

Xamarin.Forms

Project : “Δημιουργία Cross-platform ηλεκτρονικού καταστήματος Βιβλίων”

One C# code, multiple platforms



Android



iOS



Windows Phone



Windows 10

→ 1. Παρουσίαση του Xamarin.Forms: Εγκατάσταση, XAML, Pages, Layouts

1. Παρουσίαση του Xamarin.Forms: Εγκατάσταση, XAML, Pages, Layouts

1.1. Η επανάσταση του Xamarin.Forms

Η εταιρία Xamarin ιδρύθηκε το 2011 από τους μηχανικούς του project mono, με σκοπό τη δημιουργία εργαλείων που θα επέτρεπαν την ανάπτυξη εφαρμογών που εκτελούνται σε πολλές πλατφόρμες, όπως Android, IOS και Windows με διαμοιρασμό κώδικα της γλώσσας C#.

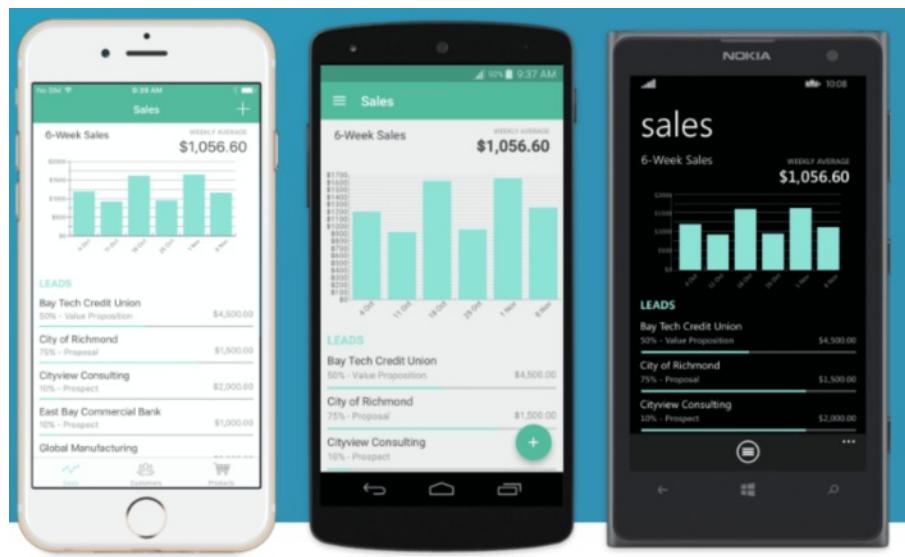
Πλέον, οι χρήστες δεν απαιτείται να μάθουν πολλές τεχνολογίες για να αναπτύξουν την ίδια εφαρμογή σε διαφορετικές πλατφόρμες. Για παράδειγμα δεν απαιτείται η εκμάθηση:

- Java για να αναπτύσσουν εφαρμογές Android στο Android Studio (ή στο Eclipse),
- Objective C ή SWIFT για να αναπτύσσουν εφαρμογές iOS
- WinForms, WPF ή Universal Windows Apps για να αναπτύσσουν εφαρμογές Windows 10/Phone.

Με τη γνώση της C#, της περιγραφικής γλώσσας XAML και του IDE Visual Studio γράφουν μια φορά τον κώδικα και τον διαμοιράζουν σε όλες τις πλατφόρμες. Σχεδόν το 95% του κώδικα μπορεί να διαμοιραστεί με τη χρήση του Xamarin.Forms.

Η εταιρία αρχικά ανέπτυξε τις εκδόσεις Xamarin.Android και Xamarin.iOS που επιτρέπουν το χρήστη να αναπτύσσει εφαρμογές αποκλειστικά για τις εν λόγω πλατφόρμες με κώδικα C#. Έτσι για παράδειγμα το Xamarin.Android επιτρέπει το χρήστη να εκμεταλλευτεί όλες τις δυνατότητες (controls, layouts, sensors κτλ) της βιβλιοθήκης του Android SDK.

