chỉnh bằng các thuộc tình BorderColor, BorderWidth, BorderRadius và TextColor. Các BorderWidth được mặc định để zero trên ios.

Thêm nút vào StackLayout của chúng tôi:  
<StackLayout HeightRequest="1500">

<Label Text = "Label" FontSize="40" HorizontalOptions="Center" />

<Label FontSize="20" HorizontalOptions="CenterAndExpand">

<Label.Text>

This control is great for

displaying one or more

lines of text.

</Label.Text>

</Label>

**<Button Text = "Make It So" FontSize="Large" HorizontalOptions="Center"**

**VerticalOptions="Fill" Clicked="ButtonClicked" />**

</StackLayout>

Figure 1-8 shows the new button.

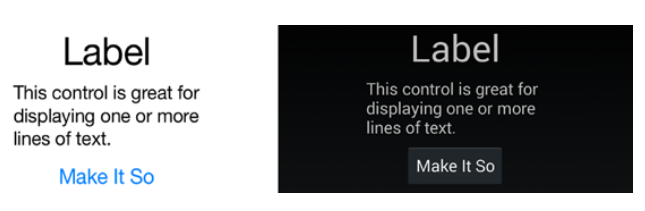


Figure 1-8. Xamarin.Forms Button

Bây giờ, hãy chỉ định một trình xử lý sự kiện trong ContentPageExample.cs , theo dòng:

button.Clicked += (sender, args) =>

{

((Button)sender). text = "It is so!";

};

Hoặc bằng cách chỉ định một phương thức:

button.Clicked += OnButtonClicked;

…Nó được gọi bên ngoài trình xây dựng trang:

void OnButtonClicked(object sender, EventArgs e)

{

((Button)sender).text = "It is so!";

};

Khi bạn bấm vào nút, văn bản nút sẽ thay đổi, như trong Hình 1-9 .

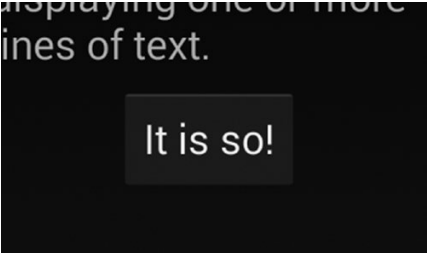


Figure **1-9**. button.Clicked event fired

**Mẹo** BorderWidth chỉ định trọng lượng của đường vẽ nút.

**Cài đặt căn chỉnh và kích thước của chế đô xem: HorizontalOptions(ngang)**

**and VerticalOptions(dọc)**

Ngang và liên kết dọc và, đến một mức độ nhất định, kích thước của điều khiển được quản lý bằng cách thiết lập Horizont aioptions và / hoặc VerticalOptions tính đến một giá trị của LayoutOptions lớp, ví dụ:

<Button HorizontalOptions="Center" VerticalOptions="Fill" />

Cân nhắc trong layout chế độ xem là không gian được cung cấp cho chế độ xem theo layout và các yếu tố xung quanh, không gian đệm xung quanh chế độ xem và kích thước của chế độ xem. Các loại định dạng này được thực hiện bằng cách sử dụng LayoutOptions và AndExpand .

**Justification with LayoutOptions**

Điều khiển layout riêng lẻ được xác định dọc theo một trục đơn bằng cách đặt thuộc tính HorizontalOptions

or VerticalOptions thành một trong các lớp LayoutOptions :

• Bắt đầu bên trái hoặc trên cùng chỉ ra điều khiển (tùy theo Định hướng bố cục).

• Trung tâm điều khiển.

• Kết thúc bên phải hoặc phía dưới Điều chỉnh lại quyền kiểm soát.

• Fill mở rộng kích thước của điều khiển để lấp đầy không gian được cung cấp.

Ví dụ:

<Button HorizontalOptions = "Start" />

**AndExpand Pads with Space**

Thiết HorizontalOptions hoặc VerticalOptions những LayoutOptions lớp cung cấp không gian đệm xung quanh xem:

• StartAndExpand sang trái hoặc trên cùng - chứng minh điều khiển và các miếng đệm xung quanh điều khiển có không gian.

