

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH

KHOA ĐÀO TẠO CHẤT LƯỢNG CAO



BÁO CÁO ĐỒ ÁN

TÌM HIỂU VỀ XAMARIN VÀ VIẾT ỨNG DỤNG MINH HỌA

Môn học: Đồ án công nghệ thông tin

Mã môn học: PROJ215879_22_1_06CLC

Giảng viên hướng dẫn: Ths.Nguyễn Minh Đạo

Nhóm sinh viên thực hiện:



Phạm Tấn Đạt – 19110184

Phạm Viết Tiên – 20110571

Tp. Hồ Chí Minh, 7 tháng 12 năm 2022

PHIẾU NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

Họ và tên Sinh viên 1 :.....MSSV 1:

Họ và tên Sinh viên 2 :.....MSSV 2:

Ngành: Công nghệ Thông tin

Tên đề tài:

Họ và tên Giáo viên hướng dẫn:

.....

NHẬN XÉT

1. Về nội dung đề tài khối lượng thực hiện:

.....

.....

2. Ưu điểm:

.....

.....

3. Khuyết điểm

.....

.....

4. Đánh giá loại :

5. Điểm :

TP. Hồ Chí Minh, ngày tháng năm 2022

Giáo viên hướng dẫn

(ký & ghi rõ họ tên)

MỤC LỤC

DANH MỤC HÌNH ẢNH	6
LỜI CẢM ƠN	8
CHƯƠNG 1: TỔNG QUAN VỀ XAMARIN	10
1.1. Xamarin là gì?	10
1.2. Cách phát triển ứng dụng di động thường được thấy ngày nay?	10
1.3. Ưu nhược điểm của việc phát triển ứng dụng di động theo cách truyền thống	11
1.4. Ưu nhược điểm của Xamarin	13
1.5. Một số đặc điểm của Xamarin	14
CHƯƠNG 2: CÀI ĐẶT VÀ SỬ DỤNG XAMARIN TRÊN VISUAL STUDIO	19
CÀI ĐẶT	19
TẠO PROJECT DEMO ĐẦU TIÊN	20
CHƯƠNG 3: TÌM HIỂU THÊM VỀ XAMARIN VÀ VÍ DỤ MINH HOẠ	23
3.1. Xamarin Android là gì?	23
3.2. Xamarin iOS là gì?	26
3.3. Xamarin Form là gì?	26
3.4. Demo xamarin Forms:	27
3.5. Môi trường phát triển Xamarin	30
3.6 Binding Value Converters trong Xamarin.Form	35
3.6.1. Binding Value Converters là gì?	35
3.6.2. Tại sao cần Binding Value Converters?	36
3.6.3. Làm thế nào để sử dụng Binding Value Converters?	37
3.7. Mô hình MVVM trong Xamarin.Form	40
3.7.1. MVVM là gì?	40
3.7.2. Cấu trúc thư mục trong MVVM	41
3.7.3. Ưu nhược điểm của mô hình MVVM	44
CHƯƠNG 4: ỨNG DỤNG APPTRANSPORT	45
4.1. Đề tài: Xây dựng ứng dụng APPTRANSPORT	45
4.1.1. Đặc tả đề tài	45
4.1.2. Mô hình sử dụng	47
4.2. Code	48

4.2.1. Models (Transports.cs)	48
4.2.2. Entity (CarData.cs)	48
4.2.3. Entity (BicycleData.cs).....	49
4.2.4. Entity (ElectricBicycleData.cs)	49
4.2.5. Entity (MotorcycleData.cs)	50
4.2.6. Entity (MotorVNData.cs)	50
4.2.7. Entity (PlaneData.cs)	51
4.2.8. Services (TransportSearchHandler).....	51
4.2.9. Views (Cars/Bicycles/ElectricBicycles/MotorVNs/Motorcycles/Plane-Page.xaml) ..	52
4.2.10. Views (Cars/Bicycles/ElectricBicycles/MotorVNs/Motorcycles/Plane-Page.cs).....	53
4.2.11. Views (Car/Bicycle/ElectricBicycle/MotorVN/Motorcycle/Plane-DetailPage.xaml)	54
4.2.12. Views (Car/Bicycle/ElectricBicycle/MotorVN/Motorcycle/Plane - DetailPage.xaml.cs)	55
4.2.13. App.xaml	56
4.2.13. AppShell.xaml	57
4.2.14. AppShell.xaml.cs.....	58
4.3. Chạy thử	58
CHƯƠNG 5: HƯỚNG DẪN SỬ DỤNG	65
CHƯƠNG 6: TỔNG KẾT	68
6.1. Ưu điểm	68
6.2. Nhược điểm	68
6.3. Ý tưởng phát triển	68
CHƯƠNG 7: TÀI LIỆU THAM KHẢO	69

DANH MỤC HÌNH ẢNH

Hình 1: Ứng dụng di động truyền thống.....	10
Hình 2: Phương pháp mới.....	11
Hình 3: Xamarin App.....	12
Hình 4: Window APIs.....	13
Hình 5: iOS - API	14
Hình 6: Android API.....	14
Hình 7: Cài đặt Xamarin	19
Hình 8: Cài đặt Xamarin 2	20
Hình 9: Tạo Project đầu tiên	21
Hình 10: Cấu trúc của Project Android.....	21
Hình 11: Activity_main.xml Android.....	22
Hình 12: Khởi động Android App trên Device.....	22
Hình 13: Libraries Android.....	23
Hình 14: App structure.....	24
Hình 15: Ứng dụng email về activity.....	24
Hình 16: UI Element trong Xamarin	25
Hình 17: View trong Xamarin	25
Hình 18: Layout trong Xamarin.....	26
Hình 19: Layout file ~.....	26
Hình 27: Shared App Logic	27
Hình 28: Xamarin cách tiếp cận.....	27
Hình 29: Tạo blank App	28
Hình 30: Cấu trúc của một Xamarin Forms.....	28
Hình 31: Chạy thử bản demo blank app	29
Hình 32: Chạy thử bản demo blank app 2	29
Hình 33: Môi trường phát triển Xamarin.....	30
Hình 34: Kiến trúc của ứng dụng Xamarin.Forms	30
Hình 35: Bên trong của 1 ứng dụng Xamarin Forms.....	31
Hình 36: Page Xamarin Forms	31
Hình 37: Views trong Xamarin Forms.....	32
Hình 38: Rõ hơn về các layout trong Xamarin	32
Hình 39: Tạo Mobile App trong Xamarin	33
Hình 40: Kiến trúc tương tự trong Xamarin Mobile App 1	33
Hình 41: Kiến trúc tương tự trong Xamarin Mobile App 2.....	34
Hình 42: Chạy Project trong Mobile Xamarin Forms	35
Hình 43: Binding trong XamarinForms.....	36
Hình 44: When dont use binding	36
Hình 45: When use binding	37
Hình 46: How to use binding?	37
Hình 47: Class in Binding 1	38
Hình 48: Class in Binding 2.....	39

Hình 49: Class in Binding 3.....	40
Hình 50: Mô hình MVVM.....	41
Hình 51: Cấu trúc thường thấy của MVVM	42
Hình 52: Cấu trúc của App Phương tiện giao thông	47
Hình 53: Model Transports.cs.....	48
Hình 54: Entity CarData.cs	48
Hình 55: Entity BicycleData.cs.....	49
Hình 56: Entity ElectricBicycleData.cs	49
Hình 57: Entity MotorcycleData.cs	50
Hình 58: Entity MotorVNData.cs	50
Hình 59: Entity PlaneData.cs.....	51
Hình 60: Services TransportSearchHandler.cs	51
Hình 61 Views - xxxPage.xaml	52
Hình 62: Views - Page.cs.....	53
Hình 63: Views DetailPage.xaml.....	54
Hình 64 Views DetailPage.xaml.cs	55
Hình 65: App.xaml.....	57
Hình 66: AppShell.xaml	57
Hình 67: AppShell.xaml.cs	58
Hình 68: Mở file download.....	65
Hình 69: Chạy file và kết quả	66

LỜI CẢM ƠN

Để hoàn thành tốt đề tài và bài báo cáo này, chúng em xin gửi lời cảm ơn chân thành và sâu sắc nhất đến giảng viên, TS. Nguyễn Minh Đạo, người đã trực tiếp hỗ trợ chúng em trong suốt quá trình làm đề tài. Cảm ơn thầy đã đưa ra những lời khuyên từ kinh nghiệm thực tiễn của mình để định hướng cho chúng em đi đúng với yêu cầu của đề tài đã chọn, luôn giải đáp thắc mắc và đưa ra những góp ý, chỉnh sửa kịp thời giúp chúng em khắc phục nhược điểm và hoàn thành tốt cũng như đúng thời hạn đã đề ra.

Chúng em cũng xin gửi lời cảm ơn chân thành các quý thầy cô trong khoa Đào tạo Chất Lượng Cao nói chung và ngành Công Nghệ Thông Tin nói riêng đã tận tình truyền đạt những kiến thức cần thiết giúp chúng em có nền tảng để làm nên đề tài này, đã tạo điều kiện để chúng em có thể tìm hiểu và thực hiện tốt đề tài. Bên cạnh đó, chúng em xin được gửi cảm ơn đến các bạn cùng khóa và các anh chị khóa trên đã cung cấp nhiều thông tin và kiến thức hữu ích giúp chúng em có thể hoàn thiện hơn đề tài của mình.

Đề tài và bài báo cáo được chúng em thực hiện trong khoảng thời gian ngắn, với những kiến thức còn hạn chế cùng nhiều hạn chế khác về mặt kỹ thuật và kinh nghiệm trong việc thực hiện một đồ án phần mềm. Do đó, trong quá trình làm nên đề tài có những thiếu sót là điều không thể tránh khỏi nên chúng em rất mong nhận được những ý kiến đóng góp quý báu của các quý thầy cô để kiến thức của chúng em được hoàn thiện hơn và chúng em có thể làm tốt hơn nữa trong những lần sau.

Cuối cùng, chúng em xin kính chúc quý thầy, quý cô luôn dồi dào sức khỏe và thành công hơn nữa trong sự nghiệp trồng người. Một lần nữa chúng em xin chân thành cảm ơn.

Tp. Hồ Chí Minh, ngày 7 tháng 12 năm 2022

Nhóm sinh viên thực hiện

Tiên

Đạt

Phạm Viết Tiên

Phạm Tấn Đạt

CHƯƠNG 1: TỔNG QUAN VỀ XAMARIN

* Tìm hiểu chung:

- Theo một số thống kê, Smartphone đang phát triển rất nhanh và ngày càng tiến bộ, trong giai đoạn vừa qua cho thấy Smartphone đang phát triển nhanh gấp 10 lần máy tính PC và nhanh hơn PC. Gấp 2 lần khi internet bùng nổ và gấp 3 lần khi mạng xã hội bùng nổ. Rõ ràng, ngày nay chúng ta thấy điện thoại thông minh ở khắp mọi nơi, với vô số thương hiệu điện thoại thông minh, ... và được cung cấp bởi nhiều hãng, chẳng hạn như Android, IOS, ...

1.1. Xamarin là gì?

- **Xamarin** là nền tảng hỗ trợ phát triển ứng dụng di động trên Android, iOS và hỗ trợ phát triển ứng dụng đa nền tảng sử dụng ngôn ngữ lập trình C# .Net. Chỉ sử dụng một ngôn ngữ, bạn có thể tạo các ứng dụng gốc với hiệu suất tương đương với sử dụng Java hoặc ObjectiveC, Swift với các API được hỗ trợ đầy đủ 100%.

1.2. Cách phát triển ứng dụng di động thường được thấy ngày nay?

* Theo cách truyền thống:

- IOS app: Objective-C (Xcode)
- Android app: Java (Eclipse)
- Windows app: .NET/C# - HTML/JS – C++ (Visual Studio)



Hình 1: Ứng dụng di động truyền thống

1.3. Ưu nhược điểm của việc phát triển ứng dụng di động theo cách truyền thống

- Ưu điểm:

+ Tạo **native** app

+ Khai thác tối đa công cụ và OS

+ Khai thác tối đa hiệu suất

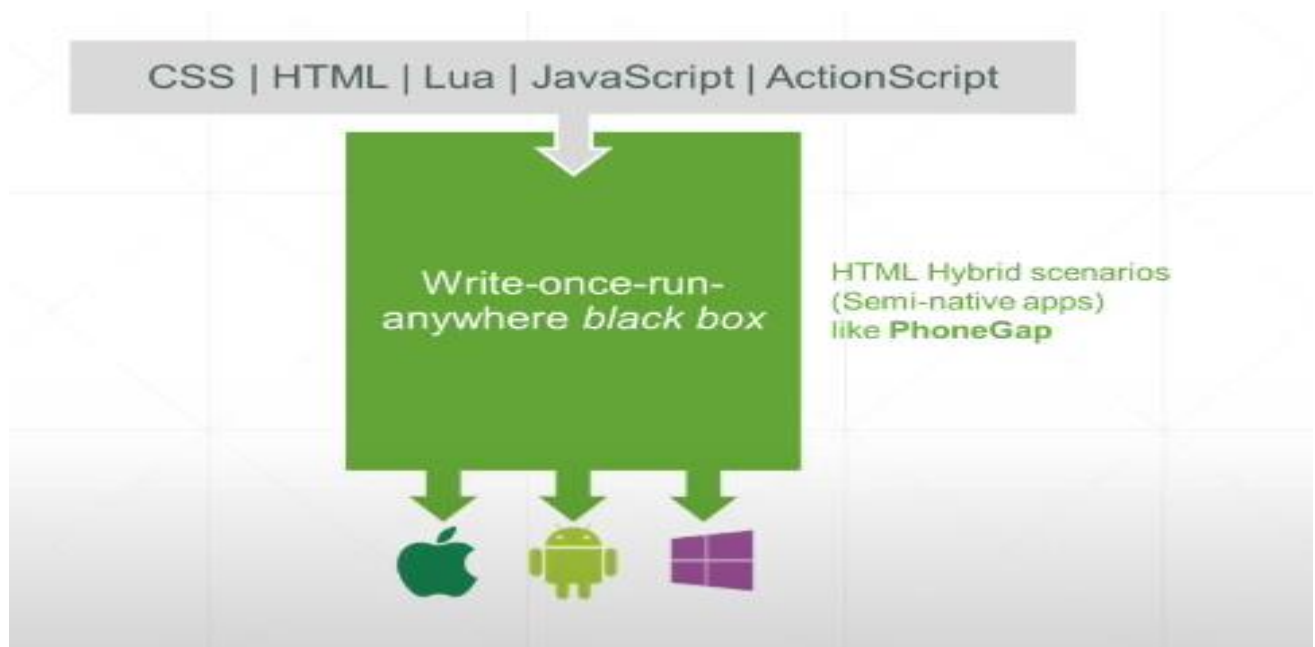
- Nhược điểm:

+ Khó khăn trong việc chia sẻ code (nếu được yêu cầu phát triển ứng dụng đa nền tảng)

+ Sử dụng nhiều tài nguyên.

➔ Nhận thấy được những khó khăn này, người ta đã ra đời 1 phương pháp

“write-once-run-anywhere” hay “hybird”.



Hình 2: Phương pháp mới

- Sau khi hybird ra đời có

- Ưu điểm :

+ Tạo ra được các ứng dụng đa nền tảng trong thời gian ngắn.

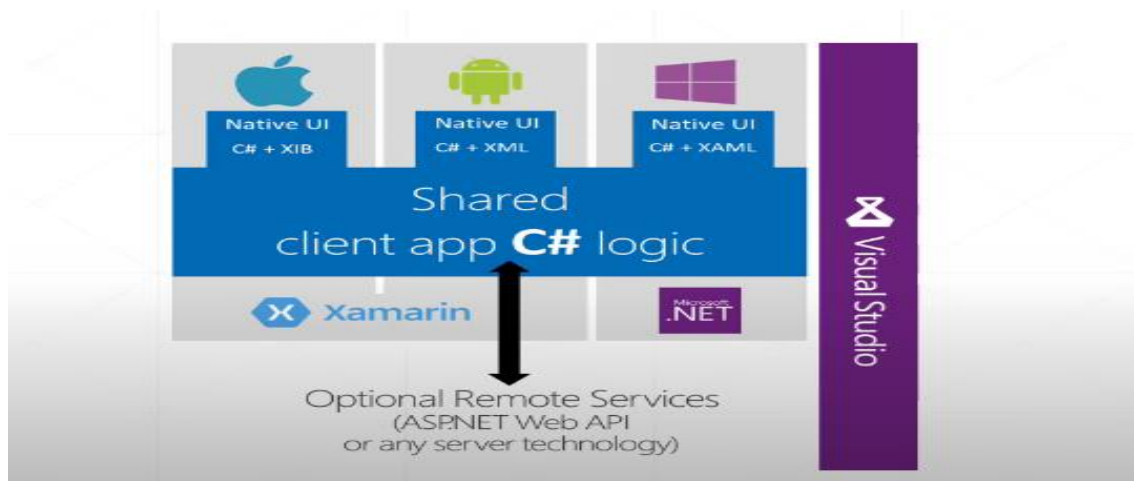
- Nhược điểm:

+ Không đảm bảo hiệu suất.

+ Không đảm bảo tiêu chuẩn UX/UI.

➔ Sự ra đời và cách dùng **C#** , **Xamarin**

* Xamarin sử dụng ngôn ngữ C# và framework.Net để tạo ứng dụng cho mọi nền tảng. Khi bạn tạo một ứng dụng di động trên Xamarin, bạn sử dụng cùng một ngôn ngữ, API và cấu trúc dữ liệu hoặc logic của ứng dụng như C#, do đó, thông thường 90% mã chức năng có thể được sử dụng trên iOS và Android. Qua đó, chi phí và thời gian phát triển ứng dụng di động cho hai trong số các nền tảng phổ biến nhất có thể giảm đáng kể. Ngoài ra, có rất nhiều IDE miễn phí hỗ trợ nó, chẳng hạn như Xamarin IDE (dành cho Mac) hay Visual Studio (dành cho Windows).



Hình 3: Xamarin App

1.4. Ưu nhược điểm của Xamarin

- Ưu điểm:

- + Tạo **native** app được viết bằng code C#.
- + Hỗ trợ 100% API của android và iOS.
- + Tool & hỗ trợ ALM từ Visual Studio.
- + Chia sẻ Cross platform, tiết kiệm tài nguyên.

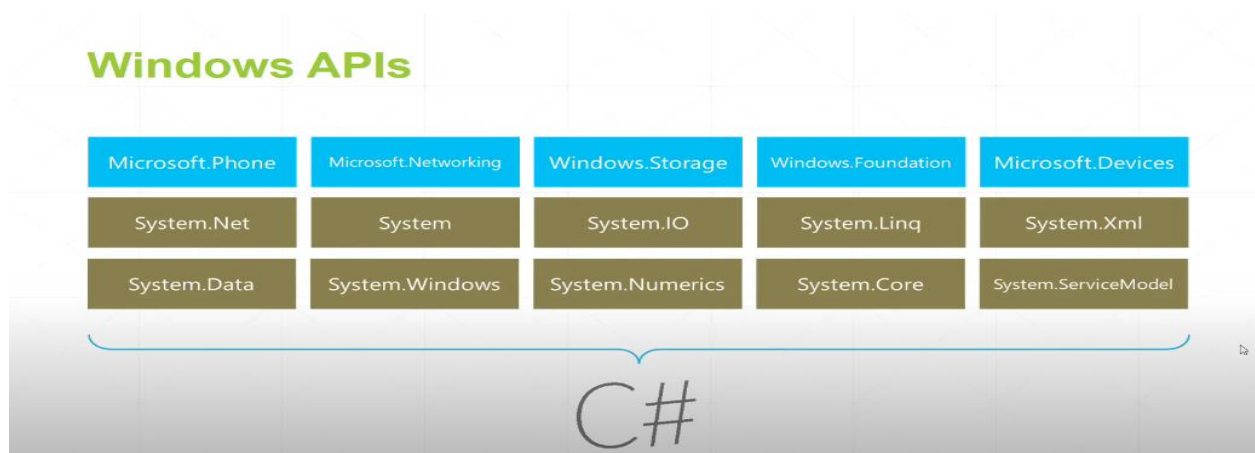
- Nhược điểm: đi kèm với đó là những hạn chế nhất định

- + Yêu cầu môi trường cao.
- + Yêu cầu lượng kiến thức nhất định.