Αργότερα προέκυψε το Xamarin.Forms που πλέον επιτρέπει την ανάπτυξη ενός κώδικα-μιας βιβλιοθήκης, που μετασχηματίζεται σε πολλές πλατφόρμες. Έτσι για παράδειγμα χρησιμοποιούμε τα ίδια layouts, controls κτλ τα οποία μέσα από τα εργαλεία του Xamarin μετασχηματίζονται σε στοιχεία που υποστηρίζει το Android και το iOS.



Εικόνα 1: Xamarin.Forms-Μια εφαρμογή μετασχηματίζεται σε πολλές πλατφόρμες (src: xamarin.com)

1.2. Το project “UniBook: A Cross-platform ebook store”

Στην παρούσα σειρά εγχειριδίων θα επιχειρήσουμε να αναπτύξουμε σταδιακά τη γνώση μας, ξεκινώντας από τις βασικές έννοιες σχεδίασης εφαρμογών κινητού. Τελικός στόχος είναι βήμα προς βήμα να δημιουργήσουμε μία εφαρμογή κινητού για πολλές πλατφόρμες, η οποία θα αφορά ένα ηλεκτρονικό κατάστημα ψηφιακών βιβλίων, το UniBook.

Θα ξεκινήσουμε από μια πρόχειρη σχεδίαση του layout, της πλοήγησης και βασικών controls που θα χρησιμοποιηθούν.

Έπειτα θα μελετήσουμε τον τρόπο αποθήκευσης δεδομένων μέσα σε μια Βάση Δεδομένων SQLite και τη λήψη δεδομένων με την αξιοποίηση δικτυακών API, όπως το GoodReads API.

Τέλος, θα μάθουμε για τη σύνδεση σε υπηρεσίες cloud, όπως το Microsoft Azure, η διασύνδεση δεδομένων, το animation και την τελική δημοσίευση της εφαρμογής μας.

Θα παρατηρήσετε ότι στα εγχειρίδια περιπλέκεται η Ελληνική γλώσσα με αγγλικούς όρους. Αυτό έγινε σκόπιμα, προκειμένου ο αναγνώστης να εξοικειωθεί με τους τεχνικούς όρους που θα συναντάει διαρκώς στο διαδίκτυο.

Για τυχόν απορίες ανατρέξτε στο γλωσσάρι, όπουν θα βρείτε τη μετάφρασή τους.