• CenterAndExpand tập trung điều khiển và miếng đệm xung quanh điều khiển với không gian.

• EndAndExpand bên phải hoặc bên dưới - chứng minh điều khiển và miếng đệm xung quanh điều khiển có không gian.

• FillAndExpand mở rộng kích thước của điều khiển và các miếng đệm xung quanh điều khiển với không gian.

Ví dụ:

<Button HorizontalOptions = "StartAndExpand" />

**Mẹo** HorizontalOptions thiết lập để Điền và FillandExpand giống nhau với một điều khiển duy nhất trong một cột.

**VerticalOptions** được đặt thành Center hoặc Fill chỉ hữu ích nếu không gian dọc được cung cấp rõ ràng. Nếu không, các tùy chọn này có thể không làm gì cả. LayoutOptions.Fill sẽ không làm cho đường viền của bạn cao hơn **nếu không có** không gian để phát triển.

VerticalOptions được đặt thành Expand và CenterAndExpand áp đặt không gian đệm xung quanh một điều khiển trong StackLayout.

Tiếp theo hãy tạo một số đầu vào của người dùng.

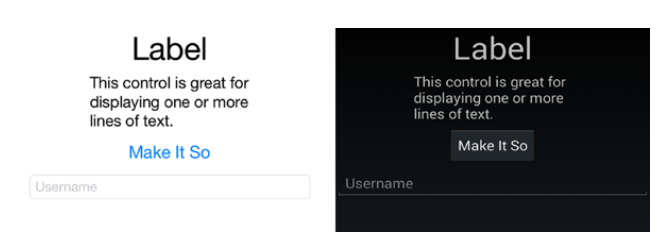
**Entry View for Text Input**

Đoạn mã sau đây tạo một hộp văn bản để người dùng nhập một dòng văn bản đơn lẻ . Entry kế thừa từ lớp InputView , một dẫn xuất của lớp View .

<Entry Placeholder="Username" VerticalOptions="Center" Keyboard="Text" />

Đầu vào của người dùng đi vào thuộc tính Text dưới dạng Chuỗi.

Lưu ý việc sử dụng thuộc tính Placeholder, nhãn nội tuyến cho tên của trường và một kỹ thuật phổ biến trong giao diện người dùng di động thường thích hợp hơn các nhãn tiêu thụ không gian được đặt ở trên hoặc bên cạnh kiểm soát mục nhập. Các Keyboard sở hữu là thành viên của InputView và cung cấp một loạt các lựa chọn cho bàn phím trên màn hình xuất hiện cho đầu vào, bao gồm chữ, số , Điện thoại, URL , và Email. Hãy nhớ thêm mục vào StackLayout của bạn (xem Liệt kê 1-6 sau trong chưomg này). Hình 1-10 cho thấy mục tiếp theo mới cho tên người dùng.



***Figure 1-10***. Xamarin.Forms user entry view

**Mẹo** đặt IsPassword = "True" để thay thế các chữ cái đã nhập bằng dấu chấm**. Đối với mục nhập đa dòng, sử dụng điều khiển Editor**.

**BoxView**

Điều khiển BoxView tạo ra một hình chữ nhật đồ họa có màu, hữu ích như một trình giữ chỗ mà sau này có thể được thay thế bằng một hình ảnh hoặc nhóm điều khiển hoặc nhóm điều khiển phức tạp khác. Điều khiển này rất hữu ích khi bạn chờ đợi nhà thiết kế thực hiện hành động của mình.

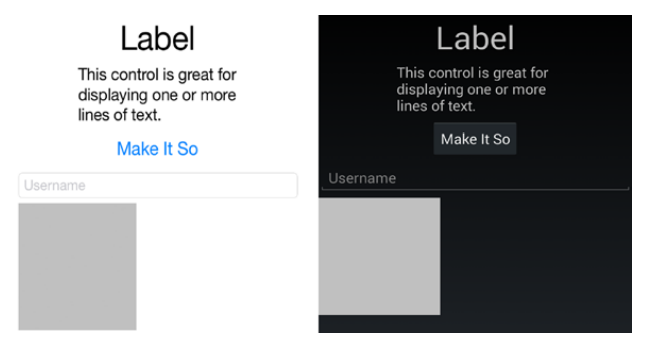
<BoxView Color="Silver" WidthRequest="150" HeightRequest="150"

HorizontalOptions="StartAndExpand" VerticalOptions="Fill" />

Các Màu sở hữu có thể được thiết lập để bất kỳ Color member. Các kích thước mặc định là 40 X 40 pixel, có thể được thay đổi dùng WidthRequest và HeightRequest properties.

