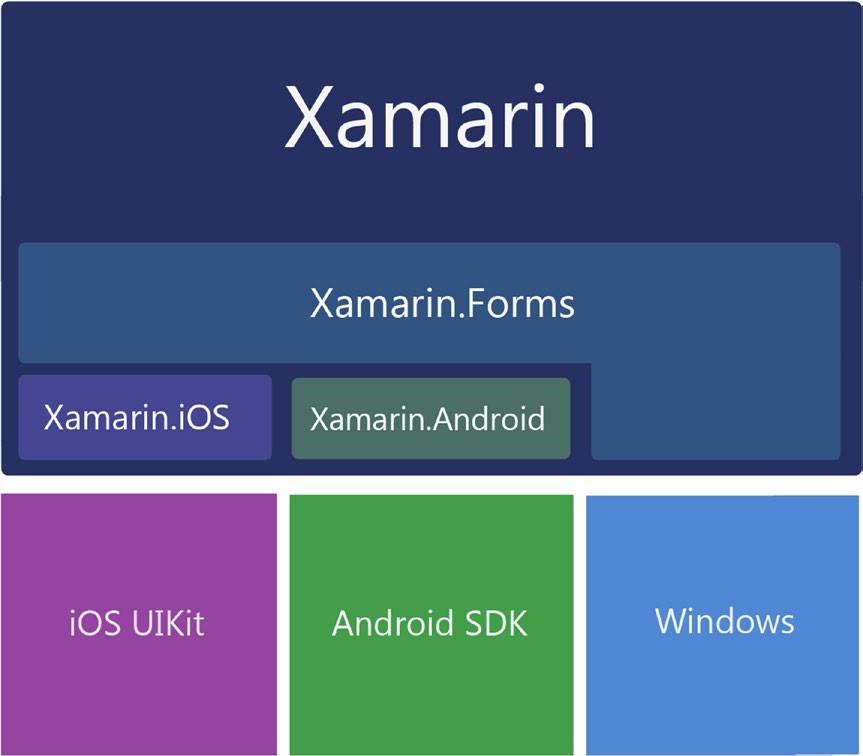
**CHƯƠNG 1   
Xây dựng ứng dụng bằng Xamarin**Xamarin đã trở thành đa nền tảng đến mức bây giờ bao gồm iOS, Android, Windows, macOS, Tizen, WPF, Hololens, GTK và các loại khác. Đó là rất nhiều nền tảng. Xamarin làm điều đó như thế nào? Xuất thân từ Dự án Mono mã nguồn mở đã mang .NET lên Linux, nền tảng 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 của tất cả bằng 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 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 amarin.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 bette r. 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. Chúng 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, chúng 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 hợp các thành phần UI đa nền tảng hiển thị trong mỗi ba hệ điều hành gốc này (xem Hình 1-1 ).   


***Hình 1-1.****Xamarin libra ries liên kết với các thư viện hệ điều hành gốc*

**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 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 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, chúng 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 chúng cho XAML.

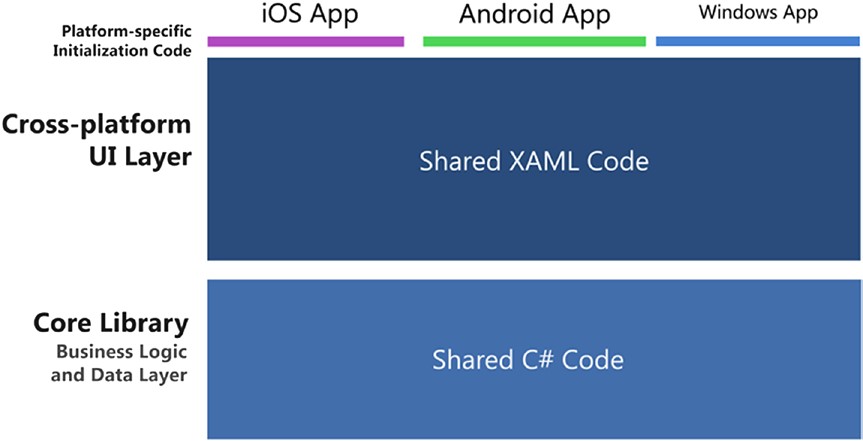
Các phần tử Xamarin.Forms được xây dựng bằng các lớp Trang , Bố cục và Xem . API này cung cấp một loạt các mẫu UI di động đa nền tảng tích hợp. Bắt đầu với các đối tượng Trang cấp cao nhất , nó cung cấp các trang menu quen thuộc như NavigationPage cho các menu khoan phân cấp, TabbedPage cho các menu tab, Mas terDetailPage để tạo ngăn kéo điều hướng, CarouselPage để cuộn các trang 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 chúng tôi có thể phát triển UI gốc, cơ bản cho cả ba nền tảng mà hầu như không có kiến ​​thức về UI và iOS. 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 iOS, 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!

