**Chương 1**

**Xây dựng ứng dụng với Xamarin**

Xamarin trở thành đa nền tảng bao gồm includes iOS, Android, Windows, macOS, Tizen, WPF, Hololens, GTK,…

Bắt nguồn từ dự án mã nguồn mở Mono – mang .NET lên nền tảng Linux, nền tảng Xamarin là một bản port của .NET cho hệ điều hành iOS và Android.

Phía dưới của Xamarin.Android là Mono for Android, và phía dưới của 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 iOS được khớp với bản phát hành Xamarin mới bao gồm các ràng buộc với API mới của chúng. Xamarin.Forms là một lớp nằm trên các liên kết UI khác, cung cấp thư viện UI đa nền tảng.

Chương này giới thiệu 2 cách để xây dựng một ứng dụng bằng Xamarin:

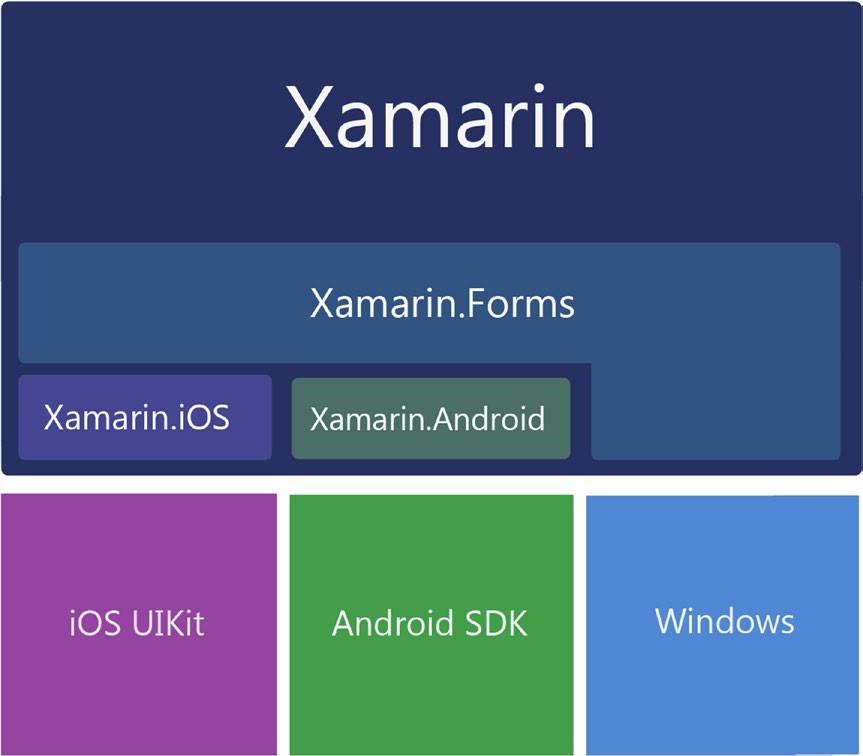
* Dùng *Xamarin.Forms* – thư viện UI đa nền tảng
* Tiếp cận 2 hướng dành riêng cho các nền tảng Xamarin.Android và Xamarin.iOS

Chúng ta sẽ tìm hiểu về Xamarin.Forms hữu dụng trong xây dựng ứng dụng trên các nền tảng khác nhau. Sau đó sẽ nghiên cứu xây dựng giao diện người dùng Xamarin.Forms với cashc sử dụng pages, layout, views.

1

# 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 hợp các thành phần UI đa nền tảng hiển thị trong mỗi ba hệ điều hành gốc này (xem Hình 1-1).



***Figure 1-1.*** *Xamarin libraries bind to native OS libraries*

## XAML vs. 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 giao diện người dùng trong Xamarin.Forms, trong C # bằng cách sử dụng API Xamarin.Forms 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 tất cả là về XAML. Bạn có thể tạo cùng loại UI trong 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 View, 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 “chúng tôi xây dựng Xamarin.Forms code đầu tiên. Điều đó có nghĩa là tất cả các tính năng được tạo trước tiên để hoạt động bằng C#, sau đó chúng tôi triển khai chúng cho XAML”

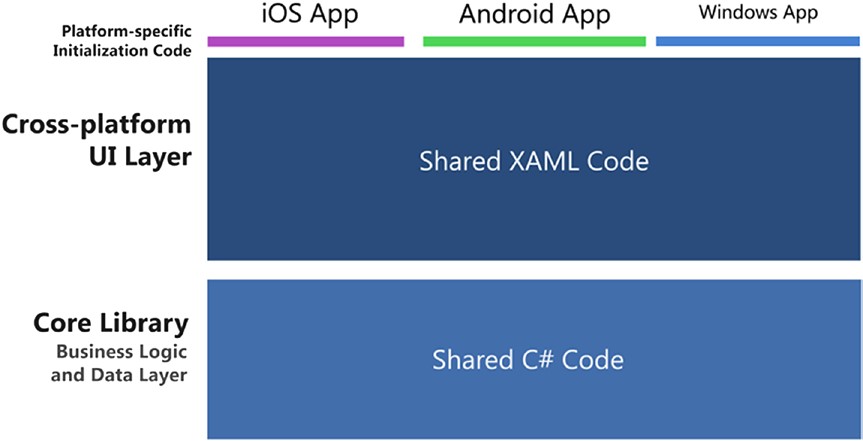
Các phần tử Xamarin.Forms được xây dựng bằng các lớp Page, Layout và View. API này cung cấp một loạt các mẫu UI di động đa nền tảng tích hợp. Bắt đầu với các đối tượng Page cấp cao nhất, nó cung cấp các trang menu quen thuộc như NavigationPage cho các menu phân cấp, TabbedPage cho các menu tab, MasterDetailPage để tạo ngăn kéo điều hướng, CarouselPage để cuộn các trang hình ảnh và ContentPage, một lớp cơ sở để tạo các trang tùy chỉnh. Các bố cục trải rộng các định dạng tiêu chuẩn được sử dụng trên các nền tảng khác nhau bao gồm StackLayout, AbsoluteLayout, RelativeLayout, Grid, ScrollView và ContentView ở lớp layout cơ sở. Sử dụng bên trong các bố cục đó là hàng tá các điều khiển hoặc chế độ xem quen thuộc, chẳng hạn như ListView, Button, DatePicker và TableView. Nhiều trong số các view này có các tùy chọn ràng buộc dữ liệu tích hợp.