**Mẹo** Hãy cẩn thận khi thiết **HorizontalOptions** và **VerticalOptions để Fill** và **FillAndExpand,** vì điều này có thể ghi **đè HeightRequest của bạn** và **WidthRequest** nhiềuchiều.

Thêm BoxView vào StackLayout của bạn (xem Liệt kê 1-6 sau trong chương này) và xem kết quả ở đây trong Hình 1-11 .



***Figure 1-11***. Xamarin.Forms BoxView

Cuối cùng, nhà thiết kế của bạn sẽ cung cấp cho bạn những biểu tượng đã hứa và bạn có thể thay thế BoxView của mình bằng hình ảnh thật.

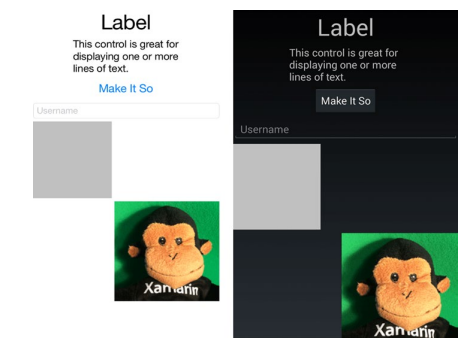
**Image View**

Các hình ảnh xem giữ một hình ảnh để hiển thị trên trang web của bạn từ một tập tin trên máy tính hoặc trực tuyến:

<Image Source="monkey.png" Aspect="AspectFit" HorizontalOptions="End"

VerticalOptions="Fill" />

***Figure 1-12*** shows the monkey image at the bottom right.



***Figure 1-12***. Image view

Hãy xem cách xử lý một hình ảnh.

**Local Images**

Các tệp hình ảnh cục bộ có các thư mục hình ảnh dành riêng cho nền tảng trong các dự án tương ứng của chúng:

Android sử dụng thư mục Resources/drawable. Đừng sử dụng các ký tự đặc biệt trong tên tệp.

Build Action phải là et et Android Resource .

IOS 9 trở lên sử dụng Asset Lists and Image Sets có thể được thiết lập trong Visual Studio trên dự án IOS. Apple đã không chấp nhận cách tiếp cận thư mục /Resources nơi chúng tôi sẽ tạo hình ảnh cho màn hình Retina với hậu tố @ 2x hoặc @ 3x trên tên tệp.

**Định cỡ hình ảnh: Thuộc tính khía cạnh(Aspect Property)**

Các Image.Aspect sở hữu quyết định hình ảnh kích thước và được thiết lập bằng cách sử dụng các Aspect Enumerator- ví dụ:

<Image Source="monkey.png" Aspect="AspectFit" HorizontalOptions="End"

VerticalOptions="Fill" />

Đây là các thành viên Aspect :

**AspectFill : Chia tỷ lệ** hình ảnh để lấp đầy khung nhìn, cắt nếu cần thiết.

**AspectFit : Chia** tỷ lệ hình ảnh để vừa trong chế độ xem duy trì tỷ lệ khung hình mà không bị biến dạng và không gian mở rộng nếu cần thiết (hộp thư).

**Fill**: Chia tỷ lệ hình ảnh để lấp đầy khung nhìn hoàn toàn và chính xác, có thể làm biến dạng hình ảnh. Đó là các tùy chọn định dạng hình ảnh. Tiếp theo chúng tôi sẽ làm cho hình ảnh của chúng tôi có thể nhấp.