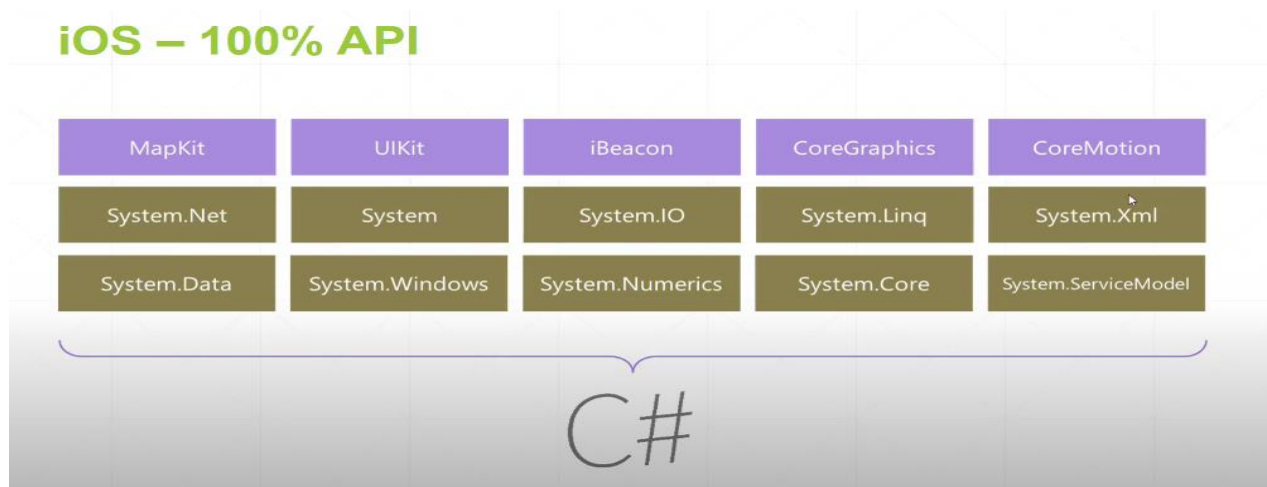
* **Native app đối với Xamarin:** là một ứng dụng đảm bảo 3 yếu tố

- Giao diện sử dụng control chính chủ.
- Đảm bảo truy cập 100% API iOS cung cấp.
- Đảm bảo performance (hiệu năng sử dụng)

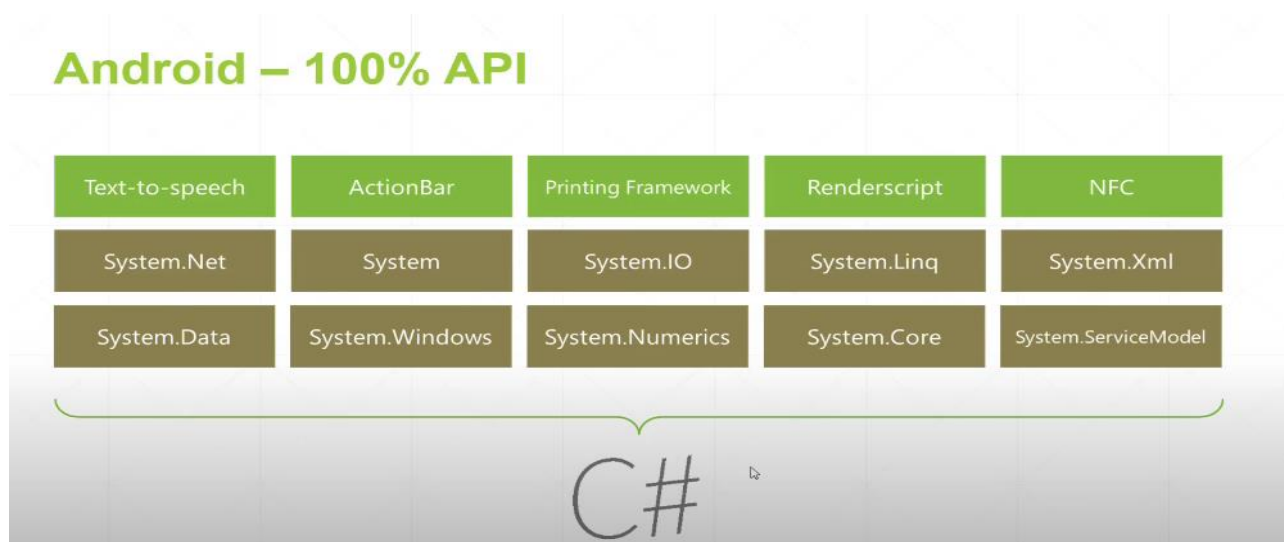
* Các API được thể hiện như sau:



Hình 4: Window APIs



Hình 5: iOS - API



Hình 6: Android API

1.5. Một số đặc điểm của Xamarin

- Dành cho người thích lập trình mobile bằng ngôn ngữ C#, có một số đặc điểm như sau:

+ **Kết nối – Binding – hoàn toàn với SDK nguyên mẫu:** Xamarin xây dựng theo cơ chế chuyển đổi trực tiếp giữa phương thức trong SDK của Android và iOS sang ngôn ngữ .Net, binding một cách mạnh mẽ, dễ dàng chuyển đổi và sử dụng. Cung cấp trình dò

lỗi và kiểm tra “compile-time” hoặc trong quá trình phát triển, giảm thiểu lỗi runtime và tăng chất lượng ứng dụng.

+ **Tương thích với Objective-C, Java, C, và C++:** Xamarin cho phép chèn trực tiếp các thư viện Objective-C, Java, C, và C++, giúp lập trình viên có thể sử dụng các thư viện của bên thứ 3 một cách dễ dàng.

+ **Modern Language Constructs:** Do sử dụng C#, một ngôn ngữ hiện đại và có những phần vượt trội hơn với Objective-C và Java, như Dynamic Language Features, Lambdas, LINQ, Parallel Programming, Generics, v.v...

+ **Base Class Library (BCL):** Ứng dụng Xamarin sử dụng .NET BCL, một bộ gồm rất nhiều class hỗ trợ toàn diện và sắp xếp hợp lý, như XML, Database, Serialization, IO, String, Networking và nhiều thứ khác. Modern Integrated Development Environment (IDE): Xamarin sử dụng bộ công cụ Xamarin Studio trên Mac OS, và Xamarin Studio hoặc Visual Studio 2013 trên Windows.

+ **Hỗ trợ Cross Platform:** Xamarin hỗ trợ đa nền tảng một cách tinh vi trên 3 hệ điều hành di động phổ biến là iOS, Android và Windows Phone. Ứng dụng có thể được viết với 90% code được chia sẻ/dùng chung. Qua đó có thể giảm đáng kể chi phí và thời gian phát triển ứng dụng di động cho 3 nền tảng phổ biến nhất.

+ **Cách Xamarin hoạt động:** Xamarin có hai sản phẩm chính Xamarin.iOS và Xamarin.Android, tất cả đều xây dựng trên Mono, đó là phiên bản mã nguồn mở của Framework .NET và được dựa theo tiêu chuẩn của .NET ECMA công bố. Mono đã có lâu giống như .NET và trên các Platform: Linux, Unix, FreeBSD, và Mac OS X.

MonoTouch.dll and Mono.Android.dll: Xamarin đã được xây dựng với một tập hợp con của .NET BCL được gọi là hồ sơ Xamarin Mobile. Hồ sơ này đã được tạo riêng

cho các ứng dụng điện thoại di động và đóng gói thành MonoTouch.dll và Mono.Android.dll (tương ứng dành cho iOS và Android). Gần giống như cách ứng dụng Silverlight (and Moonlight) xây dựng lại. Trong thực tế Xamarin di động tương đương với Silverlight 4.0 các lớp BCL được thêm. Sau khi ứng biên dịch thành công. Ứng dụng sẽ đóng gói lại thành tập tin có đuôi .apk cho Android và đuôi. app cho iOS

- Vòng đời phát triển phần mềm di động (Mobile Development SDLC): vòng đời phát triển di động rất rộng không khác mấy so với vòng đời phát triển phần mềm cho ứng dụng web hay ứng dụng cho máy tính cá nhân.

*** Thường vòng đời giành cho phần mềm di động có 5 phần chính:**

- **Khởi tạo:** tất cả ứng dụng đều bắt đầu từ một ý tưởng. Ý tưởng đó sẽ tinh tế thành cơ sở bền vững cho ứng dụng.

- **Thiết kế:** Giai đoạn thiết kế bao gồm xác định trải nghiệm người dùng của ứng dụng (UX), chẳng hạn như: bố cục chung trông như thế nào, cách thức hoạt động của ứng dụng, v.v. và chuyển UX thành thiết kế giao diện người dùng (UI) phù hợp, thường là với sự trợ giúp của một nhà thiết kế đồ họa.

- **Phát triển:** Thông thường giai đoạn chuyên sâu tài nguyên nhất, đây là tòa nhà thực tế của ứng dụng.

- **Ổn định:** Khi ứng dụng chuẩn bị hoàn thành, QA thường bắt đầu bằng việc kiểm tra ứng dụng và sửa lỗi. Thông thường, một ứng dụng sẽ bước vào giai đoạn beta giới hạn trong đó cơ sở người dùng rộng lớn hơn sẽ có cơ hội chơi với ứng dụng đó và cung cấp phản hồi cũng như thông báo về các thay đổi.

- **Triển khai:** thường thì các giai đoạn phát triển được triển khai chồng lên nhau. Như: giai đoạn phát triển được bắt đầu trong khi giai đoạn thiết kế sắp hoàn thành. Ngoài ra ứng dụng đã tới giai đoạn ổn định nhưng vẫn cần thêm vẫn cần thêm mới chức năng.

*** Những lưu ý khi phát triển di động chúng ta cần quan tâm đến tính đa nhiệm, kích cỡ của thiết bị, loại thiết bị, loại hệ điều hành và giới hạn tài nguyên:**

- **Đa nhiệm:** Có hai hạn chế quan trọng đối với việc sử dụng đa nhiệm trên thiết bị di động. Đầu tiên là giới hạn số lượng màn hình hiển thị đồng thời nhiều ứng dụng để thiết bị di động chỉ có thể chạy cùng một lúc. Thứ hai, có nhiều ứng dụng chạy và đòi hỏi nhiều tài nguyên làm hao pin thiết bị.

- **Kích cỡ thiết bị:** thiết bị di động thường là điện thoại hoặc máy tính bảng, điện thoại bị giới hạn về không gian màn hình còn máy tính bảng thì lớn hơn nhưng cả hai đều nhỏ so với laptop. Vì thế giao diện người dùng cần phải thiết kế đặc biệt để phù hợp với màn hình điện thoại và máy tính bảng.

- **Thiết bị và phân mảnh hệ điều hành:** Mỗi thiết bị có phần cứng và khả năng khác nhau. Một số ứng dụng yêu cầu các tính năng đặc biệt trên một thiết bị, vì vậy ứng dụng sẽ không hoạt động trên một thiết bị khác. Ví dụ, không phải tất cả điện thoại di động đều có camera trước, nếu chúng ta xây dựng ứng dụng trò chuyện video, một số thiết bị sẽ không hoạt động, khi thiết kế giao diện ứng dụng, chúng ta cần chú ý đến kích thước và tỷ lệ màn hình của thiết bị. .Ngoài ra bạn cần quan tâm đến độ phân giải của thiết bị, trong giai đoạn phát triển khi viết một chức năng nào đó bạn cần test chức năng này trước. Ví dụ: khi một chức năng nhất định cần được sử dụng trên một thiết bị như máy ảnh, trước tiên cần yêu cầu sử dụng chức năng đó, sau đó chức năng cần thiết phải được hệ điều hành hỗ trợ và cuối cùng là đặt nó. Thử nghiệm là một bước rất quan trọng, vì vậy hãy thử nghiệm ứng dụng của bạn trên thiết bị thực càng sớm càng tốt và thường xuyên.

- **Giới hạn tài nguyên:** Các thiết bị di động đang trở nên nhiều hơn và nhiều khả năng hơn. Những thứ giới hạn hiệu suất so với pc như dung lượng bộ nhớ thường không quan tâm khi phát triển ứng dụng cho pc vì bộ nhớ thực có thể được sử dụng cùng với bộ nhớ ảo nhưng đối với di động, chúng ta có thể sử dụng tất cả bộ nhớ có sẵn để tải xuống hình ảnh chất lượng cao. Ngoài ra, các ứng dụng yêu cầu bộ xử lý nhanh: chẳng hạn như trò chơi có thể là gánh nặng cho CPU, ảnh hưởng đến hiệu suất của thiết bị. Những lưu ý khi phát triển ứng dụng trên nền tảng hệ điều hành Windows Phone như khả năng đa nhiệm, thiết bị, cơ sở dữ liệu, bảo mật.

- **Đa nhiệm:** đa nhiệm của Windows Phone cũng giống với iOS và Android vòng đời của một trang, ứng dụng và chạy nền. Mỗi màn hình trong ứng dụng là một thể hiện của lớp trang. Trong đó mỗi sự kiện được liên kết với 1 hoạt động hoặc không.

- **Khả năng thiết bị:** mặc dù phần cứng của Windows Phone khá đồng đều vì sự ràng buộc của Microsoft nhưng vẫn có các tùy chỉnh về cấu hình nên bạn cần phải xem xét để tương thích với các thiết bị, cấu hình tùy chọn cho Windows Phone: máy ảnh, la bàn, con quay hồi chuyển.

- **Cơ sở dữ liệu:** trong khi iOS và Android hỗ trợ cơ sở dữ liệu SQLite để lưu trữ dữ liệu. Windows Phone 7 lại không hỗ trợ trong khi Windows phone 7.1 và 8 chỉ có thể sử dụng với LINQ to SQL và không hỗ trợ Transact-SQL. Nhưng bạn có thể sử dụng mã nguồn mở của SQLite để thêm vào ứng dụng.

- **Bảo mật:** ứng dụng khi cần sử dụng phần cứng của thiết bị hay chức năng của hệ điều hành cần phải yêu cầu quyền đó với Manifest (giống với Android). Một số lưu ý để phát triển ứng dụng Android như đa nhiệm, nhiều thiết bị, bảo mật.

- **Đa nhiệm trong Android có hai phần:** đầu tiên là vòng đời của Activity. Mỗi màn hình trong ứng dụng Android là một Activity và là chỗ tập hợp các sự kiện khi ứng dụng

đang chạy nền hoặc đang chạy. Thứ hai để sử dụng được đa nhiệm trong Android ta cần phải sử dụng dịch vụ. Dịch vụ sẽ hoạt động liên tục và tồn tại độc lập với ứng dụng được sử dụng để thực hiện các quá trình trong khi ứng dụng đang chạy nền.

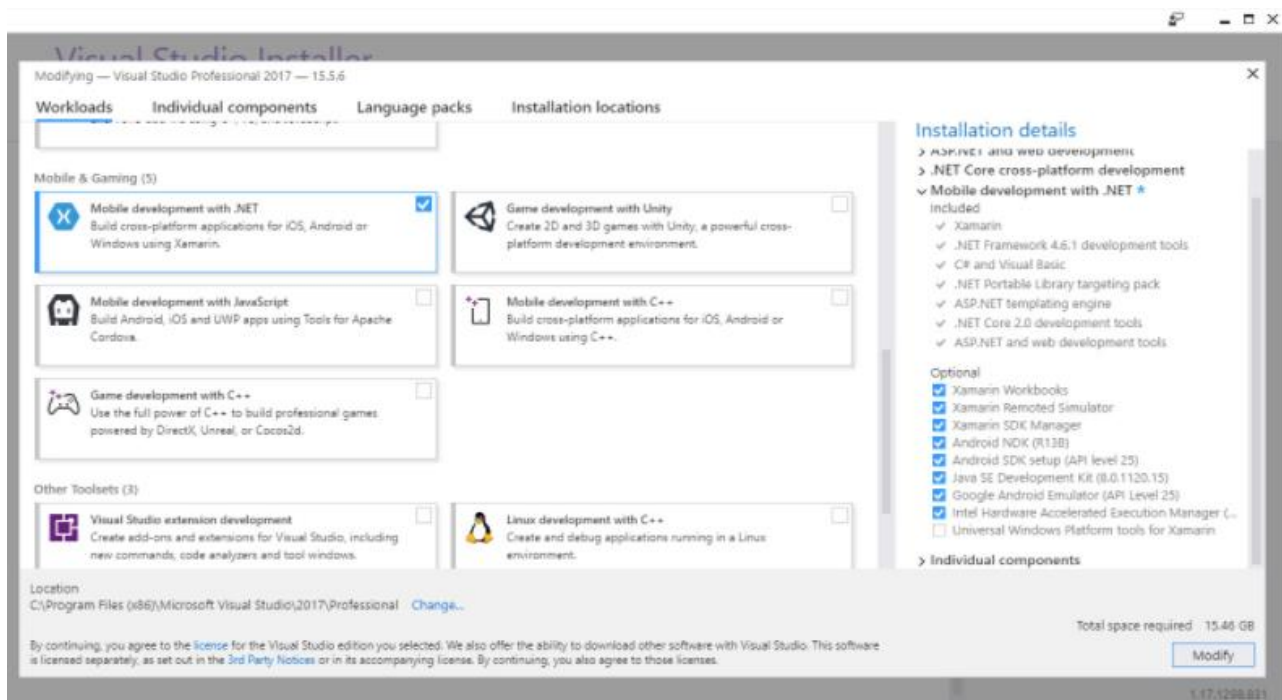
- Nhiều thiết bị không giống như iOS có rất ít thiết bị hay các thiết bị Windows Phone phải đáp ứng các yêu cầu tối thiểu. Google áp đặt giới hạn cho các thiết bị có thể chạy Android OS.

- **Bảo mật** trong lập trình ứng dụng Android đều chạy một cách riêng biệt. Lưu ý trong khi phát triển khi phát triển ứng dụng trên hệ điều hành iOS: đa nhiệm trong iOS đa nhiệm được điều khiển rất chặt chẽ.

CHƯƠNG 2: CÀI ĐẶT VÀ SỬ DỤNG XAMARIN TRÊN VISUAL STUDIO

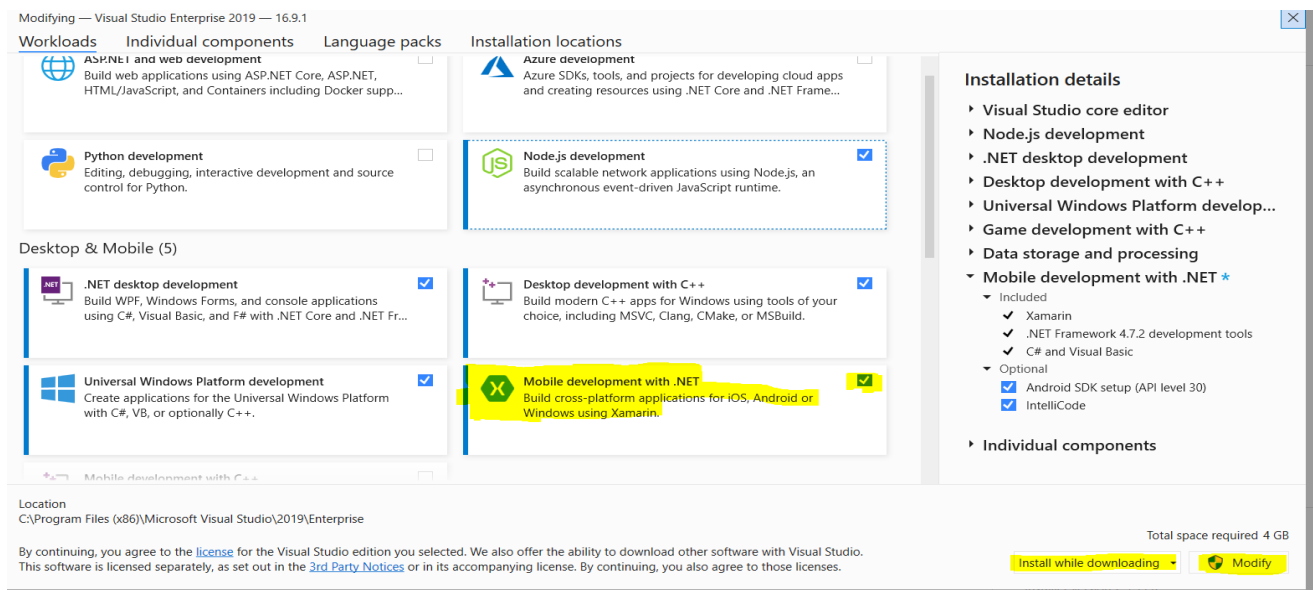
CÀI ĐẶT

* Để cài đặt **Xamarin** trên visual studio bạn chỉ cần mở VS Installer lên (tương tự ở các phiên bản cài đặt) chọn vào các mục giống trong hình để Modify



Hình 7: Cài đặt Xamarin

- Gần đây Xamarin đã có một số thay đổi nhất định



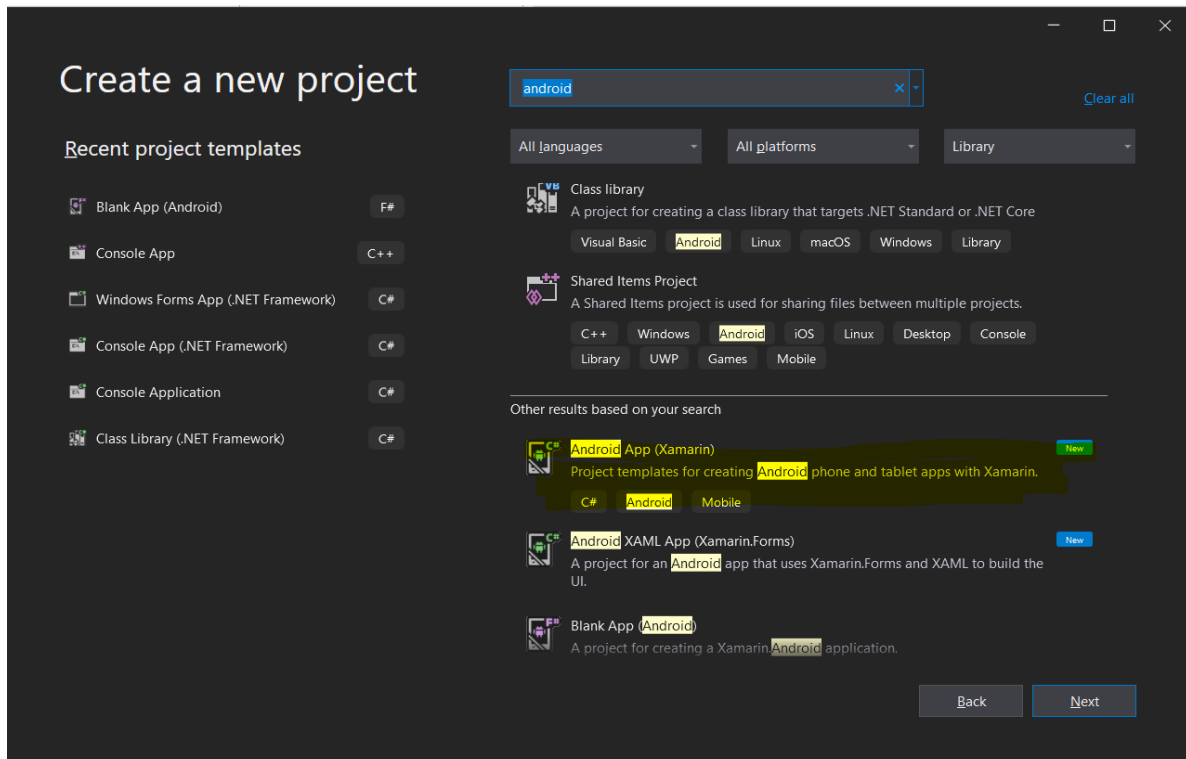
Hình 8: Cài đặt Xamarin 2

Sau khi cài đặt xong, các bạn mở Visual Studio lên và chọn Visual C#, các bạn sẽ thấy các mục Android, IOS,...