**Tip** Có các từ đồng nghĩa khác cho *screens* di động, chẳng hạn như *views* và *pages* và chúng được sử dụng thay thế cho nhau. một *view* có thể có nghĩa là một *screen* nhưng cũng có thể đề cập đến một *control* 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 chúng tôi có thể phát triển UI gốc cho cả ba nền tảng mà hầu như không có kiến thức về Android và iOS UI. Hay nhưng hãy cẩn thận! Những người theo chủ nghĩa thuần túy cảnh báo rằng cố gắng xây dựng các ứng dụng cho các nền tảng này mà không hiểu về API gốc là một cam kết liều lĩnh. Chúng ta cần quan tâm đến các nền tảng Android và iOS, sự phát triển, tính năng, đặc điểm riêng và bản phát hành của chúng.

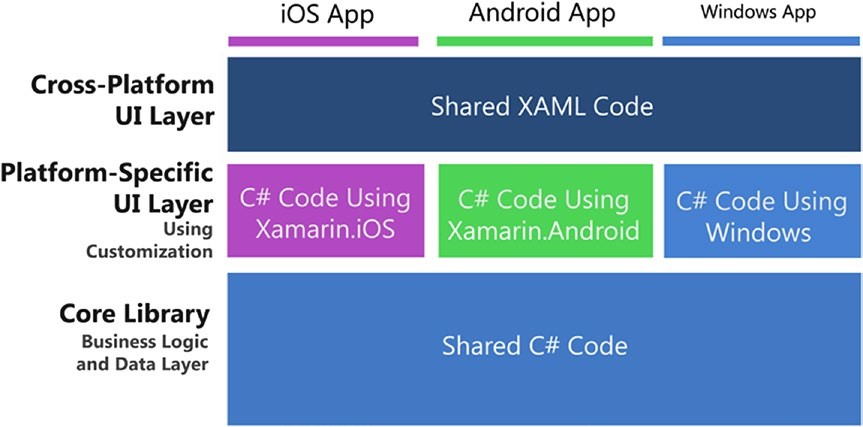
## Kiến trúc cách dùng Xamarin.Forms

Một trong những lợi ích lớn nhất của Xamarin.Forms là nó cho chúng ta khả năng phát triển các ứng dụng di động gốc cho một số nền tảng cùng một lúc. Hình 1-2 cho thấy kiến trúc giải pháp cho ứng dụng Xamarin.Forms đa nền tảng được phát triển cho iOS, Android và bất kỳ nền tảng được hỗ trợ nào khác. Theo tinh thần của kiến trúc tốt và khả năng tái sử dụng, một giải pháp đa nền tảng Xamarin.Forms thường sử dụng mã ứng dụng C # được chia sẻ chứa logic truy cập dữ liệu và lớp truy cập dữ liệu, được hiển thị ở mức dưới cùng của sơ đồ. Điều này thường được gọi là Thư viện lõi. Lớp UI Xamarin.Forms đa nền tảng cũng là C # và được mô tả là lớp giữa trong hình. Lớp mỏng, rời rạc ở trên cùng là một lượng nhỏ mã C # UI dành riêng cho các nền tảng trong các dự án để khởi tạo và chạy ứng dụng trong mỗi HĐH gốc.



### *Figure 1-2. Xamarin.Forms solution architecture: one app for multiple platforms*

Hình 1-2 được đơn giản hóa để truyền đạt các nguyên tắc cơ bản của Xamarin.Forms. Thực tế là sự lai tạo giữa Xamarin.Forms và mã dành riêng cho nền tảng là cần thiết và được khuyến khích. Nó có thể xảy ra ở một số cấp độ. Đầu tiên, trong các tùy chọn tùy chỉnh Xamarin.Forms, bao gồm trình render tùy chỉnh, hiệu ứng và chế độ xem gốc. Tùy chỉnh cung cấp cho chúng tôi các lớp dành riêng cho nền tảng để hiển thị các tính năng dành riêng cho nền tảng trên trang Xamarin.Forms. Việc lai tạo cũng có thể xảy ra trong các hoạt động Android dành riêng cho nền tảng và bộ điều khiển xem iOS chạy cùng với các trang Xamarin.Forms hoặc trong các lớp dành riêng cho nền tảng được gọi là cần thiết để xử lý chức năng gốc như vị trí, máy ảnh, đồ họa hoặc hoạt hình. Cách tiếp cận tinh vi này (hiện đang phổ biến) dẫn đến một kiến trúc phức tạp hơn, được hiển thị trong Hình 1-3, và phải được xử lý cẩn thận. Lưu ý việc bổ sung lớp UI dành riêng cho nền tảng.



### *Figure 1-3. Xamarin.Forms architecture with customization*

**Note** Chương 8 cung cấp thêm về việc sử dụng code tùy biến và code dành riêng cho các nền tảng trong các giải pháp Xamarin.Forms.

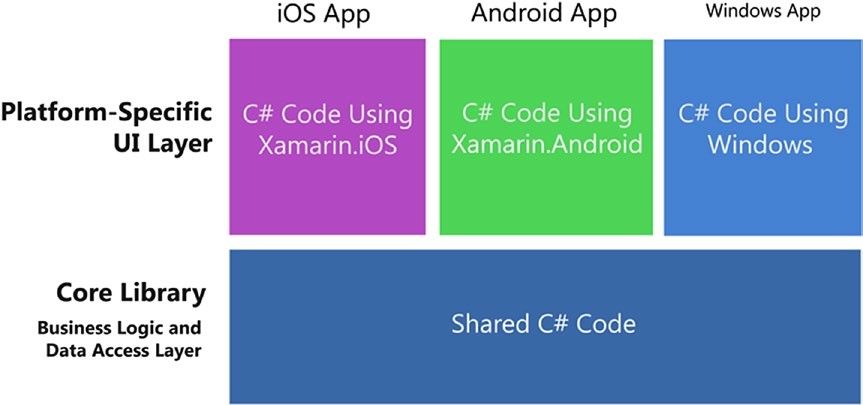
**Hiểu cách tiếp cận giao diện người dùng dành riêng cho mỗi nền tảng**

Trước Xamarin.Forms, đã có các tùy chọn UI dành riêng cho nền tảng, bao gồm các thư viện Xamarin.Android, Xamarin.iOS và Windows Phone SDK. Xây dựng screens bằng các UI dành riêng cho nền tảng đòi hỏi một số hiểu biết về UI gốc được các thư viện này trưng ra. Chúng tôi không cần mã hóa trực tiếp trong iOS UIKit hoặc Android SDK, vì chúng tôi đã loại bỏ một lớp khi sử dụng các ràng buộc Xamarin trong C #. Tất nhiên, sử dụng SDK Windows, chúng tôi đã mã hóa nguyên bản trong C # dựa trên HĐH Windows. Ưu điểm của việc sử dụng UI dành riêng cho nền tảng của Xamarin là các thư viện này được thiết lập và có đầy đủ tính năng. Mỗi lớp container và lớp điều khiển riêng có rất nhiều thuộc tính và phương thức.