**Tạo một hình ảnh có thể nhấp bằng GestureRecognizer**

Hình ảnh và biểu tượng có thể áp dụng là phổ biến trong các ứng dụng di động cho các hành động và dẫn đường. Giống như nhiều chế độ xem Xamarin.Forms, Image không có một sự kiện nhấp hoặc nhấn và phải được kết nối bằng cách sử dụng lớp GestureRecognizer. Một nhận dạng cử chỉ là một lớp có thể được thêm vào nhiều chế độ xem để đáp ứng tương tác của người dùng. Nó hiện đang hỗ trợ chỉ cần cử chỉ nhấn. Các thuật ngữ nhấp và nhấn được sử dụng thay thế cho nhau trong giao diện người dùng di động phát triển

Thêm nhận dạng cử chỉ tiêu chuẩn vào hình ảnh.

<Image Source="monkey.png" Aspect="AspectFit" HorizontalOptions="End"

VerticalOptions="Fill" >

<Image.GestureRecognizers>

<TapGestureRecognizer Tapped="ImageTapped"/>

</Image.GestureRecognizers>

</Image>

Tạo một trình xử lý để quản lý sự kiện Taps . Thay đổi Opacity của hình ảnh thành .5 trong trình xử lý, điều này sẽ làm mờ hình ảnh một chút khi gõ.

protected void ImageTapped(object sender, EventArgs e) {

Image image = ((Image)sender);

image.Opacity = .5;

image.Opacity = 1;

}

}

Hãy thử và làm cho con khỉ của bạn mờ dần để bạn có thể thấy rằng bộ nhận dạng cử chỉ hoạt động.

**Mẹo** triển khai thay thế GestureRecognizer sử dụng thuộc tính Command :

<Image.GestureRecognizers>

<TapGestureRecognizer Command="{Binding ImageTappedCommand}"/>

</Image.GestureRecognizers>

Phản hồi của người dùng là một khái niệm quan trọng trong phát triển giao diện người dùng di động. Bất cứ khi nào người dùng làm điều gì đó trong giao diện người dùng, cần có một số xác nhận tinh tế của ứng dụng. Một vòi, ví dụ, sẽ phản hồi cho người dùng với phản hồi có thể nhìn thấy. Thông thường một hình ảnh sẽ chuyển sang màu xám hoặc có nền trắng trong một giây khi chạm vào. Chúng ta hãy làm điều đó một cách chuyên nghiệp bằng cách sử dụng thuộc tính Opacity của hình ảnh nhưng thêm async / await để tạo ra một chút mờ nhạt mà không ảnh hưởng đến hiệu suất của ứng dụng.

Thêm một async / await với độ trễ sẽ làm cho hình ảnh hơi mờ đi trong một phần của giây. Hãy nhớ thêm bằng System.Threading.Tasks; đến đầu tệp .cs của bạn.

**async** protected void ImageTapped(object sender, EventArgs e) {

Image image = ((Image)sender);

image.Opacity = .5;

**await Task.Delay(200);**

image.Opacity = 1;

}

Chạm vào hình ảnh bây giờ sẽ làm mờ hình ảnh một chút, sau đó trở lại bình thường, cung cấp trải nghiệm người dùng phản ứng nhanh.

**Mẹo** Để hoạt ảnh tinh tế hơn, thay vì Opacity, hãy sử dụng phương thức Fadeto:

await image.FadeTo(0.5, 450);

await Task.Delay(1000);

await image.FadeTo(1, 450);

Trong các dự án của riêng bạn, bạn sẽ sử dụng các trình nhận dạng cử chỉ (và async / await) để thực sự làm gì đó khi một hình ảnh được gõ. Nếu bạn muốn xem async / await in action trong ví dụ này, hãy nâng cấp Delay to 2000, sau đó nhấp vào nút “Make It So” trong khi chờ đợi và bạn sẽ thấy ứng dụng vẫn phản hồi. Bạn có thể làm nhiều thứ trong trình xử lý Taps này mà không làm gián đoạn dòng chảy của ứng dụng! Thông thường khi nhấn nút hoặc hình ảnh, kết quả sẽ được làm nền bằng cách sử dụng async / await để có trải nghiệm người dùng tối ưu.

**Mẹo** **Async / await** là một kỹ thuật C# tiêu chuẩn để xếp hàng các hoạt động trong nền cho hoạt động đồng thời bằng cách sử dụng thư viện song song tác vụ Task Parallel Library (TPL).Nhiều phương pháp Xamarin và functions được cung cấp để xử lý nền bằng cách sử dụng **async / await.**

Chúng tôi có thêm một khung nhìn để thêm, một lớp container để cho phép cuộn các khung nhìn của chúng tôi.