TẠO PROJECT DEMO ĐẦU TIÊN

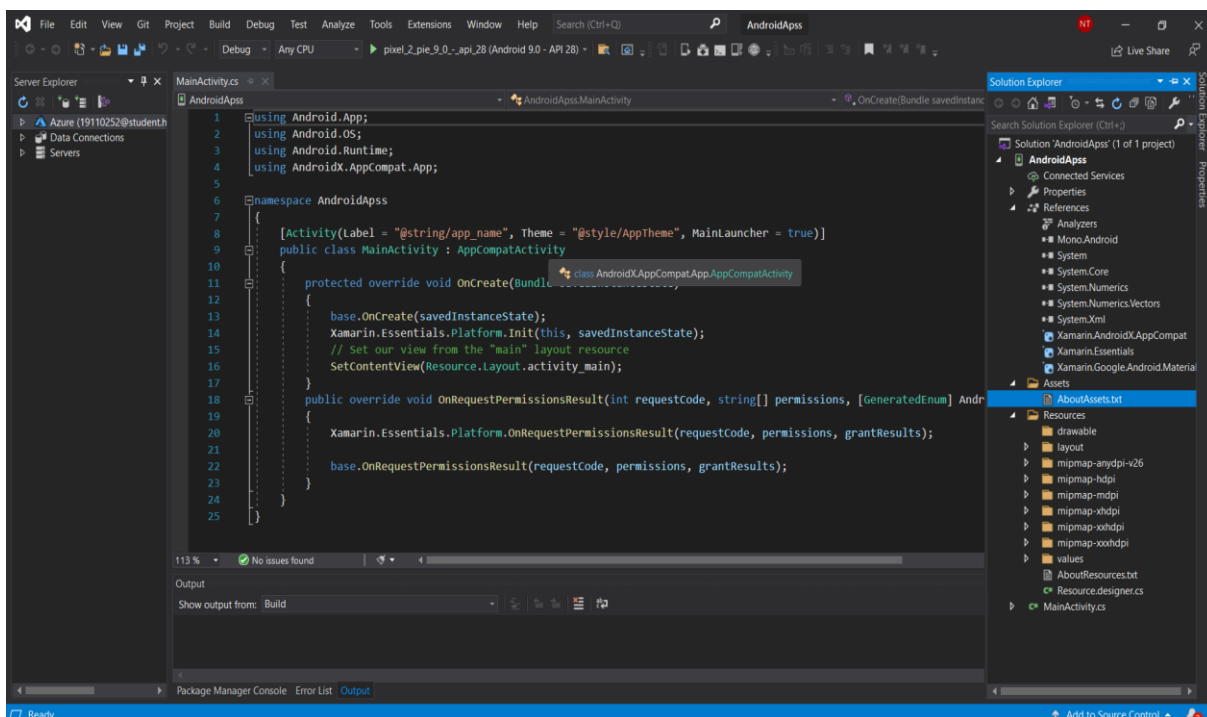
Các bạn chỉ cần New Project => Android => Blank App hoặc tùy loại mà các bạn chọn.

Tạo xong thì cứ chạy thử (nhớ cài đặt máy ảo hoặc cài luôn Visual Studio Emulator trong VS Installer). Nếu chạy được máy ảo và app hiện ra có nghĩa là đã cài thành công và có thể tiếp tục sử dụng Xamarin tiếp. Còn nếu bạn nào chưa cài được thì hãy gỡ toàn bộ Visual ra và cài lại.



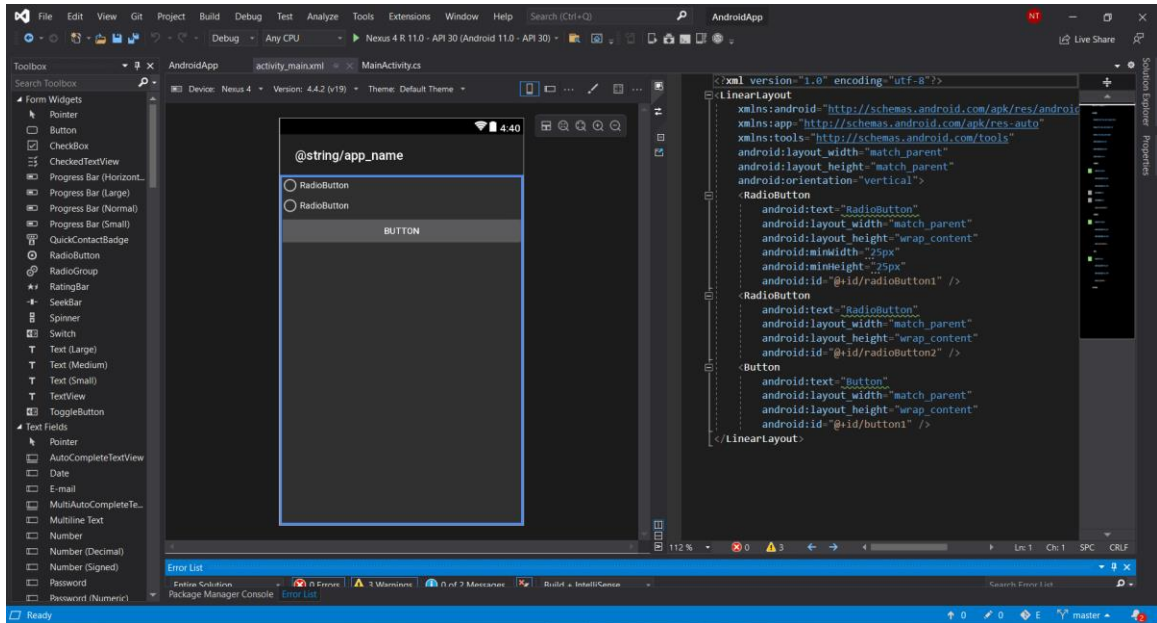
Hình 9: Tạo Project đầu tiên

- Cấu trúc của một Blank App android.

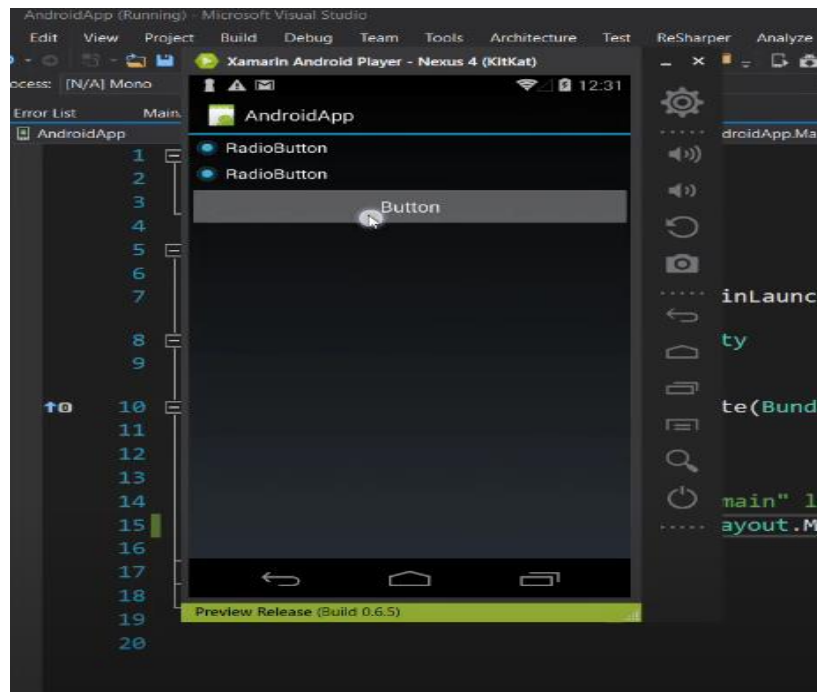


Hình 10: Cấu trúc của Project Android

- activity_main.xml



Hình 11: Activity_main.xml Android



Hình 12: Khởi động Android App trên Device

- Ngoài ra việc cài đặt và sử dụng Xamarin còn phụ thuộc vào một số yếu tố khác.

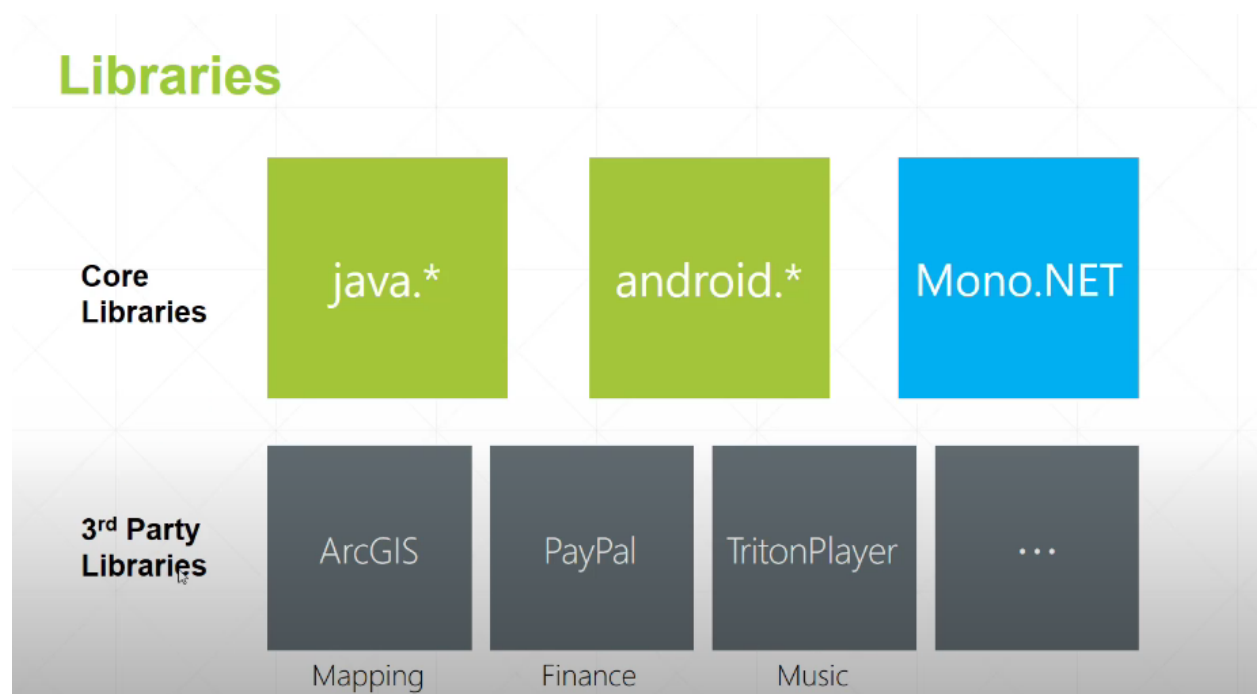
CHƯƠNG 3: TÌM HIỂU THÊM VỀ XAMARIN VÀ VÍ DỤ MINH HOẠ

3.1. Xamarin Android là gì?

- Đầu tiên Xamarin Android là một thành phần trong bộ lập trình của Xamarin, Xamarin Android giúp xây dựng một ứng dụng android với native Android được viết hoàn toàn bằng C#.

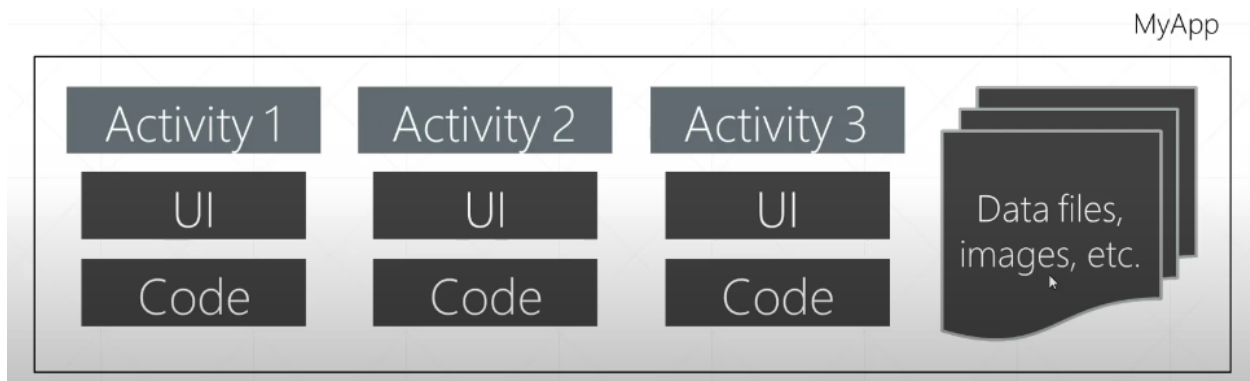
- Hỗ trợ các cấu trúc C# mới nhất như generics, asyns/await, LINQ, lambda expression...

*** Một số thư viện thường dùng:**



Hình 13: Libraries Android

*** App structure:** Một ứng dụng Android được cấu thành bởi các thành phần tương tác lẫn nhau gọi là Activity



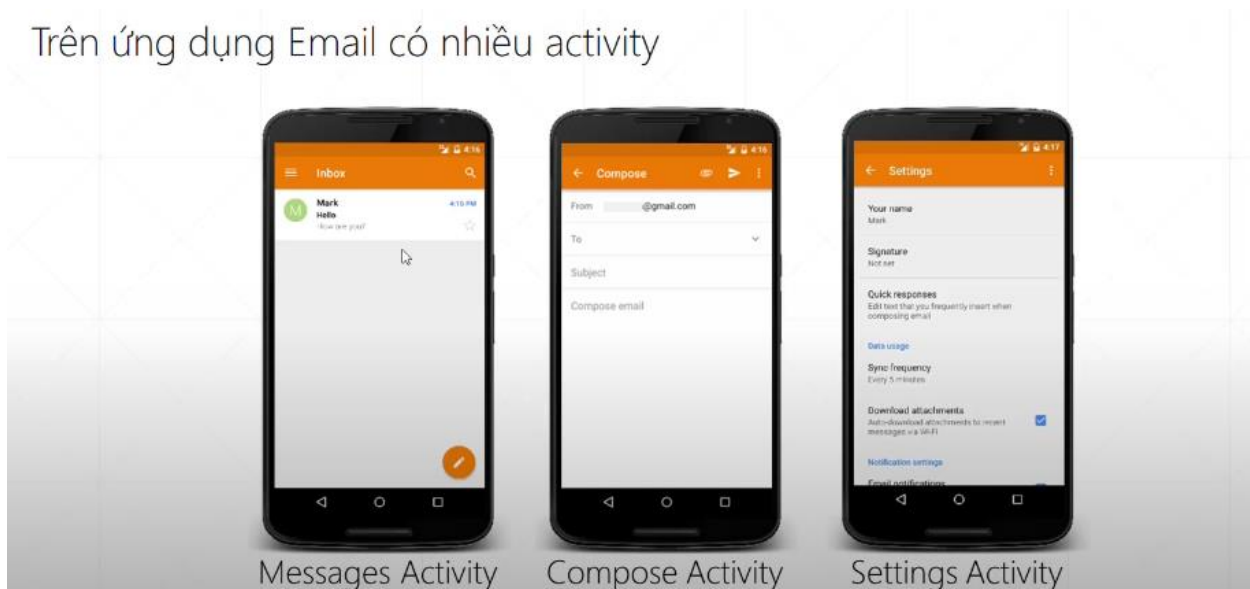
Hình 14: App structure

- Activity là gì?

Một Activity khai báo UI và xử lý cách hành vi cho người dùng cho một tác vụ nào đó.

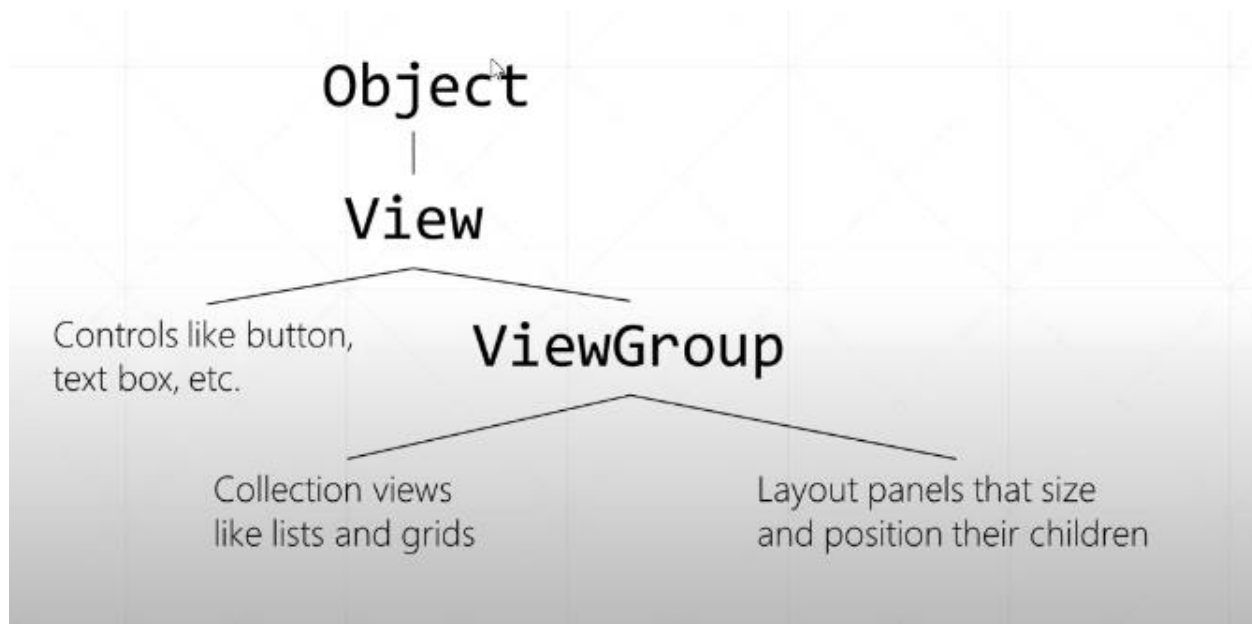
- Ví dụ: Email

Trên ứng dụng Email có nhiều activity



Hình 15: Ứng dụng email về activity

* **UI elements:** Android UI kết hợp các thành phần Views và ViewGroups.