**Note** Chúng tôi không nói về sự phát triển ui gốc bằng Objective-C hoặc Java ở đây mà là việc sử dụng các ràng platform-specific Xamarin C# cho các native ui library. Để tránh nhầm lẫn như vậy, cuốn sách này ủng hộ thuật ngữ platform-specific khi đề cập đến các thư viện Xamarin, nhưng các nhà phát triển Xamarin đôi khi sẽ sử dụng thuật ngữ native để chỉ việc sử dụng các thư viện dành riêng cho nền tảng Xamarin.iOS và Xamarin.Android.

## Kiến trúc giao diện người dùng ở các nền tảng

Hình 1-4 cho thấy cách một giải pháp dành riêng cho nền tảng được thiết kế để chia sẻ đa nền tảng mã ứng dụng C # chứa logic truy cập dữ liệu và lớp truy cập dữ liệu, giống như giải pháp Xamarin.Forms. Lớp UI là một câu chuyện khác: đó là tất cả các nền tảng cụ thể. Mã UI C # trong các dự án này sử dụng các lớp được liên kết trực tiếp với API gốc: ràng buộc sans iOS, Android hoặc Windows.



### *Figure 1-4. Platform-specific UI solution architecture*

# Nếu bạn so sánh sơ đồ này với sơ đồ Xamarin.Forms trong Hình 1-2, bạn sẽ thấy rằng có rất nhiều mã hóa được thực hiện ở đây: một giao diện người dùng cho mọi nền tảng thay vì một nền tảng cho tất cả. Tại sao mọi người sẽ bận tâm để làm theo cách này? Có khá nhiều lý do chính đáng tại sao một số hoặc thậm chí tất cả các mã có thể được thực hiện tốt hơn theo cách này. Vậy làm thế nào để chúng ta biết khi nào nên sử dụng Forms?

# Lựa chọn Xamarin.Forms hay là Platform-Specific UI

Hầu hết các dự án Xamarin đều phải đối mặt với quyết định này:

Tôi sử dụng cái nào, Xamarin.Forms hoặc Platform-Specific UI với Xamarin?

Câu trả lời cho câu hỏi của chúng tôi sẽ bao gồm từ cái này, cái kia, cho cả hai, tùy thuộc vào nhu cầu của bạn. Dưới đây là hướng dẫn đề xuất:

**Sử dụng Xamarin.Forms trong trường hợp:**

*Học Xamarin: Nếu bạn mới phát triển di động bằng C #, thì Xamarin.Forms là một cách tuyệt vời để bắt đầu!*

*Chia sẻ mã UI: Tiết kiệm thời gian và tiền phát triển và thử nghiệm bằng cách viết UI chỉ một lần cho tất cả các nền tảng của bạn với Xamarin. Biểu mẫu (ví dụ: Android, iOS).*

*Ứng dụng kinh doanh: Xamarin.Forms thực hiện tốt những điều này, hiển thị dữ liệu cơ bản, điều hướng và nhập dữ liệu. Đây là một phù hợp tốt cho nhiều ứng dụng kinh doanh.*

*Thiết kế cơ bản: Xamarin.Forms cung cấp các điều khiển với các tính năng thiết kế cơ sở, tạo điều kiện cho định dạng hình ảnh cơ bản.*

*Ứng dụng đa nền tảng đơn giản: Xamarin.Forms là tuyệt vời để tạo screen cơ bản đầy đủ chức năng. Đối với các screen phức tạp hơn, tận dụng trình kết xuất tùy chỉnh Xamarin.Forms để biết chi tiết cụ thể về nền tảng.*

**Sử dụng a platform-specific UI (Xamarin.iOS or Xamarin.Android) khi**

*Màn hình phức tạp: Khi toàn bộ màn hình (hoặc toàn bộ ứng dụng) yêu cầu cách tiếp cận giao diện người dùng và thiết kế phức tạp và phức tạp, và Xamarin.Forms không hoàn thành nhiệm vụ, hãy sử dụng UI cụ thể cho nền tảng bằng Xamarin.Android và Xamarin. iOS.*

*Ứng dụng dành cho người tiêu dùng: Giao diện người dùng dành riêng cho nền tảng có mọi thứ mà nhà phát triển cần để tạo ứng dụng dành cho người tiêu dùng với thiết kế trực quan phức tạp, độ nhạy cử chỉ sắc thái và đồ họa và hoạt hình cao cấp.*

*Thiết kế cao: Cách tiếp cận này cung cấp API UI gốc hoàn chỉnh với quyền truy cập cấp thấp vào các thuộc tính thiết kế trên mỗi điều khiển, cho phép tiêu chuẩn thiết kế trực quan cao. Hoạt hình và đồ họa bản địa cũng có sẵn với phương pháp này.*

*Ứng dụng một nền tảng: Nếu bạn chỉ xây dựng một nền tảng và cách tiếp cận đa nền tảng cho ứng dụng của bạn không quan trọng trong tương lai gần (một trường hợp hiếm gặp ngay cả khi bạn bắt đầu với một nền tảng), hãy cân nhắc sử dụng một nền tảng giao diện người dùng cụ thể.*

Tuy nhiên, các nhà phát triển thông minh đang tạo ra các ứng dụng Forms ngày càng tiên tiến hơn. Ngoài ra, nhóm phát triển Xamarin tại Microsoft phát triển nhanh chóng. Với mỗi bản phát hành mới của Xamarin.Forms, nhiều thuộc tính và phương thức được bao gồm trong các ràng buộc, đưa thư viện này đến gần hơn với các nền tảng dành riêng cho nền tảng và giúp chúng ta tăng quyền kiểm soát đối với giao diện người dùng đa nền tảng. Ngoài ra, các dự án nguồn mở và các công cụ của bên thứ ba như Telerik, UI cho Xamarin và Syncfusion Các điều khiển UI Xamarin đang nhanh chóng mở rộng các tùy chọn có sẵn với các điều khiển, biểu đồ và lưới dữ liệu bổ sung.