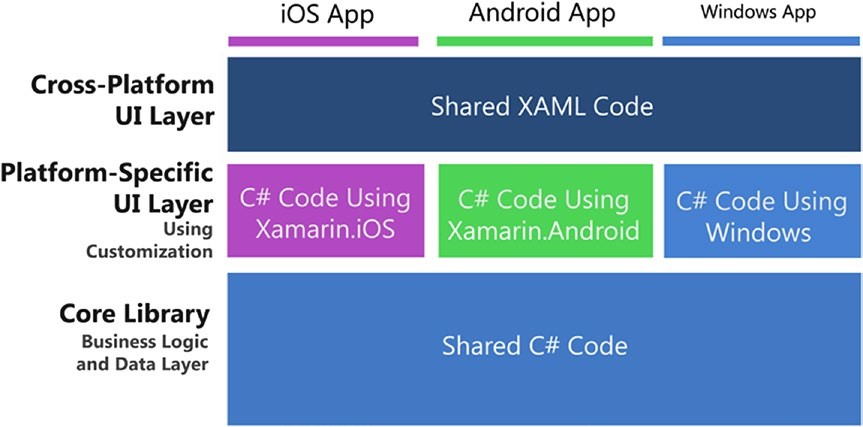
**Kiến trúc giải pháp Xamarin.Forms**

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 sử dụng lại, 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 cấp độ dưới cùng của sơ đồ. Điều này thường được gọi là Thư viện lõi. Lớp UI Xamarin.Forms dạng chéo cũng là C # và được mô tả là lớp giữa trong hình. Lớp mỏng, bị hỏng ở trên cùng là một lượng nhỏ mã C # UI dành riêng cho nền tảng trong các dự án dành riêng cho nền tảng cần có để khởi tạo và chạy ứng dụng trong mỗi hệ điều hành tự nhiên.



***Hình 1-2.****Kiến trúc giải pháp Xamarin.Forms: một ứng dụng cho nhiều nền tảng*

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 có thể, hữu ích và được khuyến khích. Nó có thể xảy ra ở một số cấp độ. Đầu tiên, trong các tùy chọn stomization Xamarin.Forms, bao gồm các trình kết xuất, hiệu ứng tùy chỉnh, quan điểm bản địa. 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 Xamar in.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.



***Hình 1-3.****Kiến trúc Xamarin.Forms với tùy biến*

**Lưu ý**Chương 8 cung cấp thêm về việc sử dụng mã tùy chỉnh và mã dành riêng cho nền tảng trong các giải pháp Xamari n.Forms.

Khi nào thì Xamarin.Forms thích hợp để sử dụng và khi nào chúng ta xem xét các lựa chọn Xamarin khác? Tôi sẽ giải quyết câu hỏi chính này một lát sau trong chương này, nhưng trước tiên hãy xác định các tùy chọn UI dành riêng cho nền tảng của Xamarin.

**Tìm hiểu về nền tảng cụ thể UI Approach**

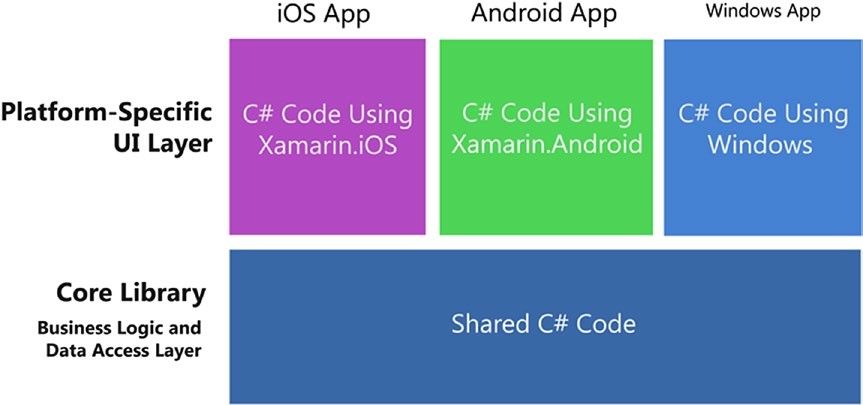
Trước Xamarin.Forms, đã có các tùy chọn UI dành riêng cho nền tảng (hoặc bản địa), bao gồm các thư viện Xamarin.Android, Xamarin.iOS và Windows Phone SDK. Xây dựng màn hình bằng cách sử dụng các giao diện người dùng dành riêng cho nền tảng đòi hỏi một số hiểu biết về các giao diện người dùng gốc được các thư viện này chỉnh sửa. Chúng tôi không cần mã trực tiếp trong iOS UIKit hoặc Android SDK, vì chúng tôi đã xóa một lớp khi sử dụng các ràng buộc Xamarin trong C #. Tất nhiên, sử dụng SDK Windows, chúng tôi đang mã hóa nguyên bản trong C # dựa trên HĐH Windows. Ưu điểm của việc sử dụng UI dành riêng cho nền tảng của Xa marin 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, và các ràng buộc Xamarin cho thấy nhiều trong số chúng ra khỏi hộp.