Tags: #Xamarin.Forms, #project_UniBook

1. Παρουσίαση του Xamarin.Forms: Εγκατάσταση, XAML, Pages, Layouts

1.3. Προαπαιτούμενα

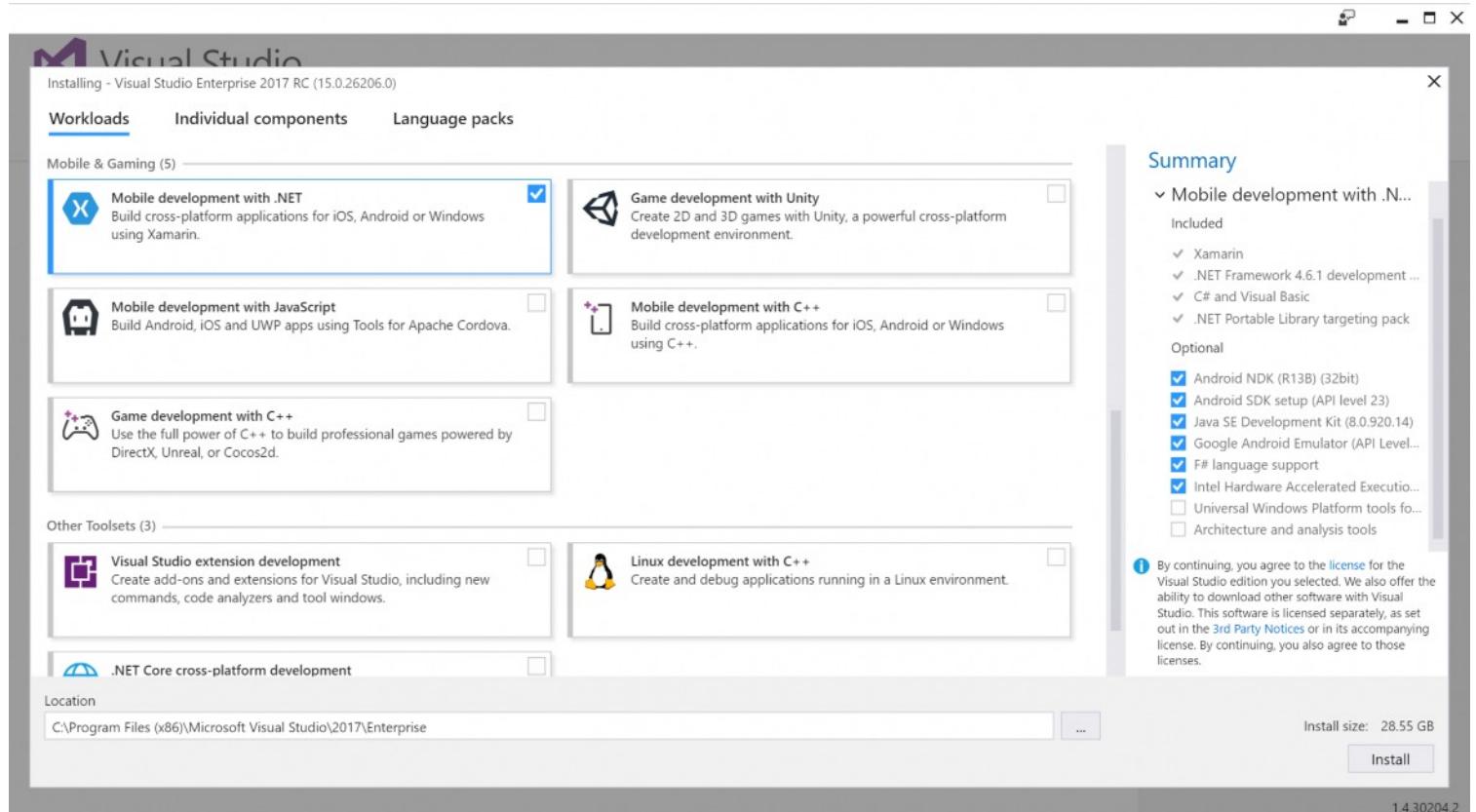
Στα πλαίσια των οδηγών πρέπει να εγκαταστήσουμε τα παρακάτω λογισμικά:

- **Pencil:** Ένα αξιόλογο εργαλείο προτυποποίησης, στο οποίο θα σχεδιάσουμε το User Interface της εφαρμογής μας προτού πιάσουμε κώδικα. Θα το βρείτε εδώ: <http://pencil.evolus.vn/>
- **Visual Studio 2017 Community edition:** Ένα από τα πιο δημοφιλή περιβάλλοντα IDE για την ανάπτυξη εφαρμογών desktop, web και mobile. Περιλαμβάνει τα εργαλεία του Xamarin και θα το βρείτε εδώ: <https://www.visualstudio.com/downloads/>

1.4. Εγκατάσταση

Αφού μεταφορτώσετε το Visual Studio 2017 Community edition, το εκτελείτε και απλά επιλέγετε τις ενότητες που σας ενδιαφέρουν. Για να συμπεριλάβετε το Xamarin επιλέξτε το “Mobile development with .NET”.

Η διαδικασία εγκατάστασης είναι πολύ απλή, όμως αν χρειαστείτε βοήθεια, εδώ θα βρείτε έναν πολύ χρήσιμο οδηγό: <https://www.youtube.com/watch?v=BDdjG--LmhE>



Εικόνα 2: Εγκατάσταση Visual Studio 2017 Community edition

1.5. XAML

Η σχεδίαση του User Interface μπορεί να γίνει είτε με τη γλώσσα C#, είτε με την περιγραφική γλώσσα XAML είτε με συνδυασμό των δύο. Στο project θα χρησιμοποιήσουμε και τις δύο γλώσσες, τη XAML και λιγότερο τη C# για το επίπεδο του User Interface και τη C# για τα επίπεδα της business logic και του data access.