## Sử dụng Custom Renderers, Effects, and Native Views

Bạn cuối cùng sẽ cần nhiều hơn từ Xamarin.Forms hơn là thứ nó mang. Khi Xamarin.Forms yêu cầu các tác vụ hoặc thiết kế phức tạp, hầu như mọi thứ đều có thể sử dụng tùy chỉnh. Trình render tùy chỉnh cung cấp quyền truy cập vào các lớp kết xuất màn hình, dành riêng cho nền tảng, được gọi là renderer, sử dụng các điều khiển dành riêng cho nền tảng để tạo tất cả các màn hình Xamarin.Forms. Bất kỳ màn hình Xamarin.Forms nào cũng có thể được chia thành các màn hình và các lớp dành riêng cho nền tảng sử dụng phương pháp này. Điều này có nghĩa là chúng ta có thể viết một trang hoặc ứng dụng Xamarin.Forms và tùy chỉnh nó theo nền tảng bất cứ khi nào cần thiết. Thông tin thêm về điều này trong Chương 8. Custom renderer rất mạnh mẽ và kỹ lưỡng trong việc triển khai dưới dạng trình hỗ trợ cụ thể nền tảng của các yếu tố UI Xamarin.Forms. Nếu bạn muốn một cái gì cao, như chỉ đơn thuần là tùy chỉnh một thuộc tính trên điều khiển Xamarin.Forms, hãy xem xét đến effects. Ngoài việc để lộ các thuộc tính, effects còn có khả năng truyền tham số cho các thuộc tính đó và xác định các sự kiện trên các điều khiển Xamarin.Forms . Bạn truyền tham số cho effect bằng các thuộc tính đính kèm hoặc common language runtime (CLR).

Đôi khi bạn chỉ muốn một kiểm soát bản địa thực sự. Bạn sẽ giải quyết cho không có gì ngoài sức mạnh tuyệt đối. Rất may, giờ đây, một cách để có được điều này trong

Xamarin.Forms thông qua khai báo chế độ xem gốc. Chúng dễ sử dụng nhất trong XAML, thứ hai là C #.

# Tất cả điều này có nghĩa là bạn có thể viết một trang hoặc ứng dụng Xamarin.Forms và tùy chỉnh nó theo nền tảng.

**Khám phá các yếu tố của giao diện người dùng di động**

Xamarin là một công cụ hợp nhất phục vụ một số nền tảng, nhiều trong số đó có thể có các tên khác nhau cho cùng một thứ. Dưới đây là một số thuật ngữ thống nhất, được cân nhắc rất nhiều theo hướng của Xamarin.Forms:

*Screen, views và pages trong ứng dụng di động được tạo thành từ một số nhóm thành phần cơ bản: pages, layouts và controls. Pages có thể là màn hình đầy đủ hoặc một phần hoặc nhóm controls. Ở Xamarin.Forms chúng được gọi là các trang vì chúng xuất phát từ lớp Pages. Trong iOS, chúng là views; và trong Android, screens, layout hoặc đôi khi được gọi một cách lỏng lẻo là các activities.*

*Controls là các thành phần UI riêng lẻ mà chúng tôi sử dụng để hiển thị thông tin hoặc cung cấp lựa chọn hoặc điều hướng. Xamarin.Forms gọi các views, bởi vì được kế thừa từ lớp View. Một số control được gọi là widget trong Android. Thông tin thêm về những điều này trong Chương 5.*

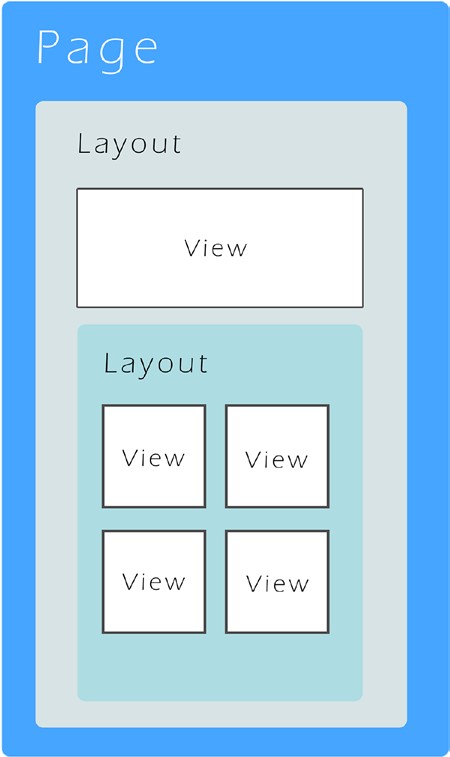
*Layouts* *là các thùng chứa cho các điều khiển xác định kích thước, vị trí và mối quan hệ của chúng với nhau. Xamarin.Forms và Android sử dụng thuật ngữ này, trong khi ở iOS, mọi thứ đều là views. Thêm về những điều này trong Chương 3.*

*Lists*, *thường có thể cuộn và có thể lựa chọn, là một trong những công cụ lựa chọn và hiển thị dữ liệu quan trọng nhất trong giao diện người dùng di động. Thêm về những điều này trong Chương 6.*

*Navigation* *cung cấp cho người dùng cách truy cập ứng dụng bằng cách sử dụng các menu, tab, thanh công cụ, danh sách, biểu tượng có thể gõ và các nút lên và xuống. Thêm về điều này trong Chương 7.*

*Modals, dialog boxes, and alerts* *thường là các màn hình bật lên cung cấp thông tin và yêu cầu một số phản hồi từ người dùng. Thêm về những điều này trong Chương 7.*

# Sử dụng giao diện Xamarin.Forms

Các Pages, Layouts và views tạo nên cốt lõi của Xamarin.Forms UI (Hình 1-5). Các Page là vùng chứa chính và mỗi màn hình được điền bởi một lớp Page duy nhất. Một page có thể chứa các biến thể của layout, sau đó có thể giữ các layout khác, được sử dụng để đặt và định cỡ nội dung của chúng. Mục đích của các page và layout là để chứa và trình bày các view, đó là các điều khiển được kế thừa từ lớp view.

***Figure 1-5.*** *Page, layouts, and views on a Xamarin.Forms screen*

Building apps using Xamarin

## Page

Lớp Page là thùng chứa chính của mỗi màn hình chính trong ứng dụng. Bắt nguồn từ Xamarin.Forms.VisualEuity, Page là lớp cơ sở để tạo các lớp UI cấp cao nhất khác. Dưới đây là các loại page chính:

* ContentPage
* MasterDetailPage
* NavigationPage
* TabbedPage
* CarouselPage

Ngoài vai trò là nơi chứa các layout và view, các page 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. Thêm về những điều này trong Chương 7.

## Layout

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

* StackLayout
* FlexLayout
* Grid
* AbsoluteLayout
* RelativeLayout
* ScrollView
* Frame
* ContentView

Các thuộc tính Content và Children của layout chứa các layout và views khác. Căn chỉnh ngang và dọc được thiết lập bởi các thuộc tính HorizontalOptions và VerticalOptions. Hàng, cột và ô trong bố cục có thể được đệm bằng khoảng cách, có kích thước để mở rộng để lấp đầy không gian có sẵn hoặc thu nhỏ để phù hợp với nội dung của chúng. Thêm về bố trí trong chương 3.