Hình 16: UI Element trong Xamarin

- View là gì?

View là các thành phần trên giao diện hỗ trợ hiển thị nội dung và ghi nhận tương tác qua các event. (nhập liệu)



Hình 17: View trong Xamarin

- Layout là gì?

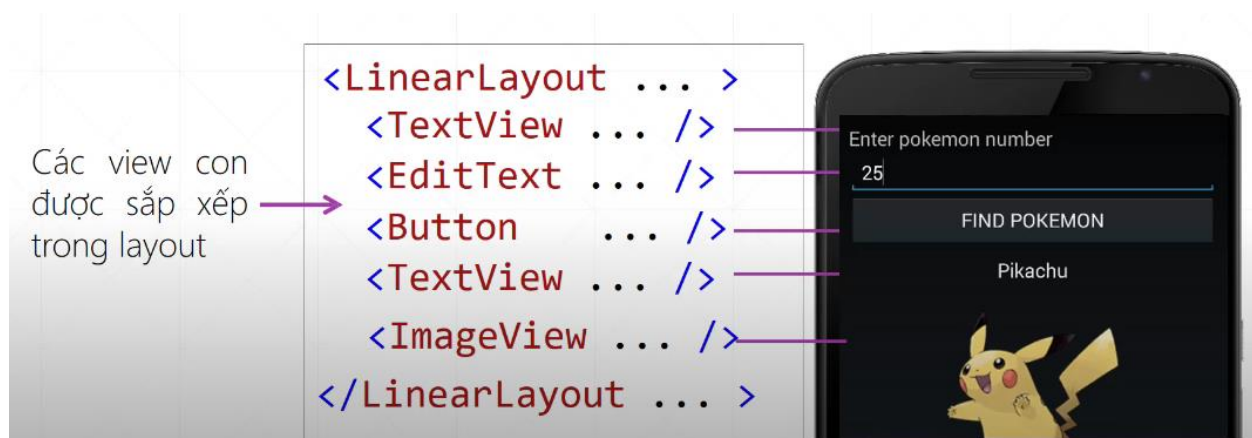
Layout tổ chức các thành phần bên trong nó, bố trí, sắp xếp, thay đổi kích thước theo các quy luật nhất định.



Hình 18: Layout trong Xamarin

- Layout file được viết như thế nào

UI View được viết theo ngôn ngữ XML với các tập tin (.axml)



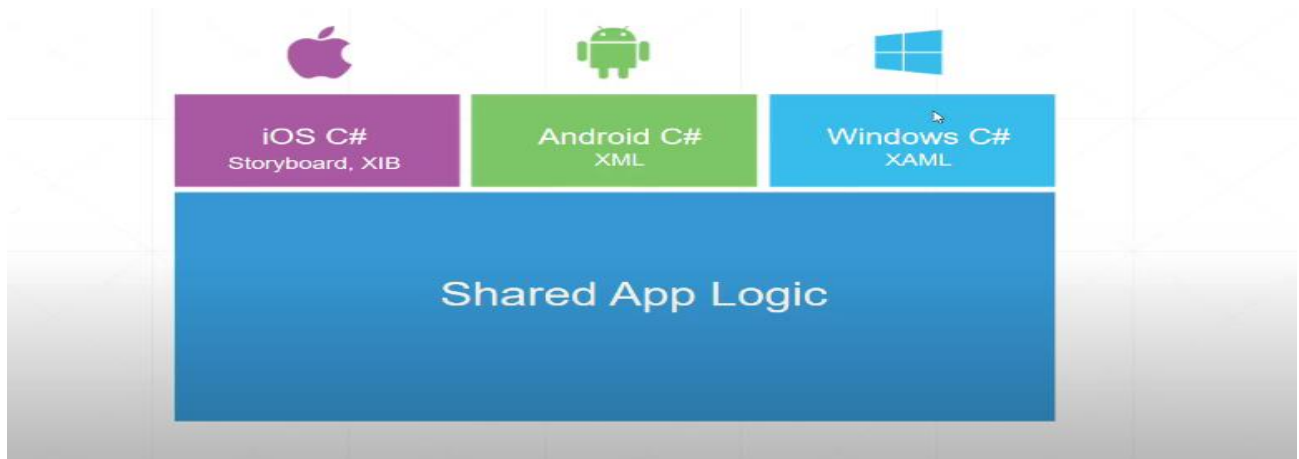
Hình 19: Layout file ~

3.2. Xamarin iOS là gì?

Vì để cài đặt Xamarin iOS cần có máy MAC nên phần này được bỏ qua trong bài báo cáo

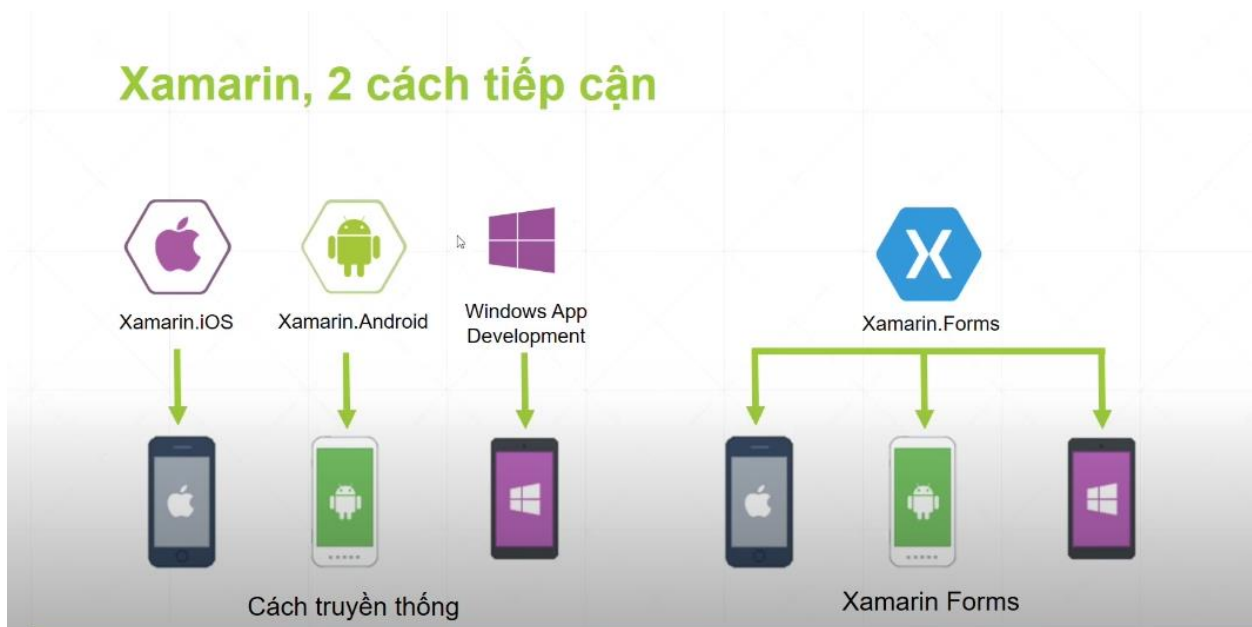
3.3. Xamarin Form là gì?

- Cách tiếp cận thuần



Hình 20: Shared App Logic

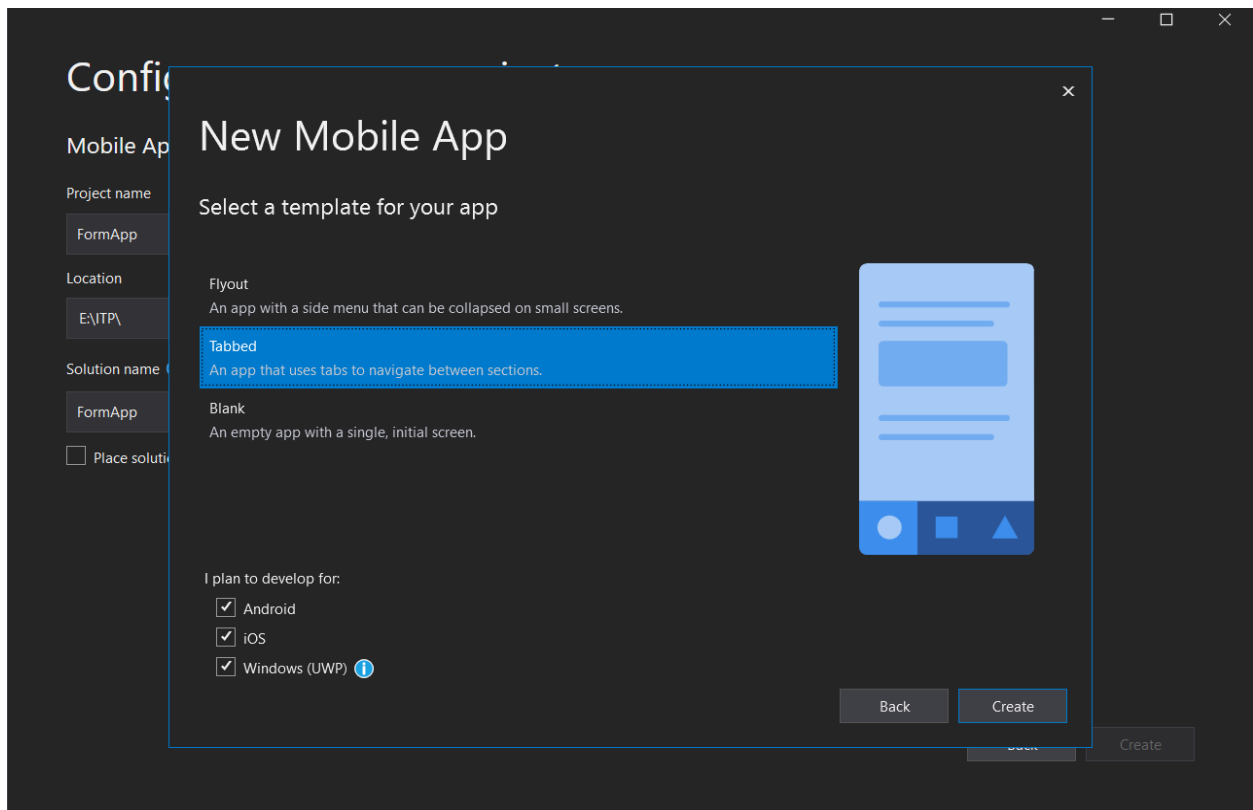
- Xamarin Forms giúp chia sẻ code giao diện (nghĩa là giao diện iOS, Android, Windows viết chung).
- Tóm gọn một cách dễ hiểu về Xamarin Forms



Hình 21: Xamarin cách tiếp cận

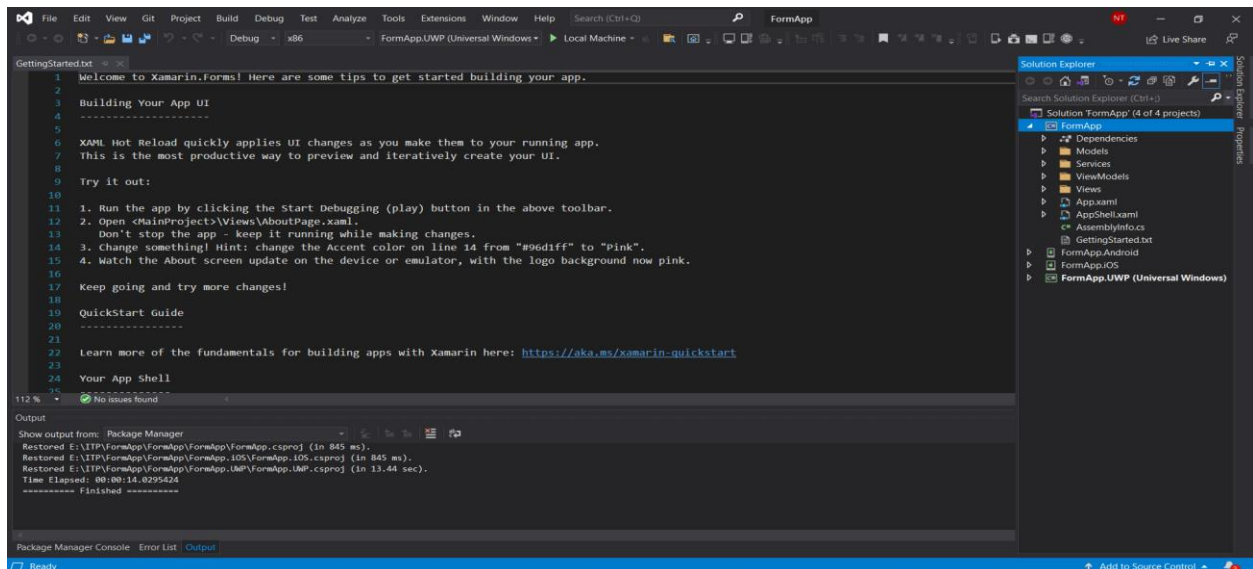
3.4. Demo xamarin Forms:

- + Tạo Form App (project)



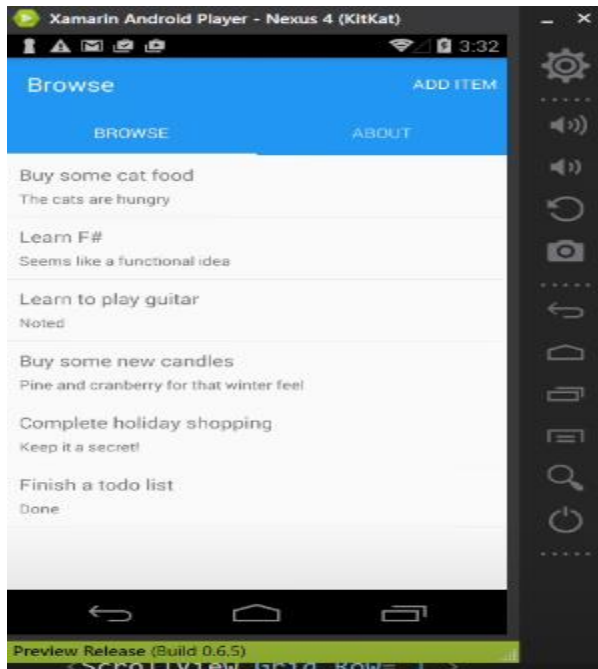
Hình 22: Tạo blank App

+ Cấu trúc của một Form Xamarin

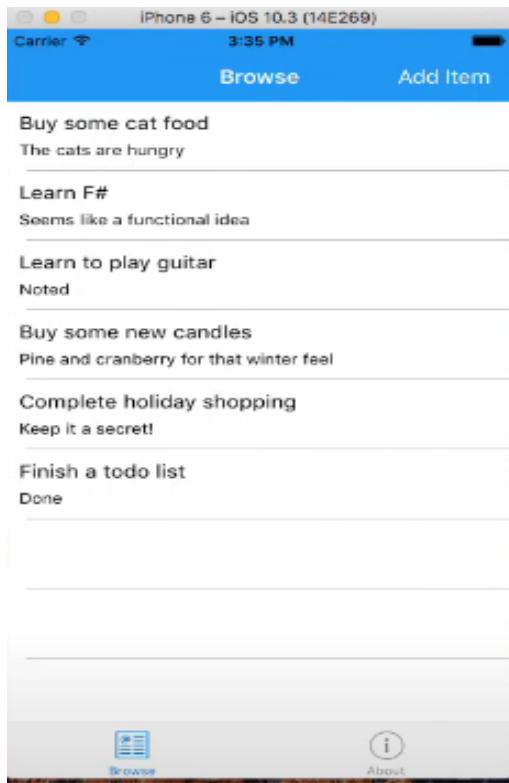


Hình 23: Cấu trúc của một Xamarin Forms

- Sau khi Set FormApp. as a Startup Project (Android and Iphone)



Hình 24: Chạy thử bản demo blank app



Hình 25: Chạy thử bản demo blank app 2

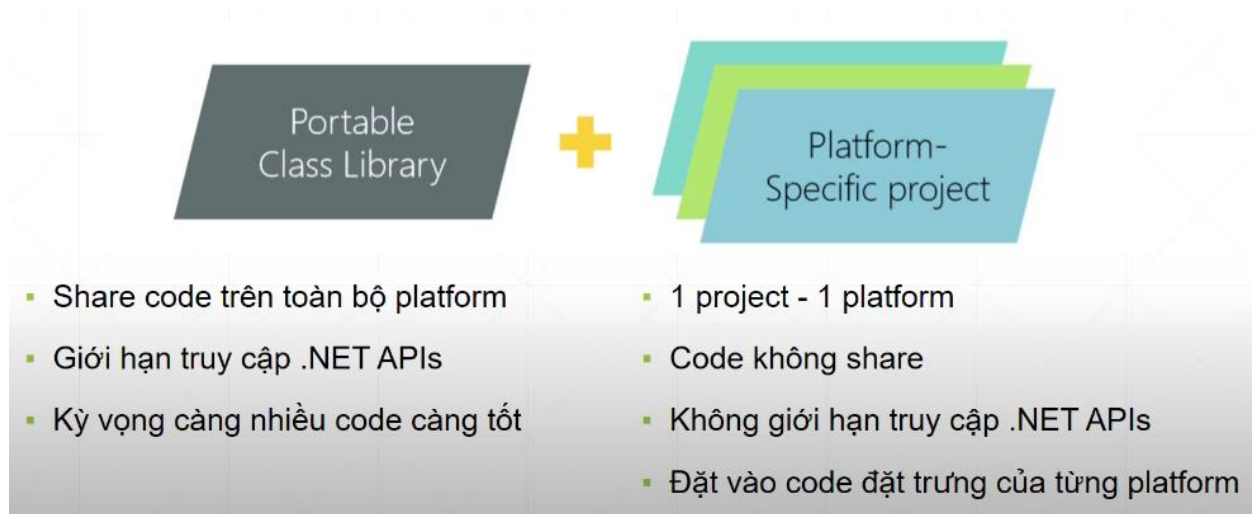
3.5. Môi trường phát triển Xamarin

- Môi trường phát triển ứng dụng Xamarin.



Hình 26: Môi trường phát triển Xamarin

- Kiến trúc của ứng dụng Xamarin.Forms



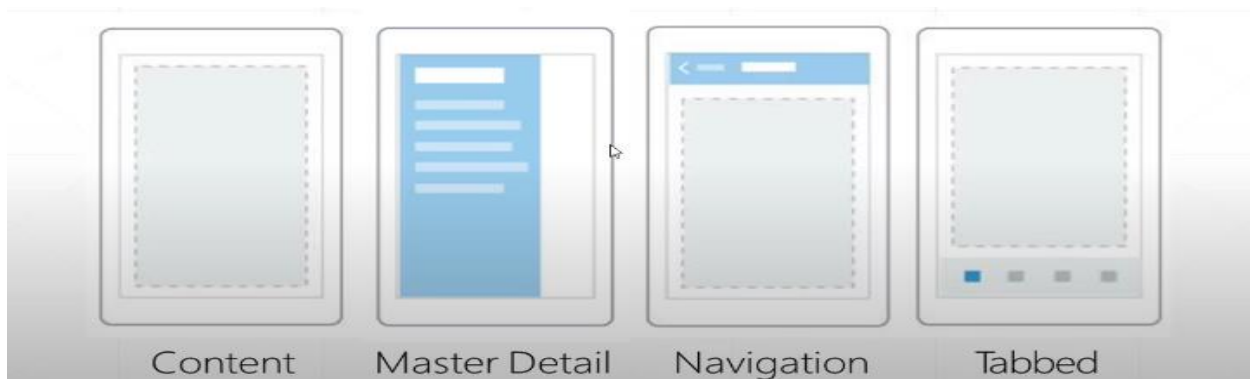
Hình 27: Kiến trúc của ứng dụng Xamarin.Forms

- Bên trong một ứng dụng Xamarin.Forms



Hình 28: Bên trong của 1 ứng dụng Xamarin Forms

- **Page** (được thấy trong Xamarin Forms): là một abstract class mà ta sử dụng để khai báo các thành phần sẽ hiển thị trên một màn hình bao gồm cách thức trình bày, tương tác và các thành phần UI (View) khác.



Hình 29: Page Xamarin Forms

- **Trong page thì chứa các view**: là base class cho tất cả các control dùng để hiển thị ra giao diện, hiện XF tại hỗ trợ gần như đầy đủ các control cần thiết.