**Lưu ý**Chúng ta không nói về phát triển ui bản địa bằng Objective-C hoặc Java ở đây mà là sử dụng các ràng buộc dành riêng cho nền tảng Xamarin C # cho các thư viện ui bản địa. để tránh nhầm lẫn như vậy, cuốn sách này ủng hộ hạn *nền tảng cụ thể*trên *bản địa*khi đề cập đến các thư viện Xamarin, b nhà phát triển ut Xamarin đôi khi sẽ sử dụng thuật ngữ *mẹ đẻ*để chỉ việc sử dụng các nền tảng cụ thể các thư viện Xamarin.iOS và Xamarin.Android .

**Kiến trúc giải pháp UI dành riêng cho nền tảng**

Hình 1-4 cho thấy cách một giải pháp dành riêng cho nền tảng được thiết kế để chia sẻ nền tảng sros

Mã ứng dụng C # chứa lớp truy cập dữ liệu và logic nghiệp vụ, giống như một Dung dịch Xamarin.Forms. Lớp UI là một câu chuyện khác: tất cả đều dành riêng cho nền tảng. 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.



***Hình 1-4.****Kiến trúc giải pháp UI dành riêng cho nền tảng*

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?

**Chọn Xamarin.Forms hoặc UI dành riêng cho nền tảng**

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

*Tôi sử dụng cái nào, Xamarin.Forms hoặc UI dành riêng cho nền tảng Xamarin?*

Sự so sánh là dễ dàng và tính di động của Xamarin.Form s so với chức năng đầy đủ tính năng của các UI dành riêng cho nền tảng của Xamarin, cụ thể là Xamarin.Android và Xamarin.iOS. Các API Xamarin dành riêng cho nền tảng có nhiều tính năng hơn đáng kể so với Xamarin.Forms ngoài hộp, mặc dù tùy chỉnh sẽ đóng góp với một số công việc.

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 cho các mục sau:**

* *Học Xamarin*: Nếu bạn chưa quen với 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. Các hình thức (ví dụ: Android, iOS).
* *Ứng dụng kinh doanh*: Xamarin.Forms thực hiện những điều này chào đón 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.
* *Màn hình đa nền tảng đơn giản*: Xamarin.Forms là tuyệt vời để tạo màn hình cơ bản đầy đủ chức năng. Đối với các màn hình phức tạp hơn, tận dụng trình kết xuất tùy chỉnh Xamarin.Forms để biết chi tiết cụ thể về nền tảng.

**Sử dụng giao diện người dùng dành riêng cho nền tảng (Xamarin.iOS hoặc Xamarin.Android)**

* *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 giao diện người dùng cụ thể 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 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 ah igh. 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 ( trường hợp hiếm khi xảy ra ngay cả khi bạn bắt đầu với một nền tảng), hãy cân nhắc sử dụng UI dành riêng cho nền tảng.

Tuy nhiên, các nhà phát triển thông minh đang tạo ra các ứng dụng Biểu mẫu ngày càng tiên tiến hơn. Ngoài ra, nhóm phát triển Xamarin tại Microsoft di chuyển 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 cụ thể của nền tảng và giúp chúng tôi tăng quyền kiểm soát đối với giao diện người dùng đa nền tảng của chúng tôi. Ngoài ra, các dự án nguồn mở và các công cụ của bên thứ ba như UI của Telerik 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.

Khi Xamarin.Forms yêu cầu các tác vụ phức tạp hoặc thiết kế cao, hầu như mọi thứ đều có thể sử dụng tùy chỉnh.

**Sử dụng Trình kết xuất, Hiệu ứng và Chế độ xem gốc**

Cuối cùng bạn sẽ cần nhiều hơn từ Xamari n.Forms hơn là nó cung cấp cho bạn ra khỏi hộp. Khi Xamarin.Forms yêu cầu các tác vụ hoặc thiết kế phức tạp, hầu như mọi thứ đều có thể sử dụng tùy chỉnh Xamarin.Forms. Trình kết xuất tùy chỉnh cung cấp quyền truy cập vào các lớp g, trình kết xuất màn hình cụ thể ở cấp độ thấp hơn, được gọi là trình kết xuất, 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 Xamarin.Forms hoặc ap p và tùy chỉnh nó theo nền tảng bất cứ khi nào cần thiết. Thông tin thêm về điều này trong Chương 8 .