**Tip** Bố cục Xamarin.Forms có nguồn gốc từ lớp View, vì vậy mọi thứ có trong một trang thực sự là một dạng của view.

## View

Views are controls, the visible and interactive elements on a page. These range from the basic views like buttons, labels, and text boxes to the more advanced views like lists and navigation. Views contain properties that determine their content, font, color, and alignment. Horizontal and vertical alignment is set by properties HorizontalOptions and VerticalOptions. Like layouts, views can be padded with space, sized to expand to fill available space, or shrunk to fit their content. Later in this chapter, we’ll code some views, then visit them again in Chapter 5 and throughout the book. These are the primary views grouped by function:

Views là các điều khiển, các yếu tố hiển thị và tương tác trên một trang. Các phạm vi này từ các view cơ bản như nút, nhãn và hộp văn bản đến các view nâng cao hơn như danh sách và điều hướng. Views chứa các thuộc tính xác định nội dung, phông chữ, màu sắc và căn chỉnh của chúng. Căn chỉnh ngang và dọc được thiết lập bởi các thuộc tính HorizontalOptions và VerticalOptions. Giống như Layout, Views có thể được đệm bằng Khoảng trống, có kích thước để mở rộng để lấp đầy không gian có sẵn hoặc thu nhỏ để phù hợp với nội dung của chúng. Sau đó trong chương này, chúng ta sẽ code một số views, sau đó tìm hiểu sâu hơn chúng trong Chương 5 và trong suốt cuốn sách. Đây là các view 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

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

# Tạo app với Xamarin.Forms

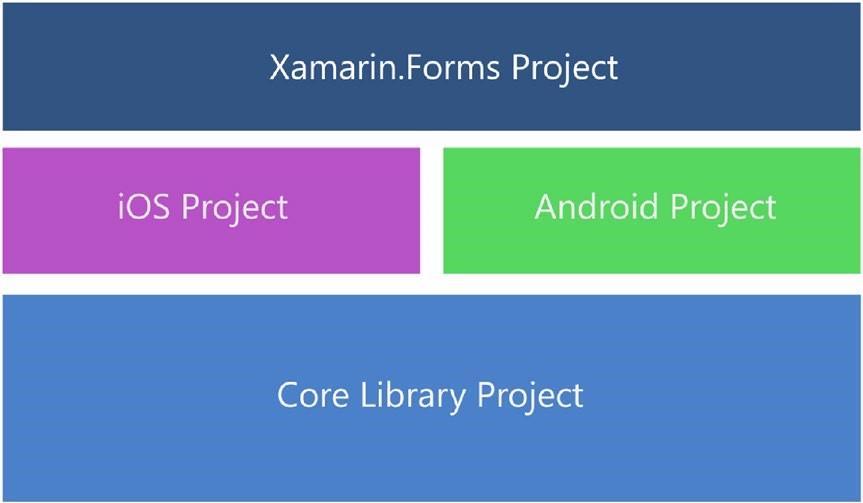
Xamarin cung cấp các template có chứa các dự án cần thiết để tạo một ứng dụng với Xamarin.Forms. Một ứng dụng đa nền tảng thường chứa các dự án nhỏ sau:

*Xamarin.Forms*: code UI đa nền tảng được gọi bởi một trong các dự án dành riêng cho các nền tảng. Điều này có thể được thực hiện bằng .NET Standard, mặc dù để tương thích ngược, Thư viện lớp di động (PCL) và dự án chia sẻ cũng có sẵn. Ví dụ chúng tôi sẽ tạo trong chương này sử dụng .NET Standard.

*Xamarin.Android*: Code dành riêng cho android

*Xamarin.iOS*: Code dành riêng cho iOS

*Core Library*: Logic ứng dụng được chia sẻ, chẳng hạn như logic nghiệp vụ và lớp truy cập dữ liệu bằng .NET Standard, PCL hoặc dự án dùng chung.



## *Figure 1-6. Xamarin.Forms solution*

**Tip** dự án thư viện Core không được thêm bởi các mẫu giải pháp và phải được tạo thủ công, dưới dạng dự án tiêu chuẩn .net hoặc dự án chia sẻ. nếu bạn mới bắt đầu sử dụng Xamarin.Forms, bạn có thể bỏ qua thư viện Core ngay bây giờ và đặt tất cả các tệp được chia sẻ của bạn vào dự án Xamarin.Forms.

Hãy cùng với nhau tạo ra một ứng dụng demo đơn giản để giúp chúng tôi khám phá nền tảng của Xamarin.Forms và nhiều tính năng thường được sử dụng.

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.

Điều này sẽ tạo ra nhiều dự án: một cho Xamarin.Forms và sau đó là các dự án dành riêng cho nền tảng bao gồm Android và iOS. Các dự án dành riêng cho nền tảng có sẵn tùy thuộc vào việc bạn sử dụng PC hay Mac, cho dù bạn là ứng dụng 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 iOS 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 is free with a Visual studio license, and Visual studio Community edition is free.

## Dự án Xamarin.Forms

Khi sử dụng Visual Studio, dự án Xamarin.Forms chứa App.cs (Listing 1-1), định nghĩa và trả về trang chính của ứng dụng. Đối tượng Application đóng vai trò là lớp cơ sở của Ứng dụng và cung cấp thuộc tính MainPage 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

}

Building apps using Xamarin

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 Application được chia sẻ và hiển thị nó làm triển khai gốc. Code mặc định đặt thuộc tính MainPage trong hàm tạo của nó trong trường hợp này là một đối tượng ContentPage có tên MainPage. Chúng tôi sẽ sớm thay thế MainPage bằng lớp ContentPage của riêng mình và đặt các điều khiển trên đó bằng XAML.

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

The OnStart, OnSleep, and OnResume method overrides created for us are used to manage our app when it is moved to and from the background.

### Vòng đời ứng dụng: OnStart, OnSleep, and OnResume

Khi người dùng nhấn Back or Home (or App Switcher) buttons on their device, Ứng dụng chuyển về trạng thái chạy nền. Khi người dùng mở lại app, ứng dụng tiếp tục chạy ở chế độ chính.Đây gọi là vòng đời ứng dụng. Lớp application có 3 phương thức để xử lý lifecycle:

* **OnStart**—Gọi khi app mới khởi động. Hữu ích để tải các giá trị vào bộ nhớ mà ứng dụng cần.
* **OnSleep**—Gọi khi app chuyển vào trạng thái chạy nền.

Hữu ích cho dọn dẹp và bắt đầu các tác vụ nền.

* **OnResume**—Gọi khi app từ chạy nền sang chạy chính. Hữu ích cho tải lại bộ nhớ và trả về các dữ liệu từ luồng chạy 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 trở lại từ trạng thái đó.

### Tạo Pages sử dụng ContentPage

Thuộc tính MainPage trong App.cs (Listing 1-1) quy định trang chính trong Xamarin. Forms: MainPage. mã XAML của MainPage is shown in Listing 1-2. Nó chứa 1 layout là StackLayout và 1 view hoặc control là Label.