**ScrollView**

Các scrollview bố trí chứa a single child và kiến thức sâu rộng scrollability đến nội dung của nó:

<ScrollView VerticalOptions="FillAndExpand">

Ở đây chúng ta lồng stackLayout trong ScrollView này, vì vậy toàn bộ bố cục chế độ xem của chúng tôi sẽ có thể cuộn được.

<ScrollView VerticalOptions="FillAndExpand">

<StackLayout HeightRequest="1500">

<Label Text = "Label" FontSize="40" HorizontalOptions="Center" />

...

</StackLayout>

</ScrollView>

**Mẹo** ScrollView cuộn theo chiều dọc theo mặc định nhưng có thể cuộn ngang bằng cách sử dụng thuộc tình Orientation(định hướng). Ví dụ: Orientation = "Horizontal"

Đó là điều đó cho các quan điểm trên trang này. Cảm ứng cuối cùng sẽ được đệm xung quanh toàn bộ trang, do đó, các lượt xem sẽ không bị trộn lẫn với các cạnh của màn hình.

**Padding Around the Entire Page**

Các Padding của ContentPage thuộc tính sẽ tạo không gian xung quanh toàn bộ pa ge. Đây là sự phân chia property:

<ContentPage.Padding> [left], [top], [right], [bottom]

</ContentPage.Padding>

Ví dụ này sẽ đặt phần đệm trái, phải và dưới cùng, nhưng không phải trên cùng:

<ContentPage.Padding> 10, 0, 10, 5 </ContentPage.Padding>

Mã này sẽ đệm các mặt ngang, trái và phải và các mặt dọc, trên và dưới

<ContentPage.Padding> 10, 5 </ContentPage.Padding>

Điều này sẽ đặt không gian bằng nhau ở cả bốn phía:

<ContentPage.Padding> 10 </ContentPage.Padding>

Nếu bạn đang sử dụng iPhone hoặc iPad, thì ứng dụng của bạn có thể mở rộng lên phía trên màn hình, che khuất thanh trạng thái. Ví dụ sau đây sẽ trượt một trang ngay bên dưới thanh trạng thái IOS trong khi vẫn giữ trang thứ đó ở phía trên màn hình cho các hệ điều hành khác. Các OnPlatform phưong pháp cung cấp các giá trị hoặc các hành động khác nhau tùy thuộc vào hệ điều hành bản địa (IOS, Android). Trong trường hợp này, thuộc tính Padding phụ thuộc vào nền tảng.

<ContentPage.Padding>

<OnPlatform x:TypeArguments="Thickness">

<On Platform="iOS" Value="10, 20, 10, 5"/>

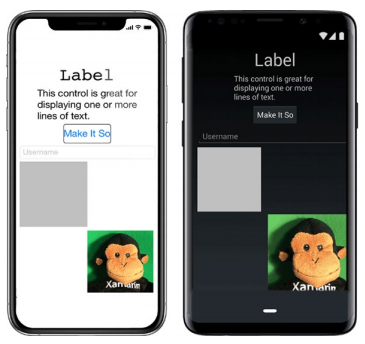
<On Platform="Android" Value="10, 0, 10, 5"/>

</OnPlatform>

</ContentPage.Padding>

Biểu thức Padding cuối cùng này là những gì chúng tôi sử dụng trong dự án này và trong hầu hết các dự án trong cuốn sách này, đệm xung quanh các cạnh của trang có nhiều chỗ hơn ở đầu trên IOS cho thanh trạng thái.

Hình 1-13 cho thấy một bản dựng cuối cùng và chạy trên cả hai dạng platforms.



***Figure 1-13***. Final build and run of the FormsExample solution

**CODE COMPLETE: Adding Xamarin.Forms Views**

Các liệt kê 1-6 và 1-7 cung cấp mã hoàn chỉnh cho các khung nhìn Xamarin.Forms được thêm vào trong giải pháp FormsExample . Danh sách này chứa hình thức OnPlatform gần đây hơn .