Trình kết xuất tùy chỉnh 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. Trình kết xuất tùy chỉnh là, tuy nhiên, pháo hạng nặng. Nếu bạn muốn một cái gì đó mang tính chiến thuật hơn, như chỉ đơn thuần là tùy chỉnh một thuộc tính trên điều khiển Xamarin.Forms, hãy xem xét hiệu ứng của Chiêu. Ngoài việc để lộ các thuộc tính, các hiệu ứng còn có khả năng truyền tham số cho các thuộc tính đó và xác định các sự kiện trên Xamari n.Forms điều khiển. Bạn truyền tham số cho hiệu ứng bằng các thuộc tính đính kèm hoặc Thời gian chạy ngôn ngữ chung (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 có 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, đó là nguồn năng lượng thô trong tay bạn khi bạn làm việc với bộ công cụ rm platfo .

Sử dụng tùy chỉnh một cách cẩn thận hoặc mạo hiểm một cơ sở mã UI bị phân mảnh mà có lẽ nên được viết hoàn toàn như một giao diện người dùng dành riêng cho nền tảng. Được sử dụng một cách thận trọng, tùy chỉnh có thể biến sản phẩm cơ bản, mờ nhạt của bạn thành một trang đa năng, độc đáo, phổ biến . Chúng ta hãy làm mới nhanh giao diện người dùng di động.

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

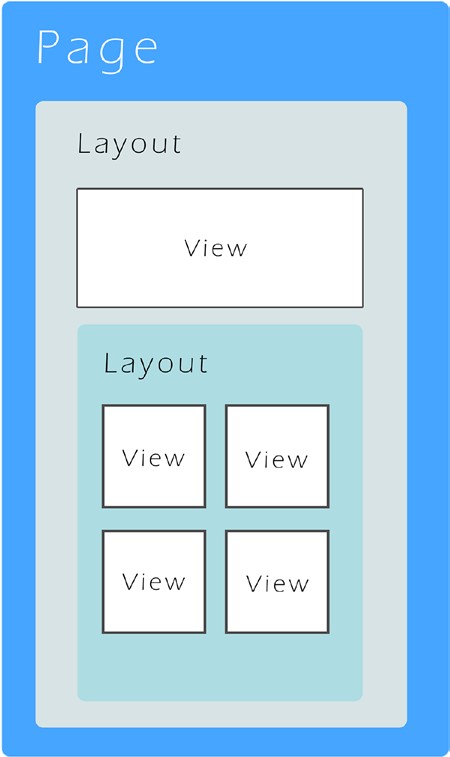
Xamarin là một công cụ hợp nhất phục vụ một số nền tảng, nhiều trong số đó có thể có các tên khác nhau cho cùng một thứ. Dưới đây là một số thuật ngữ thống nhất, được cân nhắc rất nhiều theo hướng của Xamarin.Forms:

* *Màn hình*, *quan điểm*, *và các trang*trong ứng dụng di động được tạo thành từ nhiều nhóm cơ bản của các thành phần: trang, bố cục và điều khiển. Các trang có thể là màn hình đầy đủ hoặc một phần hoặc nhóm điều khiển. Ở Xamarin. Các biểu mẫu, chúng được gọi là các trang vì chúng xuất phát từ lớp Trang . Trong iOS, họ không xem được; và trong Android, chúng là màn hình, bố cục hoặc đôi khi được gọi một cách lỏng lẻo là các hoạt động.
* *Điều khiển*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 khung nhìn này, bởi vì View là lớp điều khiển kế thừa từ đó. Một số điều khiển được gọi là widget trong Android. Thông tin thêm về những điều này trong Chương 5 .
* *Bố cục*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 có thuật ngữ này, trong khi ở iOS, mọi thứ đều là chế độ xem. Thêm về những điều này trong Chương 3 .
* *Danh sách*, 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 .
* *Điều hướng*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 .
* *Các phương thức, hộp thoại và cảnh báo*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 .

Bây giờ chúng ta có bối cảnh và một số thuật ngữ để làm việc, hãy bắt đầu với 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 (Hình 1-5 ). Các trang là vùng chứa chính và mỗi màn hình được điền bởi một lớp 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 .



***Hình 1-5.****Page, bố cục và chế độ xem trên màn hình Xamarin.Forms*

**Trang**

Lớp Trang 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 . Đây là các trang chính:

* ContentPage
* MasterDetailPage
* NavigationPage
* TabbedPage
* Băng chuyền

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 đúc sẵn với chức năng hữu ích bao gồm điều hướng và phản ứng cử chỉ. Thêm về những điều này trong Chương 7 .

**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 (lượt xem). Dưới đây là các bố cục chính:

* StackLayout
* FlexLayout
* Lưới
* Tuyệt đối
* RelativeLayout
* Cuộn
* Khung
* Xem nội dung

Các thuộc tính Nội dung và / hoặc Trẻ em của bố cục chứa các bố cục và chế độ xem khác. Căn chỉnh ngang và dọc được thiết lập bởi các thuộc tính ngang dọc và dọc . Hàng, cột và ô trong bố cục có thể được đệm bằng không gian, có kích thước để mở rộng để lấp đầy trần spa có sẵn hoặc thu nhỏ để phù hợp với nội dung của chúng. Thêm về bố trí trong chương 3 .