***Listing 1-2.*** MainPage.xaml in a New Xamarin.Forms XAML 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" 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 listing 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. Thuộc tính Padding tạo không gian xung quanh 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. Thuộc tính Title chứa văn bản và thuộc tính Icon chứa hình ảnh được hiển thị ở đầu trang khi NavigationPage được triển khai. Các phương thức vòng đời OnAppear và OnDisappear có thể được ghi đè để xử lý việc Hiển thị và Kết thúc của ContentPage. Thuộc tính ToolBarItems rất hữu ích để tạo menu thả xuống. Tất cả các thành viên liên quan đến điều hướng này được đề cập trong Chương 7.

## Xamarin.Android

Dự án Android chứa tệp khởi động có tên MainActivity.cs, định nghĩa một lớp hoạt động được kế thừa từ Xamarin.Forms.Platform.Android. FormsApplicationActivity như đã thấy trong listing 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());

}

}

}

Ở phương thức OnCreate, Xamarin.Forms được khởi tạo và LoadApplication tạo App là current Application.

## Xamarin.iOS

Dự án iOS chứa một tệp khởi động có tên là AppDelegate (listing 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 Init() method và LoadApplication đặt App là page hiện tại.

## Tất cả các trình khởi tạo dành riêng cho nền tảng của chúng tôi, Android MainActivity và iOS AppDelegate, đều nhận được trang bắt đầu từ lớp Ứng dụng Xamarin.Forms, theo mặc định, trả về một trang thử nghiệm đơn giản.

## Core Library

Core library là một dự án trong Xamarin.Forms dành cho xử lý hoặc truy cập dữ liệu của một ứng dụng, phần lớn độc lập với nền tảng. Mặc dù không được tạo rõ ràng như là một phần của các template Xamarin.Forms, một dự án Core Library là một tiêu chuẩn. Nơi 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 PCL. Đây là nơi dành cho Code không phải giao diện người dùnghoặc back-end độc lập với nền tảng. Nó được tham chiếu bởi bất kỳ hoặc tất cả các dự án khác trong giải pháp. Sử dụng nó để tối ưu hóa việc tái sử dụng mã và tách rời các dự án UI khỏi lớp truy cập dữ liệu và logic nghiệp vụ. Lưu ý Core Library là một kiến trúc giải pháp tiên tiến. Nếu bạn chỉ 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 và ngừng sử dụng Core Library ngay bây giờ.

Now we need to build out the pages of our app. Time to code!

## Thiết lập Trang chủ của app

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 dự án của bạn và chọn ContentPage. 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. Tại đây, tập tin 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" />

Building apps using Xamarin

</StackLayout>

</ContentPage.Content>

</ContentPage>

Đây là code C# trong file ContentPageExample.cs:

namespace FormsExample

{

public partial class ContentPageExample : ContentPage

{

public ContentPageExample ()

{

InitializeComponent ();

}

}

}

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

namespace FormsExample

{

public class App : Application

{

public App()

{

MainPage = new ContentPageExample();

}

Now we have the custom page class ready and can load up our ContentPageExample XAML file with controls.

# Thêm Xamarin.Forms Views

View là thuật ngữ để chỉ các control 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 nhãn hoặc nút. Từ thời điểm này, chúng tôi sẽ sử dụng các thuật ngữ views và control thay thế cho nhau.

**Tip** all example code solutions can be found under the title of this book on <https://www.apress.com/us/book/9781484240298>in the source Code/ downloads tab, or on github at [https://github.com/danhermes/xamarinxaml-book-examples](https://github.com/danhermes/xamarin-xaml-book-examples).

Let’s start simply and put some views into ContentPageExample.xaml.

## Nhãn - Label

Labels Hiển thị dòng hay nhiều dòng văn bản. 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 *implicitly* khi đủ chữ xuất hiện trên vùng hiển thị, hoặc *explicitly* với ngắt dòng

A Label view có 2 loại căn chỉnh, view-justification và text-justification. Toàn view được căn chỉnh bằng HorizonalOptions và VerticalOptions. Văn bản trong label được căn chỉnh với HorizontalTextAlignment và VerticallTextAlignment

HorizontalTextAlignment = "End"

The TextAlignment có 3 giá trị: Start, Center, and End.

## Thêm Views sử dụng StackLayout

Một Layout hoạt động như một thùng chứa cho các views khác. Vì ContentPage có thể chỉ có một layout hoặc view, nên tất cả các view trên trang của chúng tôi phải được đặt trong một vùng chứa duy nhất được gán cho thuộc tính ContentPage. Ở đây, chúng tôi sử dụng StackLayout, một lớp con của Layout có thể xếp chồng các view con xếp chồng theo chiều 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 view con vào view 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.

**Note** Các view con của StackLayout được đặt theo chiều dọc trừ khi thứ tự ngang được chỉ định bằng Orientation = " Horizontal".

Compile and run the code. Figure 1-7 shows our labels on the StackLayout for iOS and Android, respectively.



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

## Nếu 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.

## Màu nền và màu chữ

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

Nếu bạn đang làm việc trên một dự án iOS và muốn tác phẩm của mình trông giống như 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"

## Sử dụng Fonts

Định dạng văn bản với các thuộc tính:

*FontFamily*: Set tên font; FontFamily = "Courier".

*FontSize*: Kích thước 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 các giá trị NamedSize, chẳng hạn như NamedSize.Large, sử dụng các thành viên NamedSize Lớn, Trung bình, Nhỏ và Micro, ví dụ: FontSize = "Large".

*FontAttributes*: Các kiểu phông chữ như 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" None, Bold và Italic.

Multiple attributes are specified using an attribute string formatted as “[font-face],[attributes],[size]”.

**Tip** these text formatting properties can be also set up app-wide using styles, which is covered in Chapter 4.

## Dùng font cho từng platform

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>

**Tip** another way to declare the On tags in OnPlatform involves the Value parameter.

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

## Button

Nút trong Xamarin.Forms là hình chữ nhật và clickable

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

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

Thuộc tính Text chứa văn bản hiển thị trên nút. HorizontalOptions và verticalOptions (được thảo luận trong phần tiếp theo) xác định sự điều chỉnh và kích thước căn chỉnh điều khiển. Cài đặt phông chữ NamedSize này làm cho phông chữ Lớn.

**Tip** Các nút có thể được tùy chỉnh bằng các thuộc tính BorderColor, BorderWidth, BorderRadius và TextColor. BorderWidth được mặc định là 0 trên iOs.

Thêm nút vào StackLayout.

<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.