Οσοι δεν γνωρίζουν τη C# μπορούν να βρουν ένα καταπληκτικό βιβλίο εδώ: <http://www.csharpcourse.com/>. Η XAML είναι μια πανίσχυρη περιγραφική γλώσσα με την οποία μπορούμε να σχεδιάσουμε το γραφικό περιβάλλον της εφαρμογής. Ουσιαστικά πρόκειται για ένα xml αρχείο με κατάληξη .xaml. Το αρχείο ξεκινάει με τον εξής κώδικα:

```
<?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="XamlSamples.HelloXamlPage">
</ContentPage>
```

Tags: #Προαπαιτούμενα, #Εγκατάσταση, #XAML

1. Παρουσίαση του Xamarin.Forms: Εγκατάσταση, XAML, Pages, Layouts

Αυτό που χρειάζεται να γνωρίζουμε είναι το x:Class, που δείχνει το Namespace και την κλάση που δημιουργείται από αυτό το XAML αρχείο. Σε αυτή την κλάση μπορούμε να γράψουμε κώδικα C# και να υλοποιήσουμε το business process. Σε περίπτωση που χρειαστεί να δηλώσουμε ένα νέο Namespace απλά προσθέτουμε:

```
xmlns:local="clr-namespace:UniBook"
```

Ας δούμε πως μπορούμε να δημιουργήσουμε μια ετικέτα κειμένου με δύο διαφορετικούς τρόπους: τη XAML και τη C#.

XAML-MainPage.xaml

```
<Label Text="Welcome to Xamarin Forms!"  
      VerticalOptions="Center"  
      HorizontalOptions="Center" />
```

C#-MainPage.xaml.cs

```
Label label = new Label();  
label.Text = "Welcome to Xamarin Forms!";  
label.VerticalOptions = LayoutOptions.Center;  
label.HorizontalOptions = LayoutOptions.Center;
```

Με το `<Label` δηλώνουμε ότι επιθυμούμε μια ετικέτα κειμένου. Στο τέλος κλείνουμε τη δήλωση με το `/>`.

Ενδιάμεσα εισάγουμε τις ιδιότητες με τη μορφή Ιδιότητα=Τιμή . Έτσι με το `Text="Welcome"` δηλώνουμε ότι το κείμενο της ετικέτας είναι Welcome. Το `VerticalOptions="Center"` δηλώνει Κάθετη Στοίχιση στο κέντρο, ενώ το `HorizontalOptions="Center"` δηλώνει οριζόντια στοίχιση στο κέντρο. Άλλες γνωστές ιδιότητες της ετικέτας κειμένου είναι οι εξής:

- `HorizontalTextAlignment="Center"`
- `Rotation="-15"`
- `IsVisible="true"`
- `FontSize="Large"`
- `FontAttributes="Bold"`
- `TextColor="Aqua"`

Παρατηρούμε ότι ενώ στη γλώσσα C# γράφαμε `VerticalOptions=LayoutOptions.Center` στη XAML γράφουμε `VerticalOptions="Center"`. Παραδείγματα αποκοπής επιπρόσθετων λέξεων, όπως το `LayoutOptions` συμβαίνουν συχνά στη XAML και μας διευκολύνουν ώστε να γράψουμε πιο γρήγορα την τιμή της ιδιότητας. Οι κλάσεις που περιλαμβάνουν την ιδιότητα αυτή προκύπτουν από την υπερκλάση `TypeConverters`.