**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 lis ts và điều hướng. Chế độ xem 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 ngang dọc và dọc . Giống như bố cục, các khung nhìn có thể được đệm bằng không gian, có kích thước để mở rộng d để lấp đầy không gian có sẵn hoặc thu nhỏ để phù hợp với nội dung của chúng. Ở phần sau của chương này, chúng ta sẽ mã hóa một số khung nhìn, sau đó truy cập lại chúng trong Chương 5 và trong suốt cuốn sách. Đây là các khung nhìn chính được nhóm theo chức năng:

* Quan điểm cơ bản của cơ bản
* Nhãn
* Hình ảnh
* Nút
* BoxView
* Liệt kê Danh sách tạo một danh sách có thể cuộn, có thể lựa chọn
* ListView
* Tìm kiếm
* Nhập văn bản Mục nhập của người dùng trong chuỗi văn bản bằng bàn phím
* Nhập cảnh
* Biên tập viên
* Lựa chọn người dùng lựa chọn một loạt các lĩnh vực
* Bộ chọn
* DatePicker
* TimePicker
* Bước
* Thanh trượt
* Chuyển đổi
* Phản hồi của người dùng, thông báo cho người dùng về trạng thái xử lý ứng dụng
* ActivityTheicator
* ProgressBar
* Những người khác
* Bản đồ
* WebView

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

**Tạo một giải pháp Xamarin.Forms**

Xamarin cung cấp các mẫu có chứa các dự án cần thiết để tạo Xamarin. Hình thức ứng dụng. Một giải pháp đa nền tảng thường chứa các dự án này:

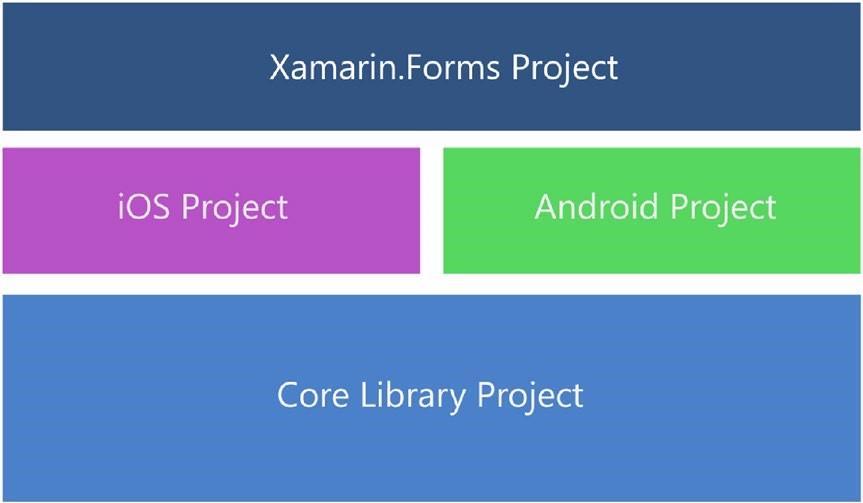
*Xamarin.Forms*: Mã UI đa nền tảng được gọi bởi một trong những dự án dành riêng cho nền tảng. Điều này có thể được hoàn thành bằng .NET Standard, mặc dù để tương thích ngược, Thư viện lớp di động (PCL) và dự án chia sẻ cũng có sẵn. Ví dụ chúng ta sẽ tạo trong chương này sử dụng .NET Standard.

*Xamarin.Android*: Mã dành riêng cho Android, bao gồm cả khởi động dự án Androi d.

*Xamarin.iOS*: Mã dành riêng cho iOS, bao gồm khởi động dự án iOS.

*Thư viện lõi*: 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 chia sẻ.

Hình 1-6 cho thấy các dự án chính thường được tìm thấy trong giải pháp Xamarin.Forms.



***Hình 1-6.****Dung dịch Xamarin.Forms*

**Mẹo**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 tạo 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.

Tạo một giải pháp Xamarin.Forms. Trong Visual Studio, tạo Dự án mới và chọn loại dự án Visual C # ➤ Đa nền tảng ➤ Mobile A pp (Xamarin.Forms). Trong Visual Studio cho Mac, tạo ra một giải pháp mới và chọn loại dự án đa nền tảng ➤ App ➤ Xamarin.Forms ➤ Forms Trống App. Đặt tên là FormsExample .

Điều này sẽ tạo ra nhiều dự án: một cho mã 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 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 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 miễn phí với giấy phép Visual studio và phiên bản Visual studio Community miễn phí.

Các phần sau đây cung cấp từng dự án trong giải pháp và mã mà chúng chứa.

**Dự án Xamarin.Forms**

Khi sử dụng Visual Studio, dự án Xamarin.Forms chứa App.cs (Liệt kê 1-1 ), 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 .