Label	Image	SearchBar
Entry	ProgressBar	ActivityIndicator
Button	Slider	OpenGLView
Editor	Stepper	WebView
DatePicker	Switch	ListView
BoxView	TimePicker	
Frame	Picker	



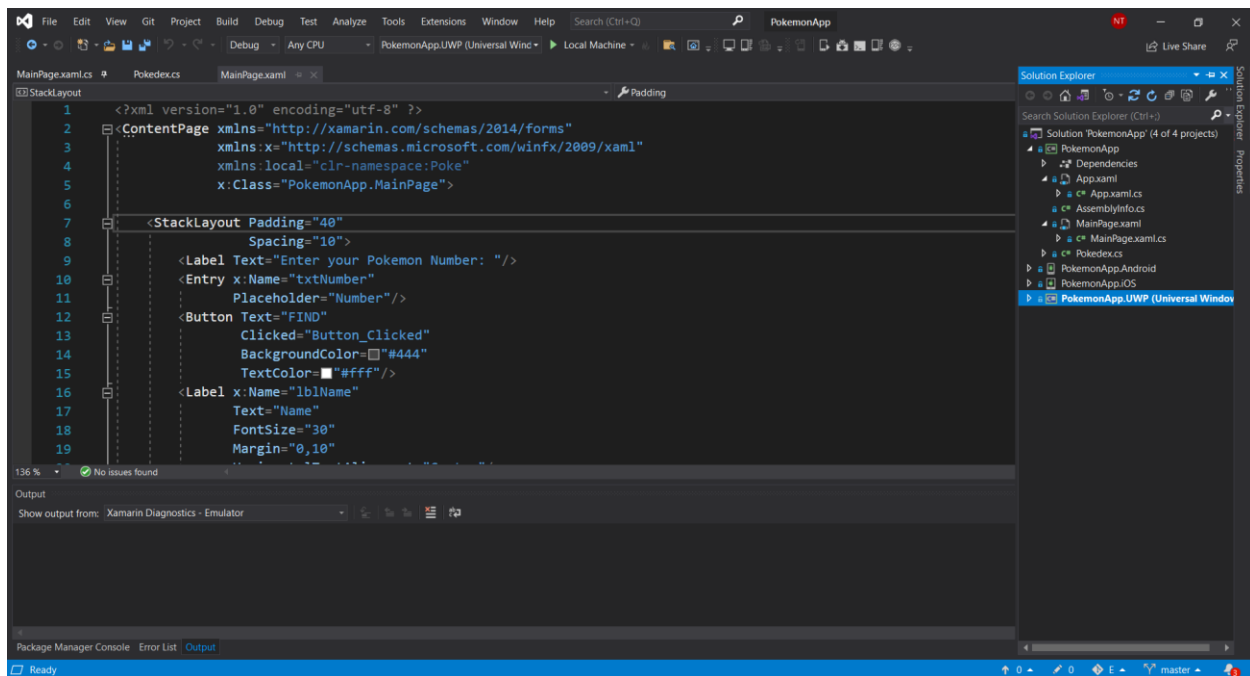
Hình 30: Views trong Xamarin Forms

- **Những view được bố trí vào Layout:** là thành phần giao diện tổ chức và bố trí các view bên trong nó theo một cấu trúc nhất định.



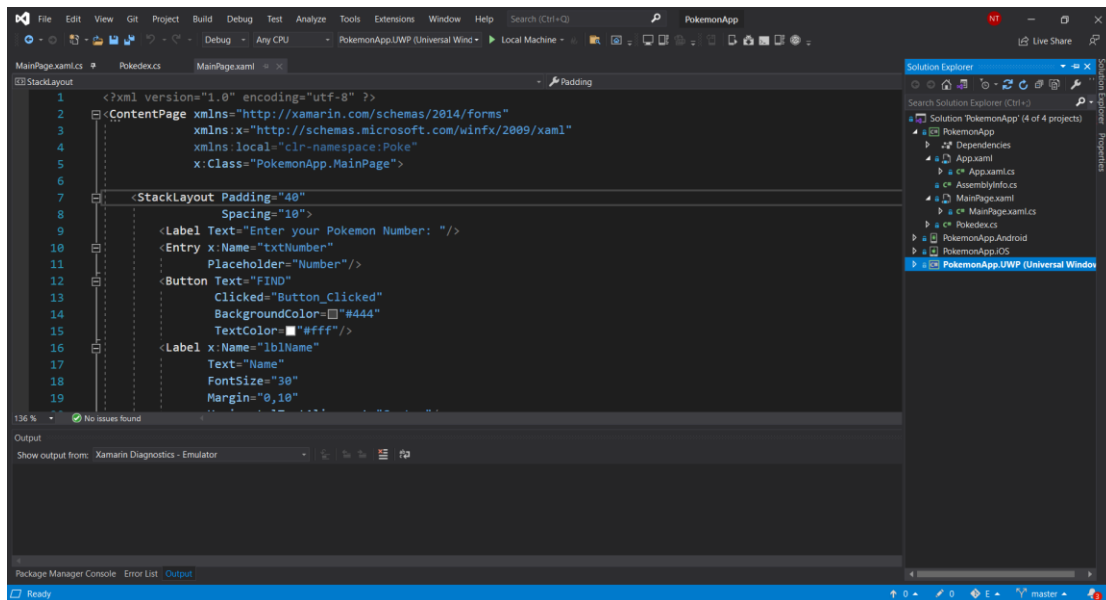
Hình 31: Rõ hơn về các layout trong Xamarin

- **Demo Pokemon cho Xamarin Forms** (tạo một Project trong visual studio 2019 – với Moblie App). Đây là cấu trúc của một Moblie App



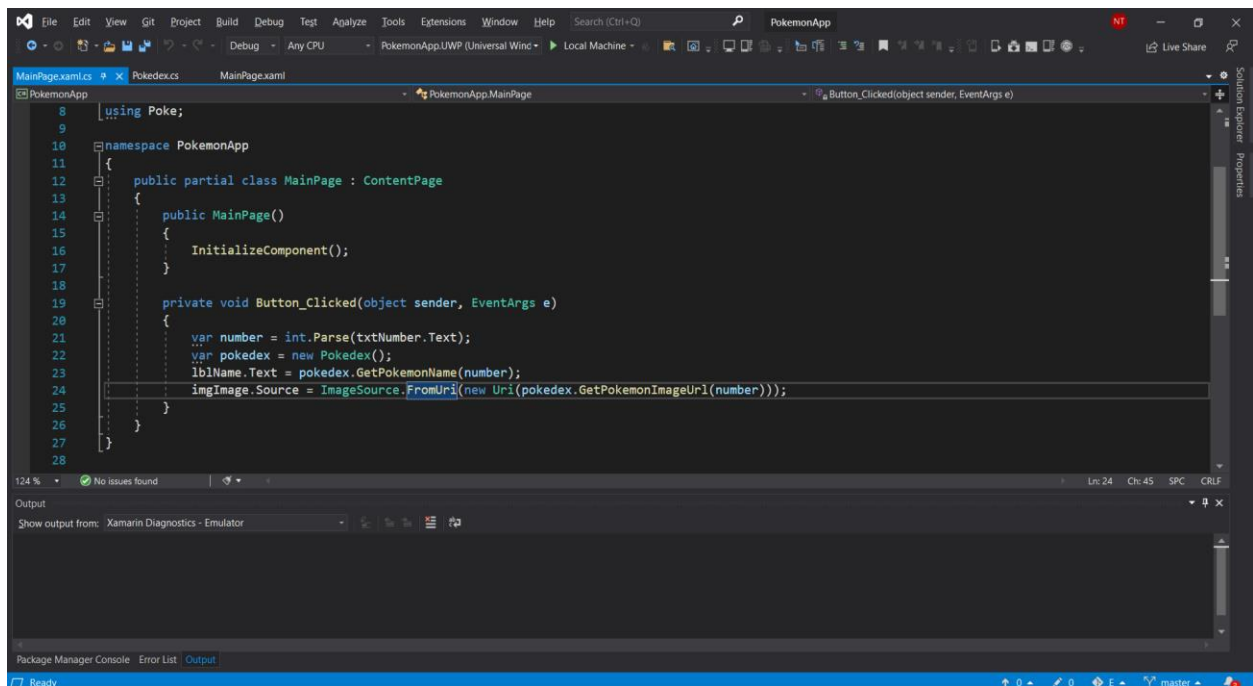
Hình 32: Tạo Mobile App trong Xamarin

- Đồng thời ta thêm đoạn code vào MainPage.xaml (dùng để tạo cấu trúc hiển thị cho một simulator - ứng dụng di động – là những gì mà trên màn hình sẽ hiện thị).



Hình 33: Kiến trúc tương tự trong Xamarin Mobile App 1

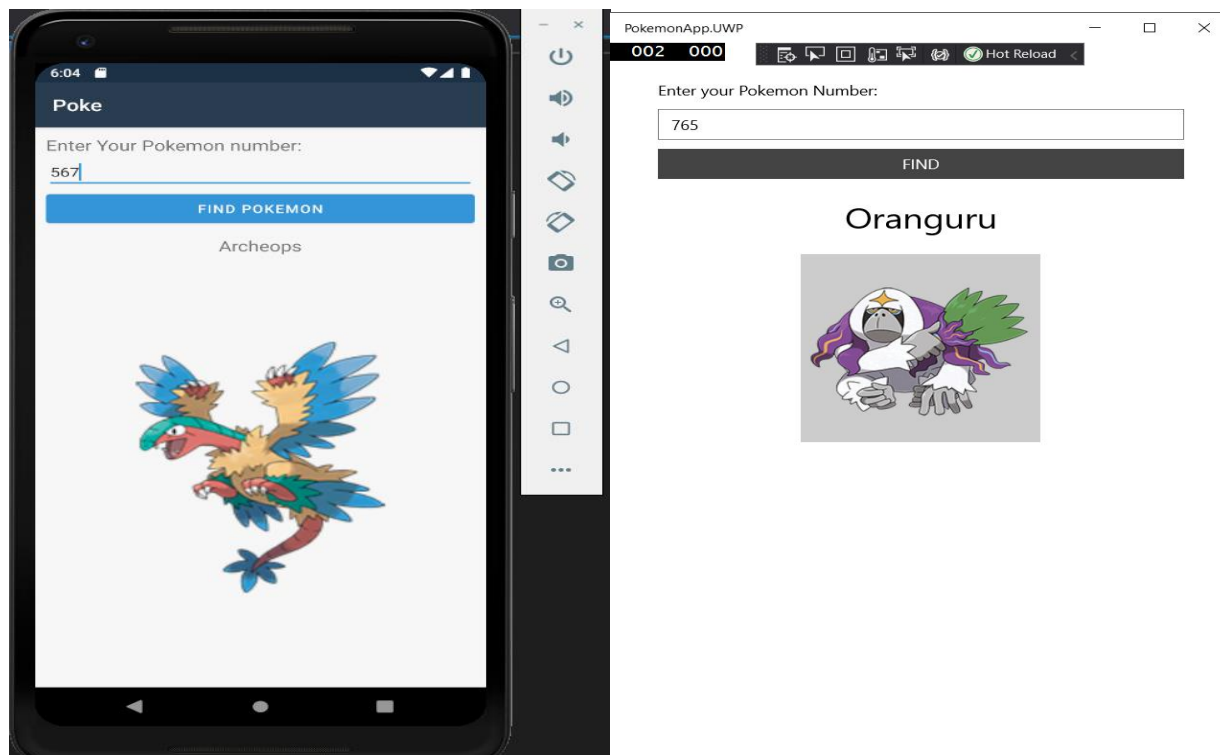
- Tiếp theo ở MainPage.xaml.cs (ta thêm đoạn code để xử lý tìm Pokemon)



Hình 34: Kiến trúc tương tự trong Xamarin Mobile App 2

- Ta nhúng code Pokedex.cs vào (chứa nội dung cho phần hiển thị).
 - Ở Moblie App ta có 3 lựa chọn chạy trên ứng dụng là Android, iOS và Windows trong bài sẽ chạy trên 2 thiết bị vì iOS phải được kết nối với máy MAC (hiện chưa có).
- Đây là kết quả khi chạy trên 2 thiết bị.

- **Android - Windows**



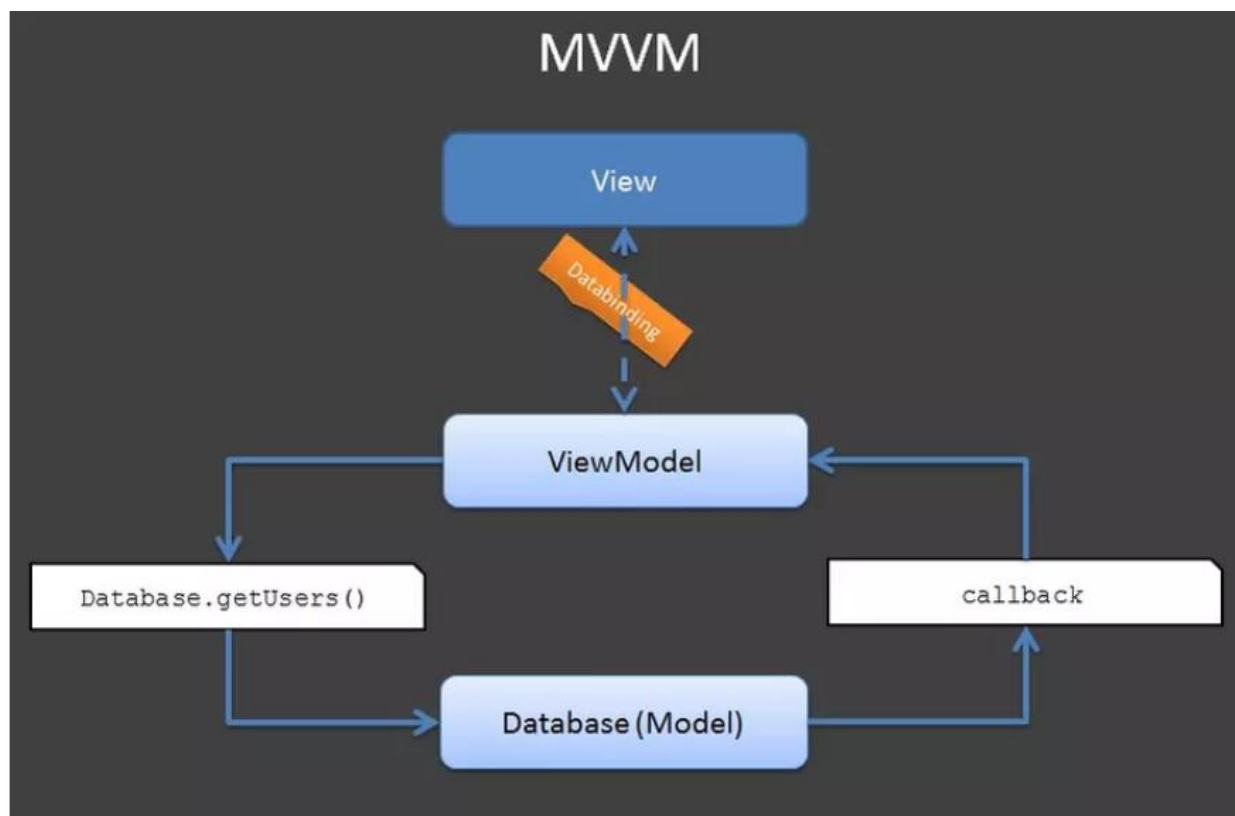
Hình 35: Chạy Project trong Mobile Xamarin Forms

3.6 Binding Value Converters trong Xamarin.Forms

3.6.1. Binding Value Converters là gì?

Nó là 1 chức năng của **DataBinding** giúp bạn chuyển đổi từ kiểu **dữ liệu gốc** sang **kiểu dữ liệu View yêu cầu**.

3.6.2. Tại sao cần Binding Value Converters?



Hình 36: Binding trong XamarinForms

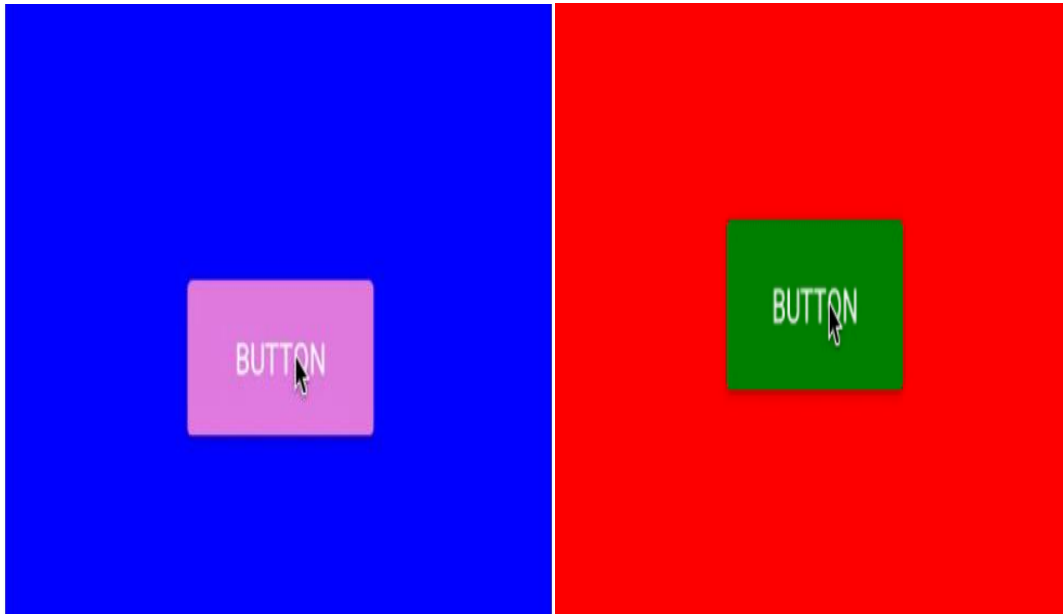
Khi bạn sử dụng **MVVM pattern** kết hợp cùng **DataBinding** thì bạn có thể thấy rằng **View** và **ViewModel** tách biệt với nhau, giao tiếp qua **DataBinding**. Vì vậy sẽ sinh ra nhiều vấn đề, chẳng hạn như khi bạn muốn tạo hiệu ứng kích vào 1 button và làm thay đổi màu Background Button của nó như này:



Hình 37: When dont use binding

Vì chỉ có thể giao tiếp qua **DataBinding** nên khi vào trường hợp này chúng ta phải tạo ra **1 biến kiểu Color** ở **ViewModel** và sẽ thay đổi giá trị khi sự kiện Click xảy ra.

Nhưng với trường hợp này:



Hình 38: When use binding

Nếu như vẫn cách cũ thì chúng ta phải tạo **thêm 1 biến kiểu Color** cho Background Screen ở **ViewModel** nữa.

3.6.3. Làm thế nào để sử dụng Binding Value Converters?



Hình 39: How to use binding?

Để xử lý trường hợp trên mình sử dụng 1 biến kiểu **Boolean** để xác định khi nào được **Click** ở **ViewModel**.

Class ViewModel:

```
public class MainViewModel : BaseViewModel
{
    public MainViewModel()
    {
        ButtonClick = new Command(() =>
        {
            IsSelected = IsSelected ? false : true;
        });
    }

    private bool _isSelected;
    public bool IsSelected
    {
        get { return _isSelected; }
        set
        {
            _isSelected = value;
            PushPropertyChanged("IsSelected");
        }
    }

    public ICommand ButtonClick { get; set; }
}
```

Hình 40: Class in Binding 1

Sau đó tạo **Value Converters** bằng cách tạo **Class imlement IValueConverter** và thực hiện việc convert theo ý muốn:

```

class ColorConverter : IValueConverter
{
    public string _type;
    public const string IsColorChange = "IsColorChange";
    public const string IsBGColorChange = "IsBGColorChange";

    public ColorConverter(string type)
    {
        _type = type;
    }

    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        parameter = _type;
        Color color = Color.Transparent;
        if (parameter.Equals(IsColorChange))
        {
            bool isColorChange = (bool)value;
            return isColorChange ? Color.Violet : Color.Green;
        }
        else if (parameter.Equals(IsBGColorChange))
        {
            bool isBGColorChange = (bool)value;
            return isBGColorChange ? Color.Blue : Color.Red;
        }
        return color;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}

```

Hình 41: Class in Binding 2

Bước cuối cùng chúng ta **SetBinding** vào **View** và sử dụng **Class imlement IValueConverter** chúng ta vừa tạo:

```

public class MainPage : ContentPage
{
    public MainPage()
    {
        InitUI();
        BindingContext = ViewModel;
    }

    void InitUI() {
        StackLayout container = new StackLayout()
        {
            Orientation = StackOrientation.Horizontal,
            Spacing = 0,
            BackgroundColor = Color.White,
            HorizontalOptions = LayoutOptions.FillAndExpand,
            VerticalOptions = LayoutOptions.FillAndExpand,
        };
        container.SetBinding(Button.BackgroundColorProperty, nameof(ViewModel.IsSelected), BindingMode.Default, new ColorConverter(ColorConverter.IsBGColorChange));

        Button button = new Button()
        {
            Text = "button",
            TextColor = Color.White,
            FontSize = 10,
            VerticalOptions = LayoutOptions.CenterAndExpand,
            HorizontalOptions = LayoutOptions.CenterAndExpand,
            BackgroundColor = Color.Green,
        };
        button.SetBinding(Button.CommandProperty, nameof(ViewModel.ButtonClick));
        button.SetBinding(Button.BackgroundColorProperty, nameof(ViewModel.IsSelected), BindingMode.Default, new ColorConverter(ColorConverter.IsColorChange));

        container.Children.Add(button);
        Content = container;
    }

    MainViewModel ViewModel {
        get {
            return new MainViewModel();
        }
    }
}

```

Hình 42: Class in Binding 3

Các bạn có thể thấy chúng ta chỉ cần sử dụng **1 biến Boolean** để xác định **hàng động** và dựa vào đó **Value Converters** sẽ giải quyết nốt cho chúng ta. Và theo cách này bạn dù có **bao nhiêu hiệu ứng** trên màn hình sau khi **Click** thì đều có thể xử lý gọn gàng.