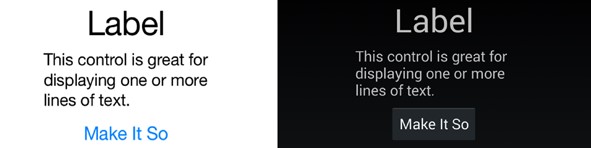
</Label.Text>

</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 để cùng nhau gán một trình xử lý sự kiện trong ContentPageExample.cs, inline:

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

{

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

};

Hoặc bằng tạo method:

button.Clicked += OnButtonClicked; …

được gọi bên ngoài trình xây dựng trang:

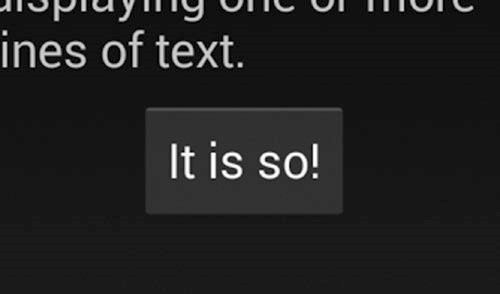
void OnButtonClicked(object sender, EventArgs e)

{

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

};

When you click the button, the button text changes, as in Figure 1-9.



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

**Tip** BorderWidth chỉ định độ đậm của đường vẽ nút.

## Thiết lập căn chỉnh và kích cỡ View: HorizontalOptions và VerticalOptions

Căn chỉnh ngang và dọc được điều chỉnh với HorizontalOptions và VerticalOptions properties:

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

### bố cục view là không gian được cung cấp cho view theo layout và các yếu tố xung quanh, không gian đệm xung quanh view và kích thước của view. Các loại định dạng này được thực hiện bằng cách sử dụng LayoutOptions và AndExpand.

### Căn chỉnh với LayoutOptions

Bố cục điều khiển riêng được xác định dọc theo một trục đơn bằng cách đặt thuộc tính HorizontalOptions hoặc VerticalOptions thành một trong các lớp LayoutOptions:

* Start: Trái hoặc trên của control (depending upon layout Orientation).
* Center: centers the control.
* End: Phải hoặc dưới của control.
* Fill: expands the size of the control to fill the space provided.

For example:

<Button HorizontalOptions = "Start" />

### AndExpand Pads với khoảng trắng

Thiết lập HorizontalOptions hoặc VerticalOptions trong LayoutOptions tạo ra các khoảng trống xung quanh views:

* StartAndExpand left or top-justifies the control and pads around the control with space.
* CenterAndExpand centers the control and pads around the control with space.
* EndAndExpand right or bottom-justifies the control and pads around the control with space.
* FillAndExpand expands the size of the control and pads around the control with space.

For example:

<Button HorizontalOptions = "StartAndExpand" />

**Tip** HorizontalOptions đặt là Fill và FillandExpand giống nhau với 1 control ở mỗi column

VerticalOptions Center hoặc Fill hữu ích chỉ khi khoảng trống dọc được cung cấp rõ ràng. Nếu ko, chẳng có tác dụng. LayoutOptions.Fill không làm cho control dài hơn nếu ko có khoảng trống.

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

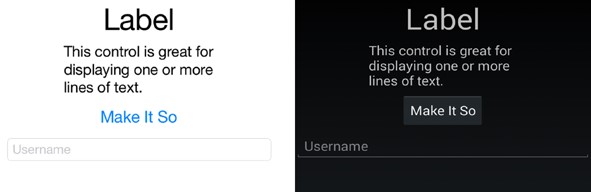
## Mục nhập văn bản

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

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

Building apps using Xamarin

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

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. Thuộc tính Bàn phím là thành viên của InputView và cung cấp một loạt các tùy chọn cho bàn phím trên màn hình xuất hiện cho đầu vào, bao gồm Văn bản, Số, Điện thoại, URL và Email. Hãy nhớ thêm mục vào StackLayout của bạn (xem Liệt kê 1-6 sau trong chương này). Hình 1-10 cho thấy điều khiển nhập mới cho tên người dùng.

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

**Tip** 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

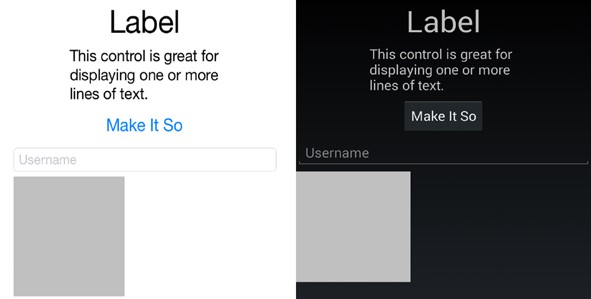
BoxView tạo ra một hình chữ nhật đồ họa có màu, hữu ích như một trình giữ chỗ mà sau này có thể được thay thế bằng một hình ảnh hoặc nhóm control phức tạp khác.

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

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

Thuộc tính Màu có thể được đặt thành bất kỳ giá trị thành viên Màu nào. Kích thước mặc định là 40 × 40 pixel, có thể thay đổi bằng các thuộc tính WidthRequest và HeightRequest.

**Tip** Hãy cẩn thận khi đặt HorizontalOptions và VerticalOptions thành Fill và FillAndExpand, vì điều này có thể ghi đè lên kích thước HeightRequest và WidthRequest của bạn.

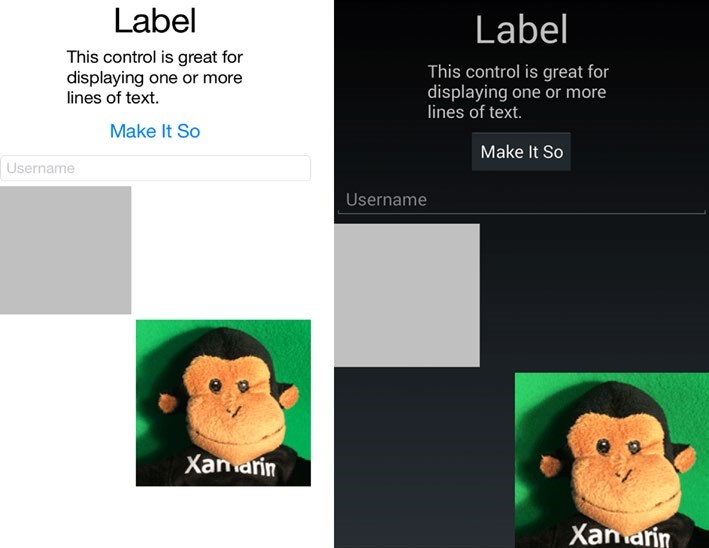


### *Figure 1-11. Xamarin.Forms BoxView*

## Hình ảnh

Chế độ xem hình ảnh giữ một hình ảnh để hiển thị trên Page từ một tệp cục bộ hoặc trực tuyến:

<Image Source="monkey.png" Aspect="AspectFit" HorizontalOptions="End" VerticalOptions="Fill" />



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

Let’s look at how an image is handled.