***Liệt kê 1-1.***Ứng dụng trong Dự án XAML mới.

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 hợp này là một đối tượng ContentPage có tên MainPage. Chúng tôi sẽ sớm thay thế MainPage bằng lớp ContentPage của riêng chúng tôi và điều khiển pl ace trên nó bằng XAML.

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

Các OnStart , OnSleep , và OnResume phương pháp ghi đè tạo ra cho chúng tôi được sử dụng để quản lý các ứng dụng của chúng tôi khi nó được chuyển đến và đi từ phía sau.

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

Khi người dùng 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.

• **OnSleep-**Tập tin mỗi khi ứng dụng được chuyển vào nền.

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

• **OnResume**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 trả về từ các luồng nền .

OnSleep ngủ 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.

**Mẹo**sử dụng từ điển Thuộc tính để duy trì đĩa trong các phương thức này khi ứng dụng được chạy nền. xem Chương 7 để biết thêm về quản lý nhà nước.

**Xây dựng trang bằng ContentPage**

Các bất động sản Mainpage trong App.cs (Bảng liệt kê 1-1 ) được gán các trang mặc định trong Xamarin. Hình thức: 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 Nhãn .

***Liệt kê 1-2.***MainPage.xaml trong Dự án XAML mới.

    <?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.

***Liệt kê 1-3.***MainPage.xaml.cs trong Dự án XAML mới.

public partial class MainPage : ContentPage

{

public MainPage()

{

InitializeComponent();

}

}

ContentPage có các thuộc tính ảnh hưởng đến sự xuất hiện của trang. Các Padding thuộc tính 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 Co ntentPage rất hữu ích cho việc điều hướng và quản lý nhà nước. Các đề 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 OnD isappear có thể được ghi đè để xử lý việc khởi tạo và hoàn thiện ContentPage . Các ToolBarItems tài sản là hữu ích cho việc tạo ra một trình đơn 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 một tệp khởi động có tên MainActivity.cs , định nghĩa một lớp hoạt động được kế thừa từ Xamarin.Forms.Pl platform.Android. FormsApplicationActivity như đã thấy trong Liệt kê 1-4 .

***Liệt kê 1-4.***MainActivity.cs trong Dự án FormsExample.Droid

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.Pl platform.iOS.FormsApplicationDelegate .

***Liệt kê 1-5.***AppDelegate.cs trong Dự án FormsExample.iOS

        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 lớp Ứng dụng Xamarin.Forms , theo mặc định, trả về trang thử nghiệm còn sơ khai.

**Thư viện lõi**

Thư viện lõi 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 Thư viện lõi 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 PCL. 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 trong giải pháp. Sử dụng nó để tối ưu hóa việc tái sử dụng mã và để tách rời UI projec ts khỏi lớp truy cập dữ liệu và logic nghiệp vụ. Lưu ý Thư viện lõi 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 và bây giờ không sử dụng Thư viện lõi.

Bây giờ chúng tôi cần xây dựng các trang của ứng dụng của chúng tôi. Thời gian để mã!

**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 Trang Nội dung. 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>

Đây là mã C # đằng sau được gọi là 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 Lượt xem**

*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 nhãn hoặc nút. Từ thời điểm này, chúng tôi sẽ sử dụng các thuật ngữ *xem*và *kiểm soát*thay thế cho nhau.

**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](https://translate.google.com/translate?hl=vi&prev=_t&sl=en&tl=vi&u=https://www.apress.com/us/book/9781484240298) trong tab Mã / tải xuống nguồn hoặc trên github tại [https://github.com/ danhermes / xamarin xaml-book- example](https://translate.google.com/translate?hl=vi&prev=_t&sl=en&tl=vi&u=https://github.com/danhermes/xamarin-xaml-book-examples) .

Hãy bắt đầu đơn giản và đặt một số lượt xem vào Content PageExample.xaml .

**Label View**

Label 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 . Nhãn văn bản được chứng minh trong một nhãn sử dụng  nhãn ‘s HorizontalTextAlignment và VerticallTextAlignment tài sản.

        HorizontalTextAlignment = "End"

Bảng liệt kê TextAlocation được gán cho các thuộc tính căn chỉnh này có ba giá trị: Bắt đầu , Trung tâm và Kết thúc .

Tiếp theo, các nhãn phải được chỉ định cho một bố cục cho vị trí trên trang. Trong ví dụ này, chúng 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 Bố cục hoạt động như một thùng chứa cho các khung nhìn khác. Sin ce một ContentPage có thể chỉ có một bố trí hoặc xem, tất cả các lượt xem trên trang của chúng tôi phải được đặt trong một khung chứa được gán cho các ContentPage sở hữu nội dung ‘s. Ở đây, chúng tôi sử dụng StackLayout , một lớp con của Bố cục có thể ngăn xếp ngăn xếp con xem ly 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 . WidthRequest 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 theo chiều dọc trừ khi thứ tự ngang được chỉ định bằng Định hướng = "Ngang" .