***Listing 1-6***. ContentPageExample.xaml in the FormsExample Project

<?xml version="1.0" encoding="UTF-8"?>

<ContentPage xmlns=<http://xamarin.com/schemas/2014/forms> xmlns:x="http://

schemas.microsoft.com/winfx/2009/xaml" x:Class="FormsExample.

ContentPageExample">

<ContentPage.Padding>

<OnPlatform x:TypeArguments="Thickness">

<OnPlatform.iOS>

10, 20, 10, 5

</OnPlatform.iOS>

<OnPlatform.Android>

10, 0, 10, 5

</OnPlatform.Android>

<OnPlatform.WinPhone>

10, 0, 10, 5

</OnPlatform.WinPhone>

</OnPlatform>

</ContentPage.Padding>

<ContentPage.Content>

<ScrollView VerticalOptions="FillAndExpand">

<StackLayout HeightRequest="1500">

<Label Text = "Label" FontSize="40"

HorizontalOptions="Center" />

<Label FontSize="20" HorizontalOptions="CenterAndExpand">

<Label.Text>

This control is great for

displaying one or more

lines of text.

</Label.Text>

</Label>

<Button Text = "Make It So" FontSize="Large"

HorizontalOptions="Center" VerticalOptions="Fill"

Clicked="ButtonClicked" />

<Entry Placeholder="Username" VerticalOptions="Center"

Keyboard="Text" />

<BoxView Color="Silver" WidthRequest="150"

HeightRequest="150" HorizontalOptions="StartAndExpand"

VerticalOptions="Fill" />

<Image Source="monkey.png" Aspect="AspectFit"

HorizontalOptions="End" VerticalOptions="Fill" >

<Image.GestureRecognizers>

<TapGestureRecognizer Tapped="ImageTapped"/>

</Image.GestureRecognizers>

</Image>

</StackLayout>

</ScrollView>

</ContentPage.Content>

</ContentPage>***Listing 1-7***. ContentPageExample.xaml.cs in the FormsExample Project

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using Xamarin.Forms;

namespace FormsExample

{

public partial class ContentPageExample : ContentPage

{

public ContentPageExample ()

{

InitializeComponent ();

}

protected void ButtonClicked(object sender, EventArgs e) {

((Button)sender).Text = "It is so!";

}

async protected void ImageTapped(object sender, EventArgs e) {

Image image = ((Image)sender);

image.Opacity = .5;

await Task.Delay(200);

image.Opacity = 1;

}

}

}

**Tóm lược**

Xamarin.Forms cung cấp một điểm khởi đầu để phát triển giao diện người dùng ứng dụng di động đa nền tảng, được tải đầy đủ với các trang, bố cục và chế độ xem được tùy chỉnh. Cuốn sách này đã khắc phục sự phát triển Xamarin bằng cách sử dụng XAML để khai báo UI vói mã C# phía sau.

Một giải pháp Xamarin.Forms thường có một dự án riêng cho từng nền tảng sau: Android và IOS. Một Xamari dự án Xamarin.Forms rất hữu ích đối với nhà ở nền tảng chéo UI, và một dự án Lõi Thư viện chứa các lớp truy cập logic kinh doanh và dữ liệu.

Các nhà phát triển phải đối mặt với quyết định của Xamarin.Forms so với cách tiếp cận UI dành riêng cho nền tảng với Xamarin.Android và Xamarin.iOS.. Càng phát hành nhiều Xamarin.Forms, càng ít quyết định, vì Xamarin.Forms tiếp cận chức năng của API UI gốc. Trình kết xuất, hiệu ứng và chế độ xem tùy chỉnh của Xamarin.Forms giúp chúng tôi kết hợp hai cách tiếp cận.

View là thuật ngữ Xamarin.Forms cho kiểm soát, và chúng tôi đào sâu vào một vài trong số các quan điểm thường xuyên nhất được sử dụng: Label, Entry, BoxView, Image, StackLayout, và ScrollView.

Xamarin.Forms XAML, giống như tất cả các ngôn ngữ đánh dấu, đi kèm với bộ cân nhắc, ngữ pháp, quy tắc và kỹ thuật riêng để giúp chúng tôi xây dựng giao diện người dùng ứng dụng của mình. Hãy đi sâu vào XAML!