3.7. Mô hình MVVM trong Xamarin.Forms

3.7.1. MVVM là gì?

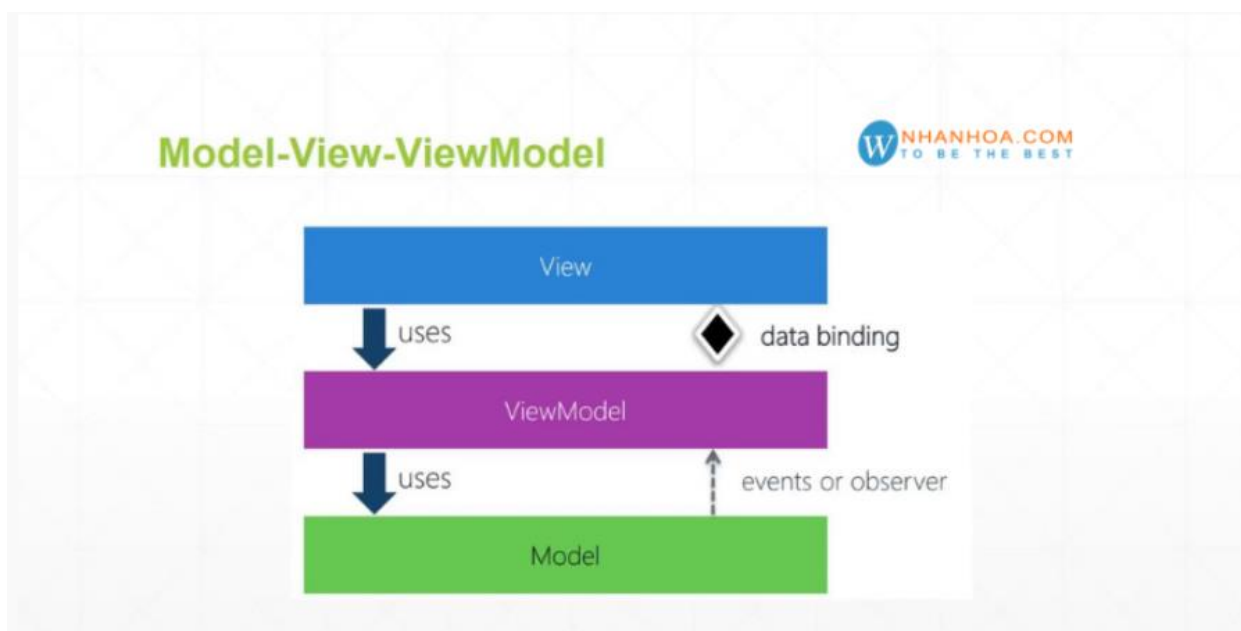
MVVM là viết tắt của Model - View - ViewModel, đây là mô hình hỗ trợ two-way data binding giữa View và View Model. Cụ thể mô hình MVVM được trình bày như sau:

- * **View**: Tương tự như trong mô hình MVC, View là phần giao diện của ứng dụng để hiển thị dữ liệu và nhận tương tác của người dùng. Một điểm khác biệt so với các ứng dụng truyền thống là View trong mô hình này tích cực hơn, nó có khả năng thực hiện các hành vi và phản hồi lại người dùng thông qua tính năng binding, command

- * **Model**: Cũng tương tự như trong mô hình MVC, Model là các đối tượng giúp truy xuất và thao tác trên dữ liệu thực sự.

- * **View Model**: Là lớp trung gian giữa View và Model. View Model có thể được xem là thành phần thay thế cho Controller trong mô hình MVC. Nó chứa các mã lệnh thực hiện

Data Binding, Command. Một điểm cần lưu ý là trong mô hình MVVM, các tầng bên dưới sẽ không biết được các thông tin gì về các tầng trên của nó.



Hình 43: Mô hình MVVM

* **Có thể hiểu rằng:** ViewModel sẽ đảm nhận công việc đồng bộ dữ liệu từ Model lên View. Mối quan hệ giữa View và View-Model là View sẽ được ánh xạ tới View Model nhưng ViewModel lại không biết thông tin gì về View. nó được ẩn giấu qua cách sử dụng Data-binding và cơ chế của mô hình Observer, một ViewModel có thể được ánh xạ từ nhiều View

3.7.2. Cấu trúc thư mục trong MVVM

Thông thường khi sử dụng với MVVM chúng ta nên tạo 3 thư mục chính chứa các file code liên quan

- Views

Tại đây chứa các file giao diện và mỗi file giao diện đều có class code-behind đi kèm. Đặc biệt file code-behind ta sẽ không sử dụng đến, mọi điều cần làm sẽ chuyển xuống class ViewModel. Tất nhiên là bạn có thể code trong file code-behind của XAML nhưng đồng nghĩa điều đó sẽ phá vỡ quy ước của MVVM. Bạn có thể khai báo thuộc tính

datacontext hoặc vài thiết lập khác nhưng nên hạn chế tối thiểu code ở đây. Views được sử dụng để kết hợp với các mô hình MVVM,... Nó dùng để cung cấp một sự chia tách gọn gàng của khái niệm giữa UI và presentation logic và data.

- Models

Trong thư mục Models tạo các tầng chứa dữ liệu và bất kỳ liên kết validation, logic nghiệp vụ để chắc chắn tính toàn vẹn của data, bạn có thể tách ra như một Repositories khác, chúng được dùng như một phần của mô hình MVVM

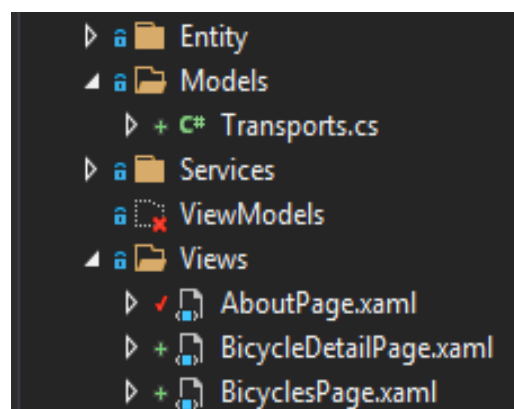
- ViewModels

Thông thường trong một file giao diện thì ta tạo ra một class View Models tương ứng (có đôi lúc ta tạo nhiều class phụ giúp tinh giản file code và gọi chúng trong class ViewModels chính).

ViewModels sẽ sử dụng các models nếu cần định nghĩa dữ liệu. Sự liên kết giữa View-ViewModel giúp chúng gửi và nhận dữ liệu, để hiểu rõ ta cần tìm hiểu các khái niệm về Binding, DataContext, Behaviors SDK, nhờ đó ta tách code-behind của View và đưa xuống View Model.

Ngoài ra một lớp ViewModels chứa presentation logic và state của ứng dụng.

ViewModel cần chứa các chức năng của ứng dụng. ViewModels định nghĩa properties, commands và events để chuyển đổi controls trong view cần data-bind.



Hình 44: Cấu trúc thường thấy của MVVM

- DataBinding

Data Binding là kỹ thuật dùng để tạo gắn kết giữa phần giao diện (UI) và dữ liệu thông qua phần business logic. Nhờ Data Binding, UI có thể tự động cập nhật lại để hiển thị các thay đổi trong dữ liệu. Ngoài ra, Data Binding trong WPF còn hỗ trợ các chiều khác nhau, nghĩa là các thay đổi có thể cập nhật từ UI vào dữ liệu. Kỹ thuật binding trong mô hình MVVM thực sự là một bước tiến mới, thỏa mãn những điều mà hầu hết lập trình viên mong đợi.

Nếu như tìm hiểu về tính năng này, bạn sẽ không ngạc nhiên gì khi nhiều người nói rằng data binding là thành phần cốt lõi tạo nên các cơ chế hoạt động trong WPF. Bạn có thể binding dữ liệu nguồn và đích từ bất kỳ đối tượng nào: như cửa sổ, các control đơn giản như TextBlock cho đến một usercontrol phức tạp.

Tất cả được thực hiện một cách dễ dàng, nhanh chóng, hiệu quả và có thể không cần dùng đến bất kỳ dòng code-behind (C#, VB.NET, ...) nào.

- Data Template

Data Template là kỹ thuật dùng để tạo ra một khuôn mẫu giao diện. Template chỉ được áp dụng cho các Control. Một template trong WPF xác định cách thức và cấu trúc mà dữ liệu hoặc control sẽ được hiển thị ra màn hình.

Nói riêng về Datatemplate, chức năng này giúp cho dữ liệu (thuộc dạng non-visual) được gắn vào một cấu trúc bao gồm một hoặc nhiều thành phần có khả năng hiển thị. Và do đó, dữ liệu sẽ được hiển thị lên cửa sổ một cách trực quan theo ý muốn của lập trình viên. Cũng như Databinding, tính năng này không yêu cầu bạn phải biết trong code-behind của ứng dụng.

- Command

Data Binding và Data Template trong wpf giúp cho người dùng thấy được những gì có trong dữ liệu và có thể cập nhật lại dữ liệu đó. Tuy nhiên để nhận được tương tác từ người dùng và xử lý, WPF cung cấp một tính năng gọi là command. Các command có thể

được xem như dữ liệu và được cung cấp cho người dùng thông qua chức năng binding. Một command binding cho phép bạn tùy ý xác định các phương thức xử lý, phím tắt hoặc thao tác chuột để kích hoạt.

3.7.3. Ưu nhược điểm của mô hình MVVM

- Ưu điểm:

- + Thực hiện Unit testing bây giờ sẽ rất dễ dàng, vì bạn thực sự không phụ thuộc vào view
- + MVVM sẽ tạo sự tương tác hiệu quả giữa designer và developer
- + Tăng khả năng sử dụng lại các thành phần hay việc thay đổi giao diện chương trình mà không cần phải viết lại code quá nhiều
- + Phát triển ứng dụng nhanh, đơn giản, dễ nâng cấp, bảo trì...

- Nhược điểm:

- + Khả năng duy trì khi view có thể gán cả biến và biểu thức, các logic không liên quan sẽ tăng dần theo thời gian, ảnh hưởng đến việc thêm code vào XML
- + Đối với dự án nhỏ việc áp dụng mô hình MVVM gây cồng kềnh, tốn thời gian trong quá trình phát triển. Tốn thời gian trung chuyển dữ liệu của các thành phần
- + Đối với dự án lớn hơn, nó gây khó khăn và mất thời gian để thiết kế các ViewModel
- + Việc liên kết dữ liệu cho tất cả các thành phần gây khó khăn trong việc debug khi cơ sở dữ liệu phức tạp

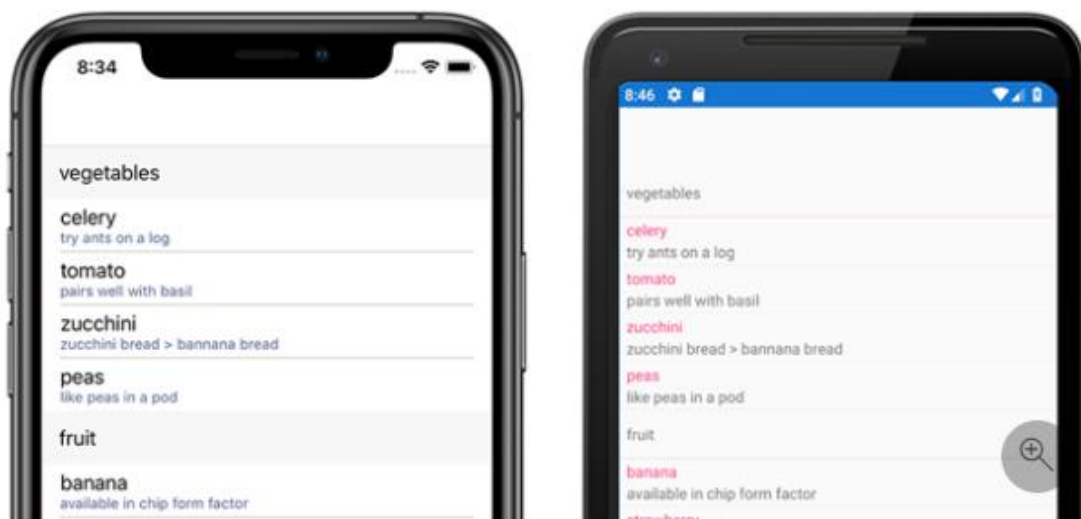
CHƯƠNG 4: ỨNG DỤNG APPTRANSPORT

4.1. Đề tài: Xây dựng ứng dụng APPTRANSPORT

4.1.1. Đặc tả đề tài

- Phần mềm AppTransport như một cuốn từ điển nhỏ về các phương tiện tham gia giao thông trên đường sắt, đường bộ, đường thủy, đường hàng không ... Ở đây người dùng có thể tra cứu một vài thông tin cơ bản như: tên, nơi xuất xứ, hình ảnh, thông tin chi tiết của các phương tiện.
- Dữ liệu, thông tin đầu vào: ở đây sẽ làm về 6 loại phương tiện
 - + Xe motor
 - + Xe máy Việt Nam
 - + Xe hơi
 - + Xe đạp
 - + Xe đạp điện
 - + Máy bay
- Ở mỗi loại phương tiện sẽ có 4 thông tin cơ bản:
 - + Name (Tên phương tiện)
 - + Origin (Xuất xứ)
 - + Details (Thông tin giới thiệu cơ bản)
 - + ImageUrl (Ảnh)
 - + Info (Link rút gọn để truy cập trang web)
- Mục đích, tính năng: Tạo ra 1 cuốn từ điển nhỏ để cho người dùng – những người có nhu cầu tìm kiếm các loại phương tiện giao thông dễ dàng hơn có nơi để tra cứu thông tin. Ở đây ngoài những phương tiện phổ biến như xe máy, xe hơi ... còn có có loại rất đặc biệt như: máy bay phản lực, máy bay chiến đấu, máy bay quân sự...
- Về giao diện dự kiến: XamarinForm
 - + Dựa theo mẫu của 1 cuốn từ điển Anh-Việt
 - + Giao diện sẽ thiết kế theo:

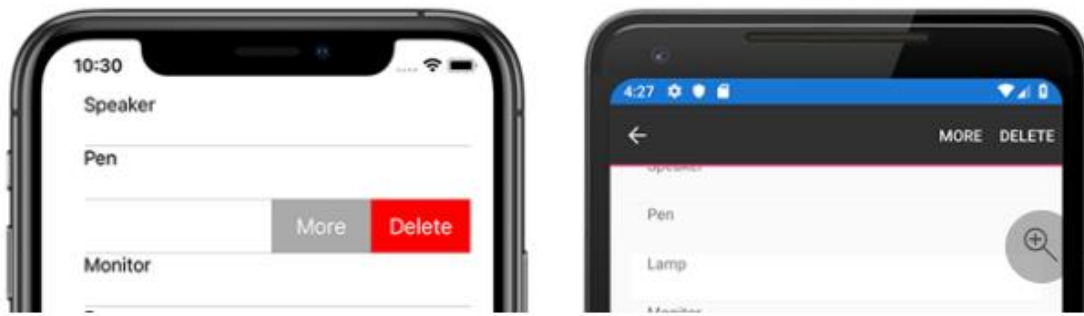
Dạng chia theo Groups: Mỗi loại phương tiện đại diện bởi 1 button và 1 icon image. Khi nhấn vào đó thì sẽ ra list của loại đó. Vd: xe máy thì gồm các loại xe Vison, SH...



Dạng chia theo Cells: Các mục dữ liệu trong ListView được gọi là các ô. Mỗi ô tương ứng với một hàng dữ liệu. Có các ô tích hợp để chọn hoặc bạn có thể xác định ô tùy chỉnh của riêng mình. Cả ô tích hợp và ô tùy chỉnh đều có thể được sử dụng/xác định trong XAML hoặc mã. Các ô tích hợp, chẳng hạn như TextCell và ImageCell, tương ứng với các điều khiển gốc và đặc biệt hiệu quả. TextCell hiển thị một chuỗi văn bản, tùy chọn với văn bản chi tiết. Văn bản chi tiết được hiển thị dưới dạng dòng thứ hai bằng phông chữ nhỏ hơn với màu nhàn. Một ImageCell hiển thị một hình ảnh với văn bản. Xuất hiện dưới dạng TextCell với hình ảnh ở bên trái.



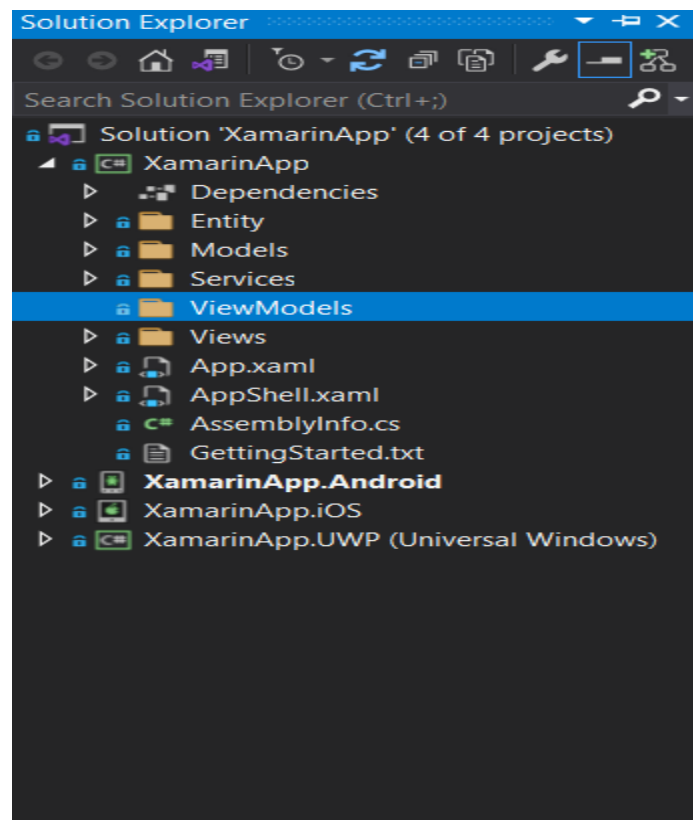
Dạng chia theo Functionality: một vài chức năng cơ bản như thanh tìm kiếm theo tên, xóa hoặc sửa trực tiếp trên giao diện...