### 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. không sử dụng các ký tự đặc biệt trong tên tệp. Build Action phải được đặt thành Android Resource.

**iOS** 9 và mới hơn sử dụng Asset Lists và Image Sets có thể được thiết lập ở visual studio. Apple đã không cho phép tiếp cận thư mục /Resources nơi chúng ta 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**

Thuộc tính Image.Aspect xác định kích thước hình ảnh và được đặt bằng cách sử dụng bộ liệt kê Aspect. ví dụ:

Building apps using Xamarin

<Image Source="monkey.png" **Aspect="AspectFit"** HorizontalOptions="End" VerticalOptions="Fill" />

Các giá trị 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 không bị biến dạng và chừa khoảng trống 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.

### Làm cho hình ảnh có thể nhấn với GestureRecognizer

Hình ảnh và biểu tượng có thể chạm là phổ biến trong các ứng dụng di động cho các hành động và điều hướng. Giống như nhiều view trong Xamarin.Forms, Image không có một sự kiện nhấp hoặc nhấn và phải được kết nối bằng cách sử dụng lớp GestureRecognizer. Trình nhận dạng cử chỉ là một lớp có thể được thêm vào nhiều view để đáp ứng tương tác của người dùng. Nó hiện chỉ hỗ trợ các cử chỉ nhấn.

Add the standard gesture recognizer to the image.

<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 hình ảnh, Opacity thành 0,5 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;

}

**Tip** một cách triển khai thay thế của GestureRecognizer sử dụng thuộc tính Command:

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

Phản hồi 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 của ứng dụng. Một chạm, ví dụ, sẽ phản hồi cho người dùng 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. Hãy thực hiện điều đó một cách chuyên nghiệp bằng cách sử dụng thuộc tính Opacity nhưng thêm async / await để tạo một độ trễ nhỏ trong độ mờ 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 vài giây. Hãy nhớ thêm System.Threading.Tasks; vào 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 tốt.

**Tip** Để có hình ảnh động 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);

Building apps using Xamarin

Trong các dự án của riêng bạn, bạn sẽ sử dụng các nhận dạng cử chỉ (và async / await) để thực sự làm gì đó khi một hình ảnh được chạm. Nếu bạn muốn xem async / await trong ví dụ này, hãy nâng cấp Delay to 2000, sau đó nhấp vào nút “Make it so” trong khi nó chờ đợi và bạn sẽ thấy ứng dụng vẫn phản hồi. Bạn có thể làm nhiều thứ trong trình xử lý Chạm 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.

**Tip** Async / await là một kỹ thuật C # tiêu chuẩn để quản đồng thời. nhiều phương thức và chức năng Xamarin được cung cấp để xử lý nền bằng cách sử dụng async / await.

## ScrollView

ScrollView chứa một con duy nhất và cung cấp khả năng cuộn cho nội dung của nó: <ScrollView VerticalOptions="FillAndExpand">

Here we nest the StackLayout within this ScrollView, so our entire layout of views will now be scrollable.

<ScrollView VerticalOptions="FillAndExpand">

<StackLayout HeightRequest="1500">

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

</StackLayout>

</ScrollView>

**Tip** ScrollView cuộn theo chiều dọcmặc định nhưng cũng có thể cuộn sang một bên bằng thuộc tính Orientation. Ví dụ: Orientation = " Horizontal".

## Khoảng đệm xung quanh toàn trang

Thuộc tính Padding của ContentPage tạo không gian xung quanh toàn bộ trang. Ở đây:

<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>

This code will pad horizontal sides, left and right, and vertical sides, top and bottom:

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

Ví dụ này sẽ chia đều ở cả bốn phía:

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

Nếu bạn 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 đó ở phía trên màn hình cho các hệ điều hành khác. Phương thức OnPlatform cung cấp các giá trị hoặc hành động khác nhau tùy thuộc vào HĐH gốc (iOS, Android). Trong trường hợp này, thuộc tính Padding phụ thuộc vào nền tảng. <ContentPage.Padding>

<OnPlatform x:TypeArguments="Thickness">

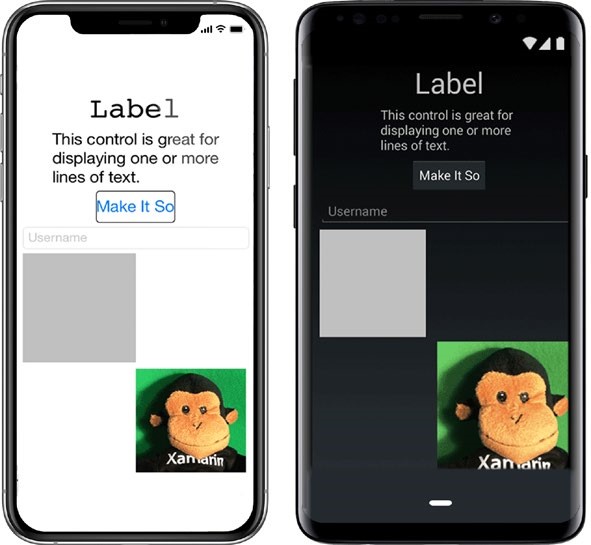
<On Platform="iOS" Value="10, 20, 10, 5"/>

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

</OnPlatform> </ContentPage.Padding>

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.

Figure 1-13 shows a final build and run on both platforms.



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

## Hoàn tất: Sử dụng Xamarin.Forms

Listings 1-6 and 1-7 provide the complete code for the added Xamarin.Forms views in the FormsExample solution. This listing contains the more recent form of OnPlatform.

***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>

Building apps using Xamarin

</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;

}

}

}

# Tổng kết

Xamarin.Forms cung cấp một điểm khởi đầu để phát triển giao diện người dùng ứng dụng di động đa nền tảng, có sự hỗ trợ đầy đủ với các Page, Layout và view có thể tùy chỉnh.

Một giải pháp Xamarin.Forms thường có một dự án riêng cho từng nền tảng sau: Android và iOS. Dự án Xamarin.Forms rất hữu ích cho các giao diện người dùng đa nền tảng và dự án Core Library chứa lớp truy cập dữ liệu và xử lý logic nghiệp vụ.

Các nhà phát triển sẽ lựa chọn Xamarin.Forms hay cách tiếp cận UI dành riêng cho nền tảng với Xamarin.Android và Xamarin.iOS

*View* trong Xamarin.Forms còn được coi là *control*, và chúng ta đã tìm hiểu qua về: Label, Entry, BoxView, Image, StackLayout, và ScrollView.

Xamarin.Forms XAML, giống các ngôn ngữ đánh dấu khác, đi kèm với bộ cấu trúc, quy tắc để hỗ trợ xây dựng giao diện người dùng.