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

***Hình 1-7.****Xamarin.Forms Nhãn trên 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 có thể giúp bạn.

**Màu nền và màu phông chữ**

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 xem 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"

Chúng tôi sử dụng phông chữ trong nhiều điều khiển, vì vậy hãy làm một tổng quan nhanh về những điều đó.

**Sử dụng phông chữ**

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

* *FontFamily*: Đặt tên của phông chữ trong fontFamily bất động sản; 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 tôi mbers lớn , vừa , nhỏ , và Micro, f hoặc ví dụ, FontSize = "lớn" .
* *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 Không có, In đậm và In nghiêng .
* Nhiều thuộc tính được chỉ định bằng cách sử dụng một chuỗi thuộc tính được định dạng một s [[mặt chữ], [thuộc tính], [kích thước].

**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 OnPl Platform , 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 OnPl Platform 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 chữ bất động sản bao gồm các văn bản rõ ràng vào nút. VerticalOptions và verticalOptions (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 .

**Các**nút **Mẹo**có thể được tùy chỉnh bằng các thuộc tính 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.

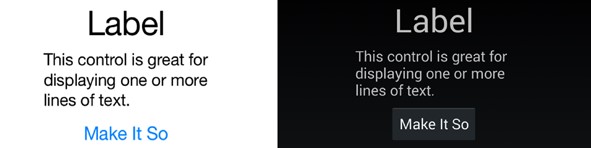
</Label.Text>

</Label>

**<Button Text = "Make It So" FontSize="Large" HorizontalOptions="Center"**  **VerticalOptions="Fill" Clicked="ButtonClicked" />**

</StackLayout>

Hình 1-8 cho thấy nút mới.



***Hình 1-8.****Nút Xamarin.Forms*

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

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

{

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

};

Hoặc bằng cách chỉ định một phương thức:

        button.Clicked += OnButtonClicked; Cấm được gọi bên ngoài trình xây dựng trang:

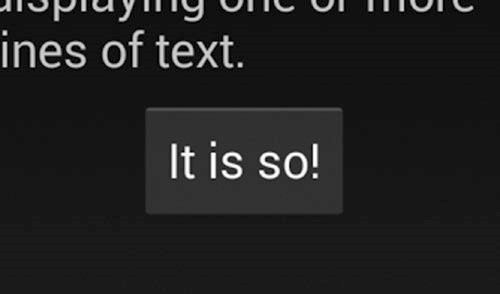
        void OnButtonClicked(object sender, EventArgs e)

{

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

};

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



***Hình 1-9.****nút. Sự kiện bị sa thải*

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

**Setting View Alignment and Size: HorizontalOptions and VerticalOptions**

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 alOptions và / hoặc VerticalOptions tính đến một giá trị của LayoutOptions lớp, ví dụ:

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

Cân nhắc trong bố cục chế độ xem là *không gian được cung cấp*cho chế độ xem theo bố cục 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 layo riêng lẻ được xác định dọc theo một trục đơn bằng cách đặt thuộc tính ngang dọc hoặc dọcOptions thành một trong các lớp LayoutOptions :

* Start left or top—justifies the control (depending upon layout Orientation).
* Center centers the control.
* End right or bottom—justifies the control.
* Fill expands the size of the control to fill the space provided.

For example:

<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.

Dọc được đặt thành Mở rộng và CenterAndExpand áp đặt không gian đệm xung quanh một điều khiển trong StackLayout .

Có nhiều ví dụ định dạng sau trong chương này và nhiều hơn nữa về chủ đề bố trí và căn chỉnh điều khiển trong Chương 3 . Tiếp theo hãy tạo một số đầu vào của người dùng.

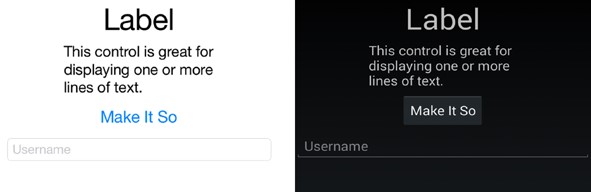
## Entry View for Text Input

Đoạn mã sau đây tạo một hộp văn bản để người dùng nhập một dòng văn bản đơn lẻ . Mục nhập 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 chương này). Hình 1-10 cho thấy mục tiếp theo mới cho tên người dùng.



***Hình 1-10.****Xamarin.Forms xem mục nhập người dùng*

**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 Màu trị viên. Các kích thước mặc định là 40 × 40 pixel, có thể được thay đổi u hát WidthRequest và HeightRequest tài sản.

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

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 .