4.1.2. Mô hình sử dụng

* Thiết kế và cài đặt trên Mobile App (XamarinForms) – Visual Studio Code

* Cấu trúc:



Hình 45: Cấu trúc của App Phương tiện giao thông

4.2. Code

4.2.1. Models (Transports.cs)

```
namespace XamarinApp.Models
{
    94 references
    public class Transport
    {
        81 references
        public string Name { get; set; }
        67 references
        public string Origin { get; set; }
        67 references
        public string Details { get; set; }
        67 references
        public string ImageUrl { get; set; }
        72 references
        public string Info { get; set; }
    }
}
```

Hình 46: Model Transports.cs

4.2.2. Entity (CarData.cs)

```
public static class CarData
{
    23 references
    public static IList<Transport> Cars { get; private set; }

    0 references
    static CarData()
    {
        Cars = new List<Transport>();

        Cars.Add(new Transport
        {
            Name = "Toyota Avanza",
            Origin = "Nhật Bản",
            Details = "Toyota Avanza là mẫu MPV 7 chỗ của hãng ô tô hàng đầu Nhật Bản, được xếp dưới đàn anh Toyota In
            ImageUrl = "https://upload.wikimedia.org/wikipedia/commons/thumb/e/e0/2022_Toyota_Avanza_1.5_G_Toyota_Safe
            Info = "FYND6S"
        });
    }
}
```

Hình 47: Entity CarData.cs

4.2.3. Entity (BicycleData.cs)

```
2 references
public static class BicycleData
{
    8 references
    public static IList<Transport> Bicycles { get; private set; }

    0 references
    static BicycleData()
    {
        Bicycles = new List<Transport>();

        Bicycles.Add(new Transport
        {
            Name = "Road Txed Road Flat 29 inch",
            Origin = "Trung Quốc",
            Details = "Xe đạp thể thao Road Txed Road Flat 29 inch có kiểu dáng thể thao, năng động với logo TXED chạy
            Info = "VBMMd3",
            ImageUrl = "https://cdn.tgdd.vn/Products/Images/9758/250985/txed-flat-titanium-size-l-glr-min-4.jpeg"
        });
    }
}
```

Hình 48: Entity BicycleData.cs

4.2.4. Entity (ElectricBicycleData.cs)

```
public static class ElectricBicycleData
{
    10 references
    public static IList<Transport> ElectricBicycles { get; private set; }

    0 references
    static ElectricBicycleData()
    {
        ElectricBicycles = new List<Transport>();

        ElectricBicycles.Add(new Transport
        {
            Name = "Asama Ray-EBK-RY2001",
            Origin = "Việt Nam",
            Details = "Nghe tới cái tên Asama chắc hẳn các bạn sẽ nghĩ ngay đến xe đạp asama hay được sử dụng cho các
            Info = "NBsCpX",
            ImageUrl = "https://dailyxedien.vn/wp-content/uploads/2020/03/xe-dap-dien-asama-01-600x600.jpg"
        });
    }
}
```

Hình 49: Entity ElectricBicycleData.cs

4.2.5. Entity (MotorcycleData.cs)

```
public static class MotorcycleData
{
    13 references
    public static IList<Transport> Motorcycles { get; private set; }

    0 references
    static MotorcycleData()
    {
        Motorcycles = new List<Transport>();

        Motorcycles.Add(new Transport
        {
            Name = "Harley-Davidson Sportster",
            Origin = "Mỹ",
            Details = "Sportster S 1250 là chiếc xe đang hot nhất của gia đình Harley Davidson hiện nay , trang bị rất",
            Info = "Z19jjC",
            ImageUrl = "https://upload.wikimedia.org/wikipedia/commons/thumb/f/f1/1957_Harley-Davidson_XL_Sportster.jpg",
        });
    }
}
```

Hình 50: Entity MotorcycleData.cs

4.2.6. Entity (MotorVNData.cs)

```
public static class MotorVNData
{
    12 references
    public static IList<Transport> MotorVNs { get; private set; }

    0 references
    static MotorVNData()
    {
        MotorVNs = new List<Transport>();

        MotorVNs.Add(new Transport
        {
            Name = "Exciter 155 VVA",
            Origin = "Việt Nam",
            Details = "Yamaha Exciter 2022 là mẫu xe côn tay được ưa chuộng nhất tại thị trường Việt Nam với thiết kế",
            ImageUrl = "https://yamaha-motor.com.vn/wp/wp-content/uploads/2021/12/Exciter-155-RC-STD-Mat-Red-004.png",
            Info = "jAvpnk"
        });
    }
}
```

Hình 51: Entity MotorVNData.cs

4.2.7. Entity (PlaneData.cs)

```
public static class PlaneData
{
    13 references
    public static IList<Transport> Planes { get; private set; }

    0 references
    static PlaneData() {
        Planes = new List<Transport>();

        Planes.Add(new Transport
        {
            Name = "Airbus A321-200",
            Origin = "Mỹ",
            Details = "Airbus là công ty con của một công ty có tên viết tắt là EADS (tên đầy đủ "The European Aeronau
            ImageUrl = "https://truongphatlogistics.com/uploads/pictures/2020/9/22/content_5_5-loai-may-bay-dan-dung-
        });
    }
}
```

Hình 52: Entity PlaneData.cs

4.2.8. Services (TransportSearchHandler)

```
namespace XamarinApp.Services
{
    2 references
    public class TransportSearchHandler : SearchHandler
    {
        //Tạo ra 2 đối tượng là list transport và tạo targetitem Navigation
        1 reference
        public IList<Transport> Transports { get; set; }
        1 reference
        public Type SelectedItemNavigationTarget { get; set; }
        Random namerd = new Random();

        0 references
        protected override void OnQueryChanged(string oldValue, string newValue)
        {
            base.OnQueryChanged(oldValue, newValue);
            //string textrd = Convert.ToString((char)namerd.Next(97, 122));
            if (string.IsNullOrEmpty(newValue))
            {
                ItemsSource = null;
            }
            else
            {
                //Sử dụng linq để query ra con vật cần search
            }
        }
    }
}
```

Hình 53: Services TransportSearchHandler.cs

4.2.9. Views (Cars/Bicycles/ElectricBicycles/MotorVNs/Motorcycles/Plane-Page.xaml)

```
<Shell.SearchHandler>
  <services:TransportSearchHandler Placeholder="Nhập gì đó..."
    ShowsResults="true"
    ItemTemplate="{StaticResource TransportSearchTemplate}"
    Transports="{x:Static entity:CarData.Cars}"
    SelectedItemNavigationTarget="{x:Type views:ElectricBicycleDetailPage}"/>
</Shell.SearchHandler>

<CollectionView Margin="20"
  ItemsSource="{x:Static entity:CarData.Cars}"
  ItemTemplate="{StaticResource TransportTemplate}"
  SelectionMode="Single"
  SelectionChanged="OnCollectionViewSelectionChanged"/>
</ContentPage>
```

Hình 54 Views - xxxPage.xaml

- Biding Tương tự cho các Bicycles/ElectricBicycles/MotorVNs/Motorcycles/Plane-Page.xaml
- + Transports: lấy dữ liệu từ entity được truyền vào
- + ItemTemplate: Lấy cấu trúc là TransportSearchTemplate/TransportTemplate (được định nghĩa ở App.xaml)
- + SelectedItemNavigationTarget: event được định nghĩa ở TransportSearchHandler để navigation và truyền nội dung vào trang được navigation đến.
- + SelectionChanged: sự kiện click được định nghĩa ở trang (**transports**).cs

4.2.10. Views (Cars/Bicycles/ElectricBicycles/MotorVNs/Motorcycles/Plane-Page.cs)

```
namespace XamarinApp.Views
{
    ///[XamlCompilation(XamlCompilationOptions.Compile)]
    4 references
    public partial class CarsPage : ContentPage
    {
        0 references
        public CarsPage()
        {
            InitializeComponent();
        }

        0 references
        async private void OnCollectionViewSelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            string carName = (e.CurrentSelection.FirstOrDefault() as Transport).Name;

            //
            await Shell.Current.GoToAsync($"cardetails?name={carName}");
        }
    }
}
```

Hình 55: Views - Page.cs

- Tương tự cho các trang Bicycles/ElectricBicycles/MotorVNs/Motorcycles/Plane-Page.cs
- Xử lý sự kiện click khi click vào cái item trên collectionview – chuyển trang tới details-transport – route được định nghĩa ở App.xaml.cs

4.2.11. Views (Car/Bicycle/ElectricBicycle/MotorVN/Motorcycle/Plane- DetailPage.xaml)

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="XamarinApp.Views.CarDetailPage"
             Title="Car Details">
    <ScrollView>
        <StackLayout Margin="20">
            <Label Text="{Binding Name}"
                  HorizontalOptions="Center"
                  Style="{DynamicResource TitleStyle}"/>

            <Label Text="{Binding Origin}"
                  FontAttributes="Italic"
                  HorizontalOptions="Center"/>

            <Image Source="{Binding ImageUrl}"
                  HeightRequest="200"
                  WidthRequest="200"
                  HorizontalOptions="CenterAndExpand"/>

            <Label Text="{Binding Details}"
                  Style="{DynamicResource BodyStyle}"/>

            <Button Text="OPEN WEBSITE"
                   Margin="20"
                   Background="#546DFE"
                   Clicked="ButtonWebsiteClicked"/>
        </StackLayout>
    </ScrollView>
</ContentPage>
```

Hình 56: Views DetailPage.xaml

- Tương tự cho các trang Bicycle/ElectricBicycle/MotorVN/Motorcycle/Plane-
DetailPage.xaml

4.2.12. Views (Car/Bicycle/ElectricBicycle/MotorVN/Motorcycle/Plane - DetailPage.xaml.cs)

```
namespace XamarinApp.Views
{
    ///[XamlCompilation(XamlCompilationOptions.Compile)]
    ///[QueryProperty("Info", "info")]
    [QueryProperty("Name", "name")]
    5 references
    public partial class CarDetailPage : ContentPage
    {
        string transports;
        0 references
        public string Name
        {
            set
            {
                loadTransport(value);
            }
        }
        0 references
        public string Info
        {
            set
            {
                loadTransport(value);
            }
        }
    }
    3 references
}
```

Hình 57 Views DetailPage.xaml.cs

```
private void loadTransport(string name)
{
    try
    {
        //Transport transport = CarData.Cars.FirstOrDefault(a => a.Name == name);
        Transport transport = CarData.Cars.FirstOrDefault(a => a.Name == name);
        BindingContext = transport;
        transports = transport.Info;
        Console.WriteLine(transports);
    }
    catch (Exception)
    {
        Console.WriteLine("Khong the tai moi khi click!");
    }
};

0 references
public CarDetailPage()
{
    InitializeComponent();
}

0 references
public static bool IsUnicode(string input)
{
    return Encoding.ASCII.GetByteCount(input) != Encoding.UTF8.GetByteCount(input);
}
```

```

0 references
private void ButtonWebsiteClicked(object sender, EventArgs e)
{
    string url;
    //if (IsUnicode(transports))
    {
        url = "https://bom.so/" + transports;
    }

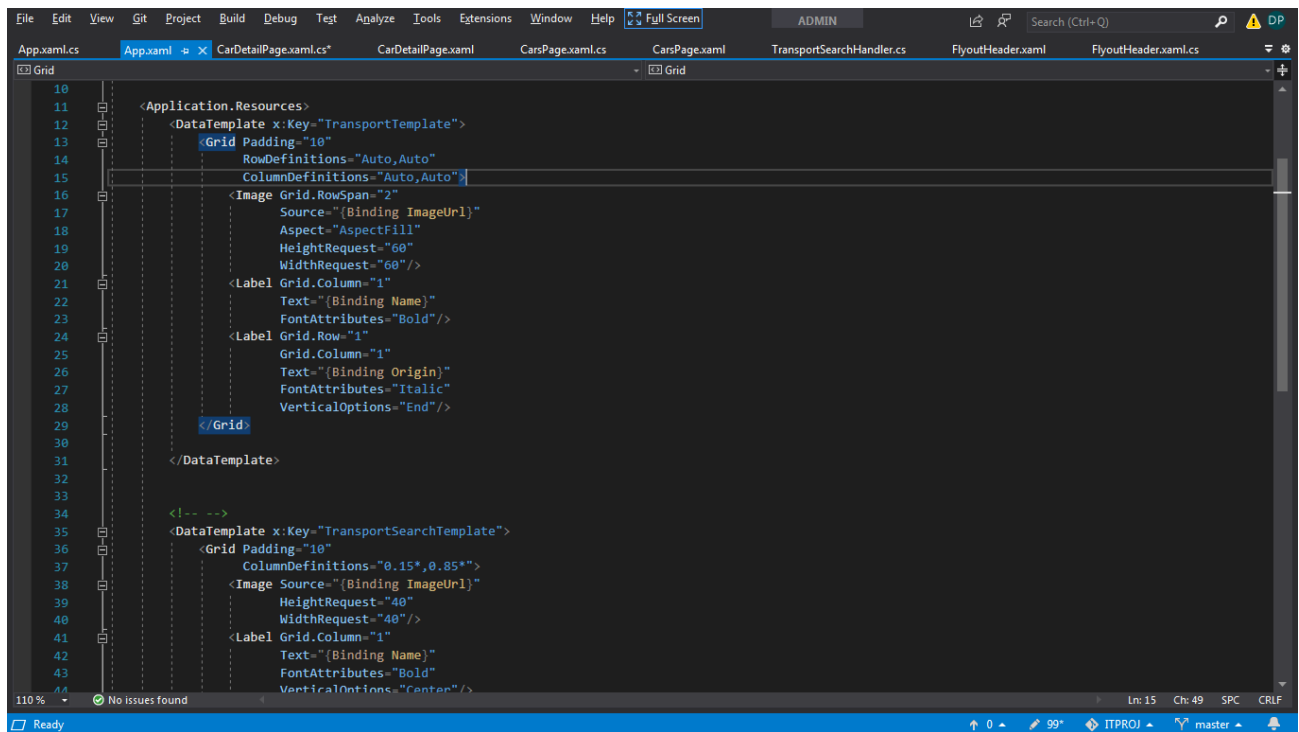
    Xamarin.Essentials.Launcher.OpenAsync(url);
}
}

```

- Tương tự cho các trang Bicycle/ElectricBicycle/MotorVN/Motorcycle/Plane - DetailPage.xaml.cs
- Trang xử lý lấy thông tin bằng linq và binding – xử lý sự kiện cho button open website

4.2.13. App.xaml

- Định nghĩa hiển thị cho *TransportPage* bao gồm TransportTemplate và TransportSearchTemplate



Hình 58: App.xaml

4.2.13. AppShell.xaml



Hình 59: AppShell.xaml

- Tương tự cấu trúc cho các FlyItem được định nghĩa sẵn trước đó.

4.2.14. AppShell.xaml.cs

```
1 reference
void RegisterRoutes()
{
    Routes.Add("motorVNdetails", typeof(MotorVNDetailPage));
    Routes.Add("cardetails", typeof(CarDetailPage));
    Routes.Add("bicycledetails", typeof(BicycleDetailPage));
    Routes.Add("electricBicycledetails", typeof(ElectricBicycleDetailPage));
    Routes.Add("motorcycledetails", typeof(MotorcycleDetailPage));
    Routes.Add("planedetails", typeof(PlaneDetailPage));

    foreach (var item in Routes)
    {
        Routing.RegisterRoute(item.Key, item.Value);
    }
}
```

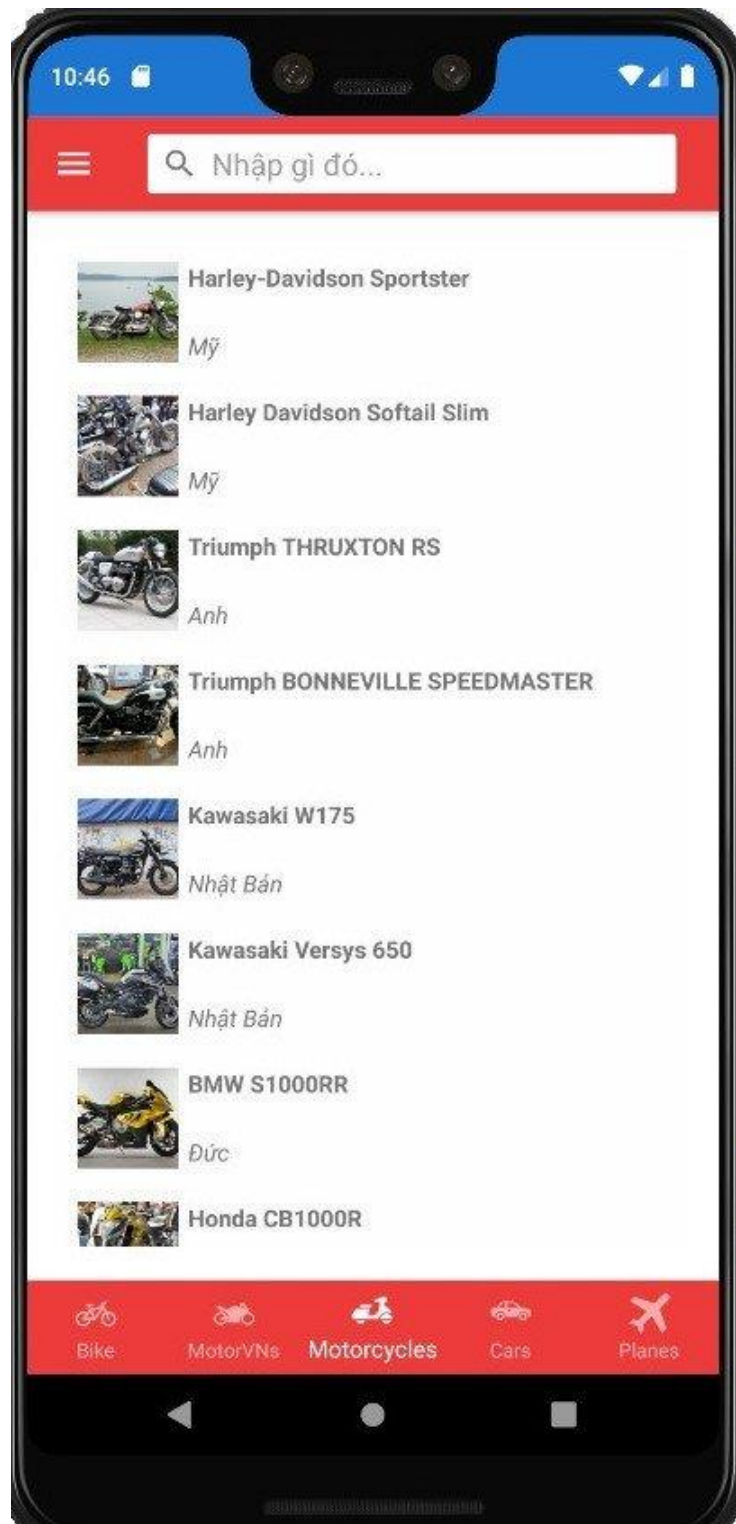
Hình 60: AppShell.xaml.cs

4.3. Chạy thử

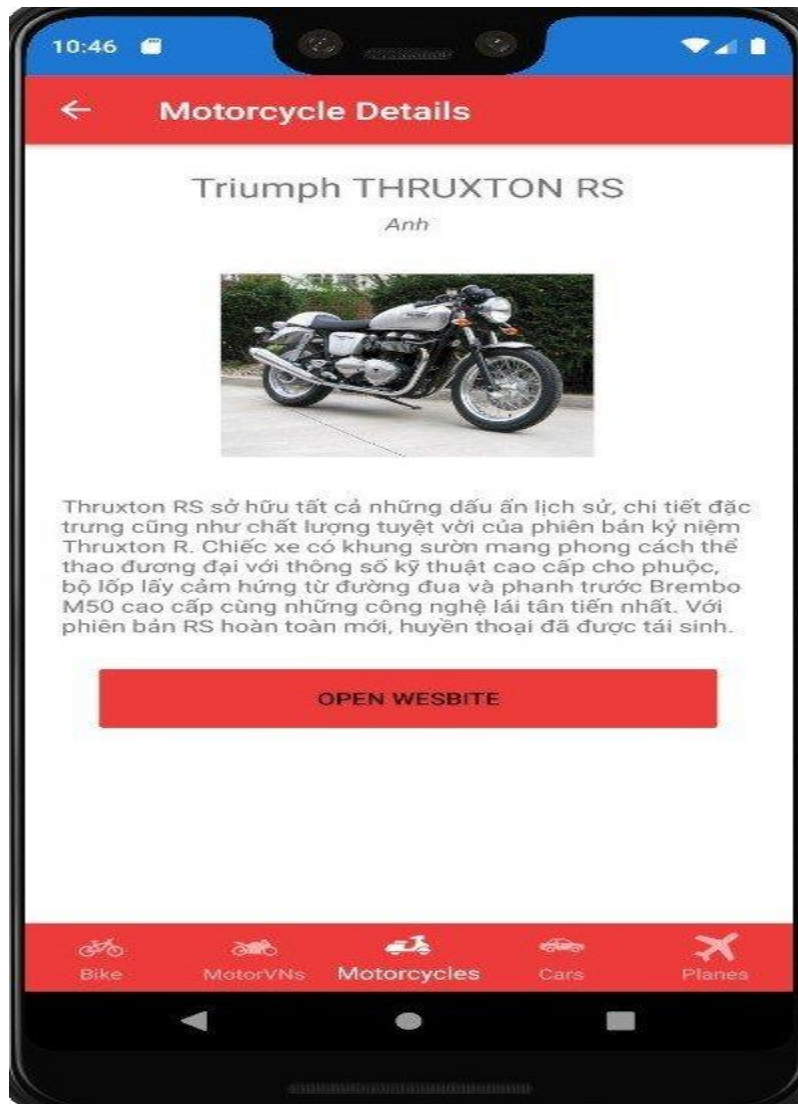
- Giao diện:
- + Giao diện chính khi mở ứng dụng.



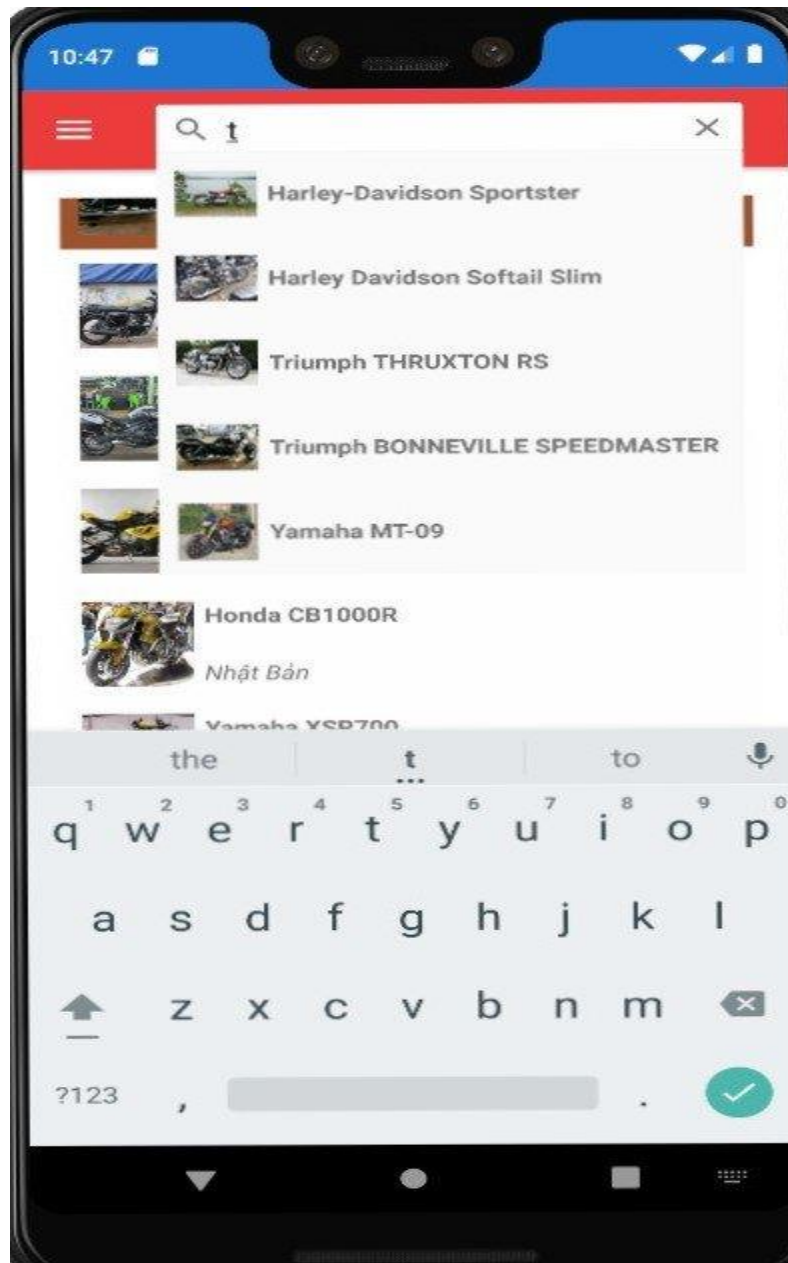
- Giao diện khi mở trang khác



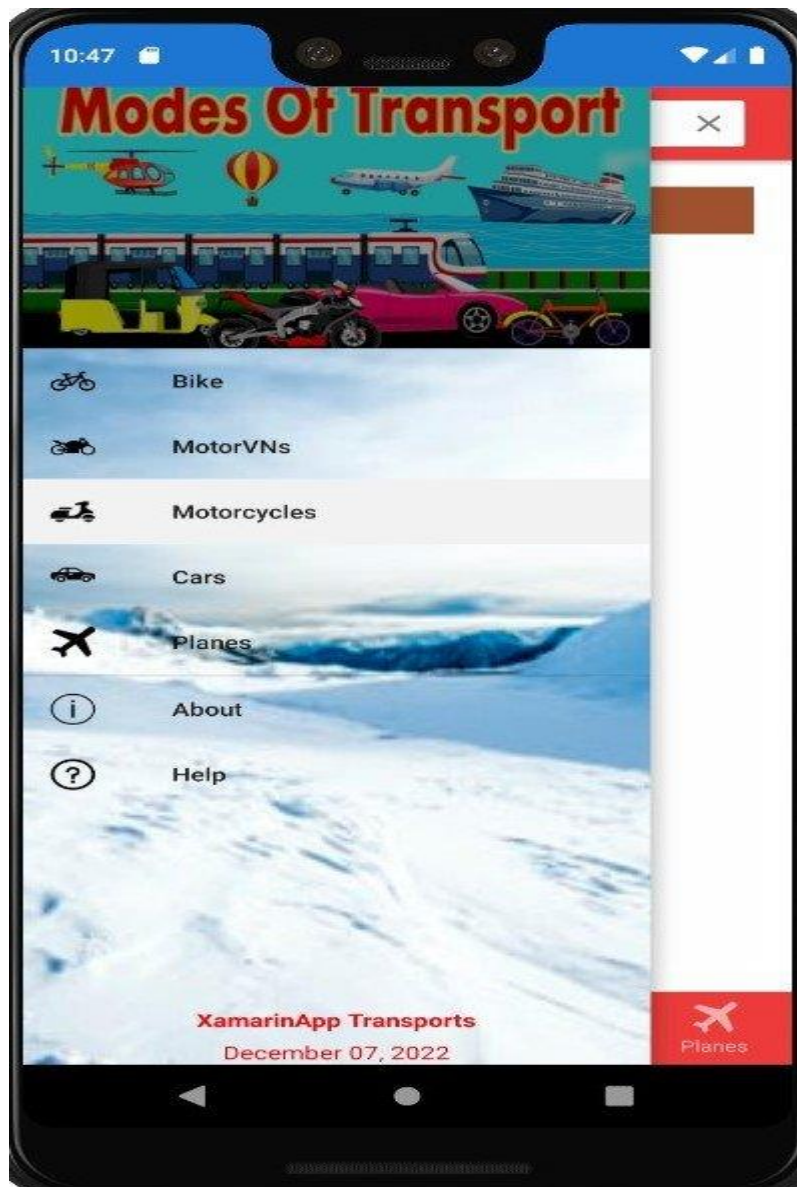
- Giao diện 1 trang detail

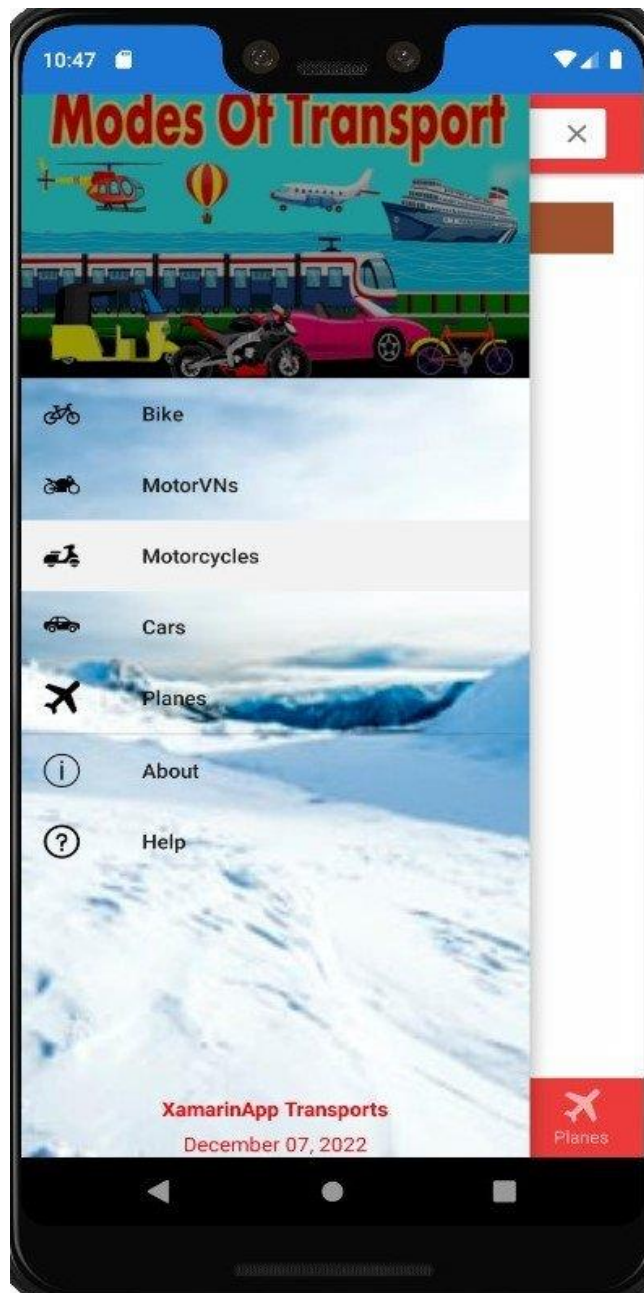


- Giao diện khi search



- Navigation bar





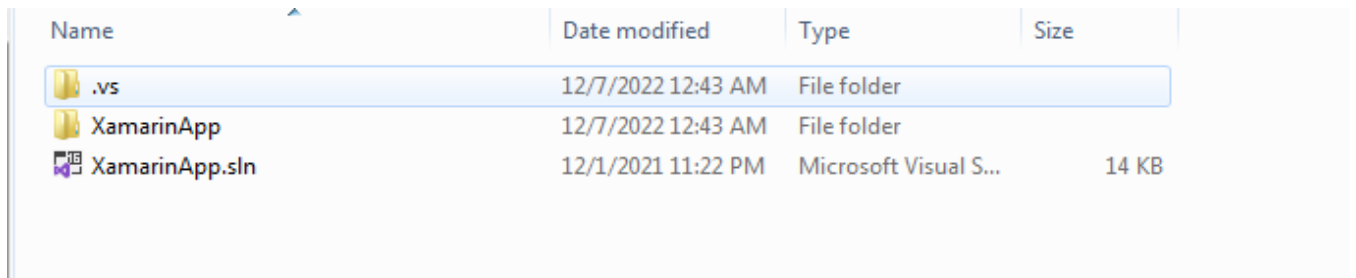
CHƯƠNG 5: HƯỚNG DẪN SỬ DỤNG

Bước 1: Vô đường link sau để tải đồ án về:

https://drive.google.com/drive/folders/1mfXAAZij_cxAglzMVjRJndGUZvR_1o9i?usp=share_link

Hình 61: link drive download

Bước 2: Sau khi tải về vào thử mục (XamarinApp)– mở file XamarinApp.sln

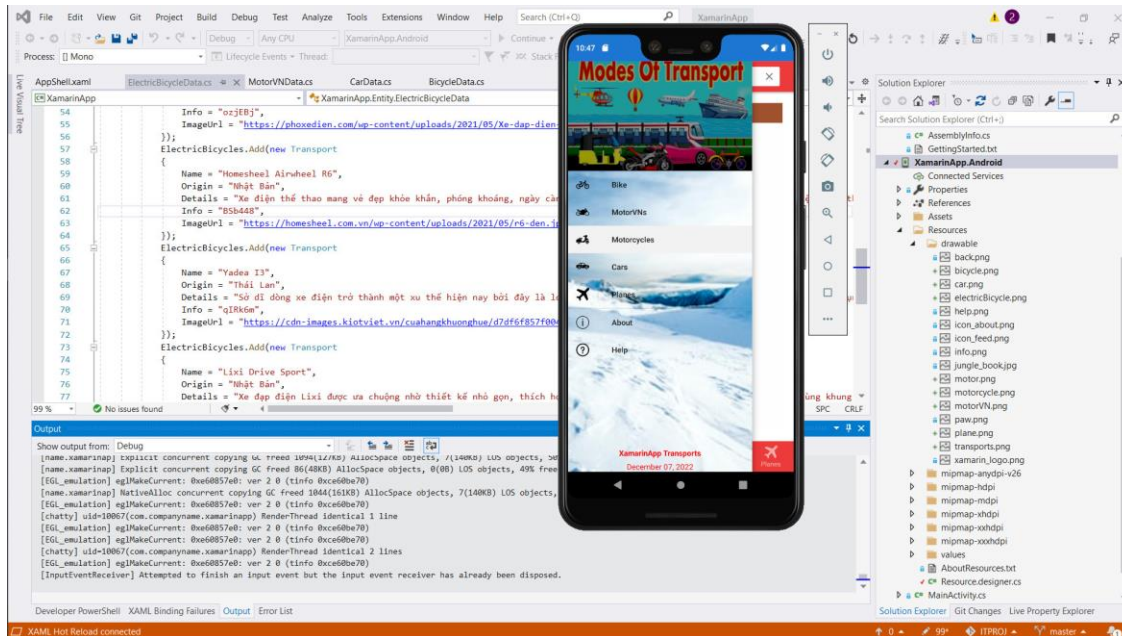


Name	Date modified	Type	Size
.vs	12/7/2022 12:43 AM	File folder	
XamarinApp	12/7/2022 12:43 AM	File folder	
XamarinApp.sln	12/1/2021 11:22 PM	Microsoft Visual S...	14 KB

Hình 62: Giải nén và Mở file download

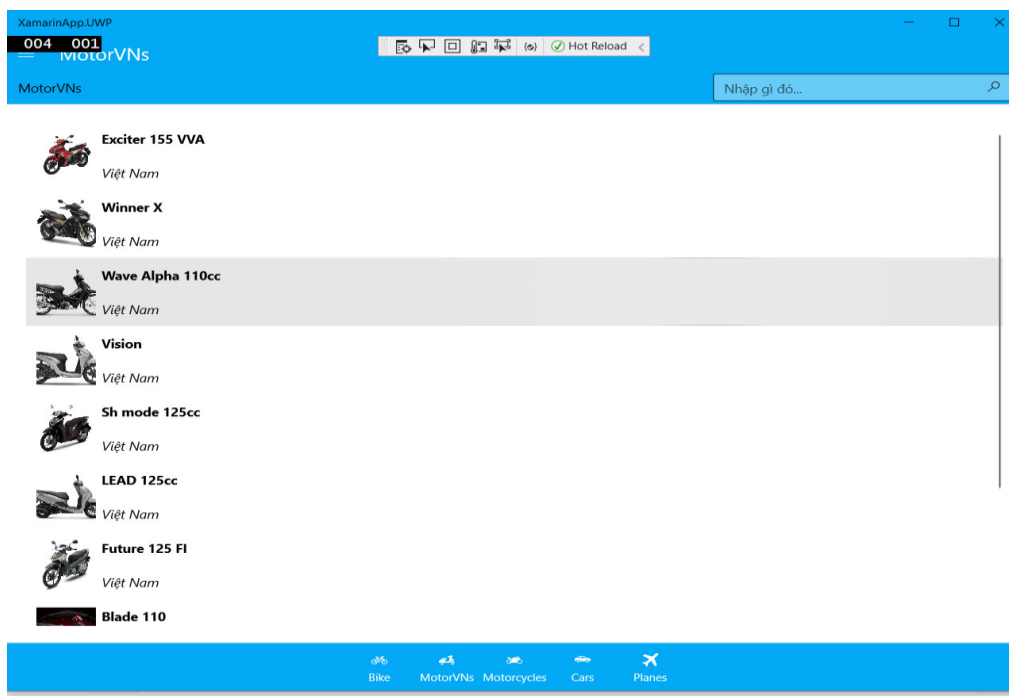
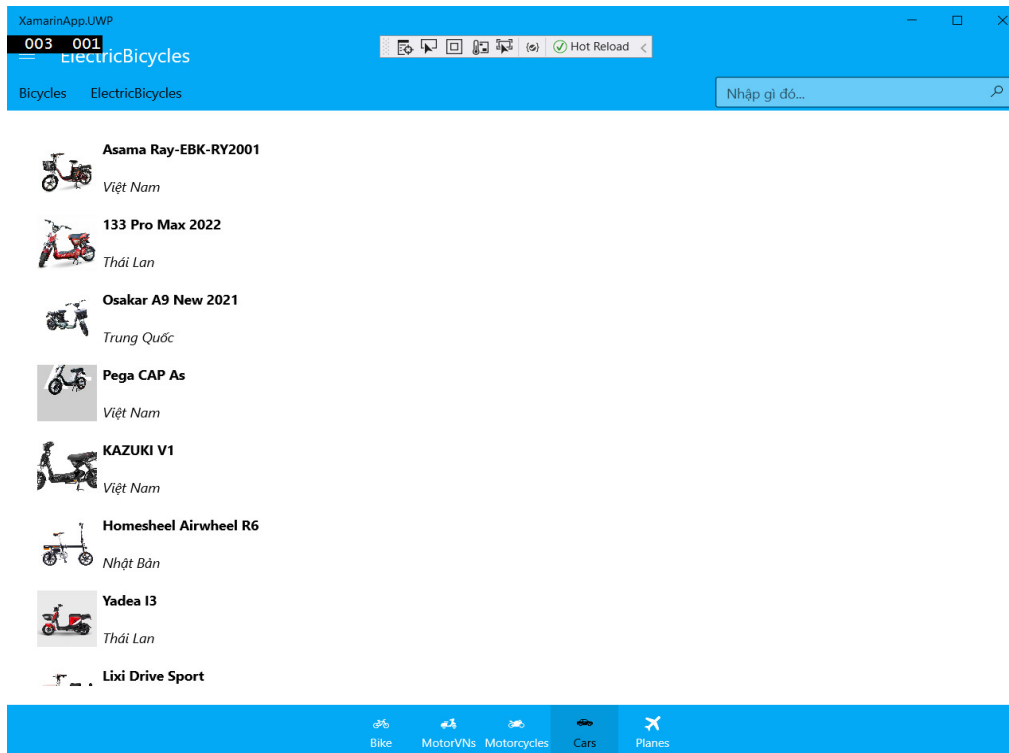
Bước 3: Chờ khoảng 1,2 phút cho chương trình đồng bộ, update cái package cần thiết rồi chạy chương trình.

- Kết quả đạt được tương tự



Hình 63: Chạy file và kết quả

Chạy thử với Window được kết quả:



CHƯƠNG 6: TỔNG KẾT

6.1. Ưu điểm

- Đồ án có giao diện đơn giản, dễ sử dụng.
- Đồ án có các tính năng của một app đơn giản, có thể được dùng để giải trí sau những giờ làm việc mệt mỏi, có thể phát triển mở rộng.
- Code sạch dễ đọc dễ hiểu vì sử dụng gần như theo mô hình.

6.2. Nhược điểm

- Phần demo cho App Transport còn chưa có nhiều tính năng (như thêm trực tiếp data mới vào hay xóa thông tin bị nhầm lẫn đi) , người dùng chỉ sử dụng 1 số tính năng đơn giản được tạo sẵn còn lại phải view trên website

6.3. Ý tưởng phát triển

- Nếu có thời gian và cơ hội chúng em sẽ tiếp tục phát triển thêm những event mới và nhiều đối tượng hơn nữa cho ứng dụng như phát triển thành ứng dụng mà ở đó chúng ta có thể có các list đồ điện tử, động vật, lịch sử và nhiều hơn nữa khi click vào động vật thì xuất hiện các lớp như mèo, chó, gấu, voi → khi click vào lớp voi thì tiếp tục ra các item của voi và có thể search được chú voi mình muốn tìm kiếm. Mình có thể phát triển app như thế này để phục vụ cho đối tượng trẻ em

CHƯƠNG 7: TÀI LIỆU THAM KHẢO

Tài liệu 1: TOPDEV

<https://topdev.vn/blog/xamarin-la-gi/>

Tài liệu 2: DOTNET

<http://dotnetguru.org/xamarin-la-gi/>

Tài liệu 3: Xamarin là gì?

<https://www.youtube.com/watch?v=gSGr2jLWGQI>

Tài liệu 4: hocox

https://www.youtube.com/channel/UChbpU0RV-r_u5qvKFYucnIg

Tài liệu 5: jamesmontemagno

<https://www.youtube.com/user/jamesmontemagno>

Tài liệu 6: học lập trình tiếng việt

<http://duyanhpham.com/tai-lieu-hoc-lap-trinh-xamarin-tieng-viet/>

Tài liệu 7:

<https://en.wikipedia.org/wiki/Xamarin>

Tài liệu 8: nhanhoa

<https://nhanhoa.com/tin-tuc/mvvm-la-gi.html>