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

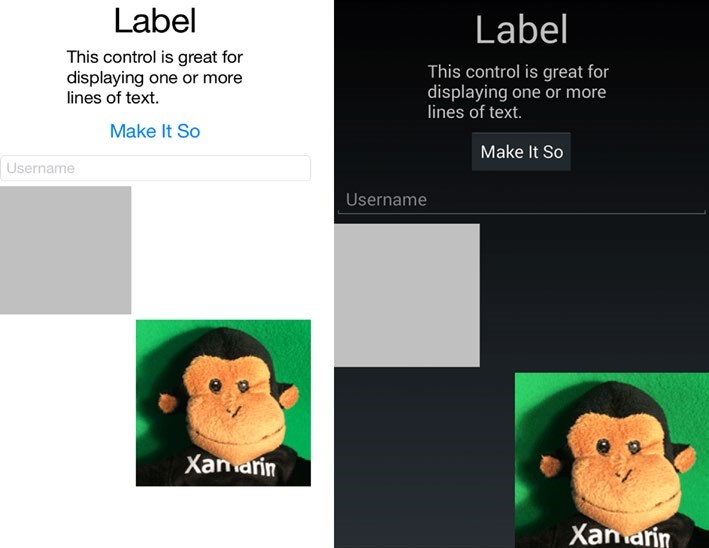
Cuối cùng, nhà thiết kế của bạn sẽ cung cấp cho bạn những biểu tượng đã hứa và bạn có thể thay thế BoxView của mình bằng hình ảnh thật.

## Image View

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

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

Hình 1-12 cho thấy hình ảnh con khỉ ở phía dưới bên phải.



***Hình 1-12.****Hình ảnh vie w*

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 Tài nguyên / có thể vẽ . 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 Danh sách tài sản và Bộ ảnh có thể được thiết lập trong Visual Studio trên dự án iOS. Apple đã không chấp nhận cách tiếp cận thư mục / Tài nguyên nơi chúng tôi sẽ tạo hình ảnh cho màn hình Retina với hậu tố @ 2x hoặc @ 3x trên tên tệp.

**Sizing: Aspect Image 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).
* **Lấp đầy**: 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.

**Making an Image Clickable with a GestureRecognizer**

Hình ảnh và biểu tượng có thể áp dụng là phổ biến trong các ứng dụng di động cho các hành động và điều hướng. Giống như nhiều chế độ xem Xamarin.Forms, tuổi Im không có 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 chế độ xem để đáp ứng tương tác của người dùng. Nó hiện chỉ hỗ trợ các cử chỉ nhấn. Các thuật ngữ *nhấp*và *ta p*được sử dụng thay thế cho nhau trong phát triển giao diện người dùng di động.

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.GestureRecognologists>

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

</Image.GestureRecognators>

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. Nhớ thêm bằng System.Threading.T Nhiệm vụ; 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 Make So So 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 (tpl). nhiều phương pháp Xamarin và func tions 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 một đứa trẻ độc thân và kiến thức sâu rộng scrollability đến nội dung của nó:

    <ScrollView VerticalOptions="FillAndExpand">

Ở đây chúng ta lồng S tackLayout 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 tính Định hướng. Ví dụ: Định hướng = "Ngang".

Đó là cho các lượt xem trên trang này. Cảm ứng cuối cùng sẽ được đệm xung quanh toàn bộ trang, vì vậy 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 công tài sản:

         <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 phương 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">

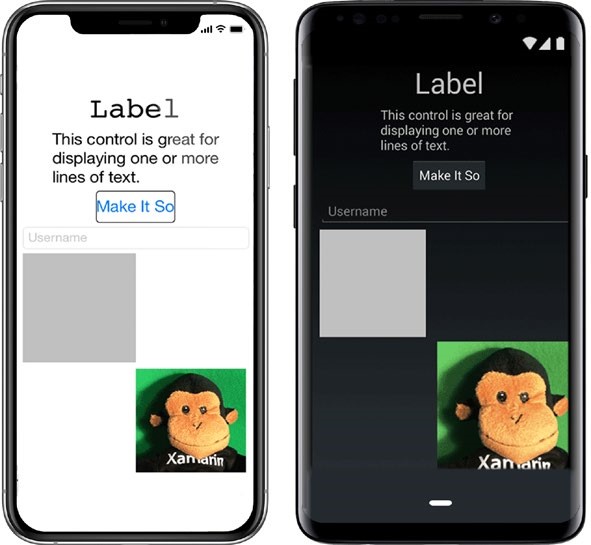
<On Platform="iOS" Value="10, 20, 10, 5"/>

<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 plat .



***Hình 1-13.****Bản dựng cuối cùng và chạy giải pháp FormsExample*

**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 OnPl Platform gần đây hơn .

***Liệt kê 1-6.***ContentPageExample.xaml trong Dự án FormsExample

<?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>

***Liệt kê 1-7.***ContentPageExample.xaml.cs trong Dự án FormsExample

    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;

}

}

}

# Summary

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 với 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 n.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 Xam arin.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.

*Xem*là thuật ngữ Xamarin.Forms cho *kiểm soát*, và chúng tôi delved vào một vài trong số các quan điểm thường xuyên nhất được sử dụng: Nhãn , nhập cảnh , BoxView , hình ảnh , 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!