1.6. Pages

Η κλάση Page είναι ένα οπτικό αντικείμενο που απασχολεί το μεγαλύτερο μέρος της οθόνης και περιέχει ένα μόνο παιδί. Η Xamarin.forms διαθέτει τα παρακάτω Pages:

- `ContentPage`: Αποτελεί μια μόνο όψη
- `MasterDetailPage`: Προβάλει δύο όψεις, μία Master και μία Detail
- `NavigationPage`: Χειρίζεται αποτελεσματικά την πλοήγηση μεταξύ σελίδων
- `TabbedPage`: Επιτρέπει την πλοήγηση μεταξύ θυγατρικών σελίδων με τη χρήση tabs
- `TemplatedPage`: Προβάλει περιεχόμενο σε ολόκληρη την οθόνη
- `CarouselPage`: Επιτρέπει την αλλαγή σελίδων με τη μορφή gallery (που κινούμε σέρνοντας το δάχτυλό μας)

Tags: #Label, #Pages

1. Παρουσίαση του Xamarin.Forms: Εγκατάσταση, XAML, Pages, Layouts

1.7. Τα βοηθητικά projects

Στα πλαίσια του παρόντος μαθήματος αναπτύχθηκαν δύο συνοδευτικά projects. Το ένα περιέχει τα τεχνικά στοιχεία που μαθαίνουμε σε κάθε εγχειρίδιο, ενώ το δεύτερο περιέχει το project του ηλεκτρονικού καταστήματος που αναπτύσσουμε.

Θα βρείτε τα project στο github και συγκεκριμένα στη διεύθυνση:

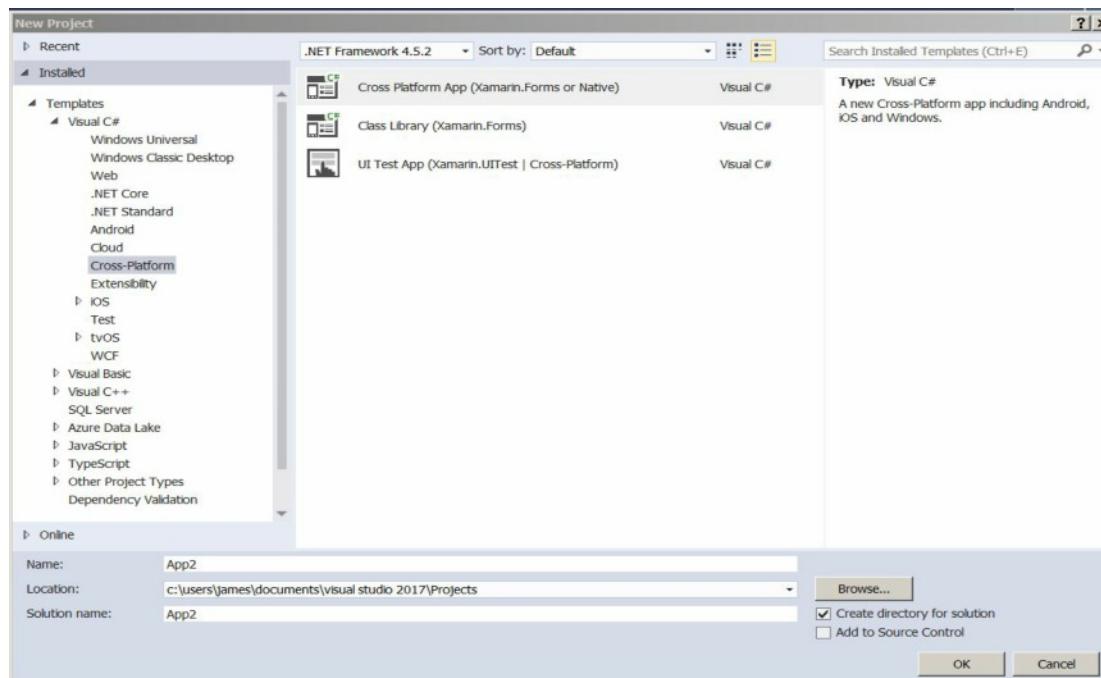
<https://github.com/iwizu/LearnXamarinForms>

Κάθε φάκελος περιέχει την αρίθμηση του εκάστοτε μαθήματος που αφορά.

1.8. Ξεκινώντας ένα νέο project

Για να δημιουργήσουμε μια νέα εφαρμογή, αφού έχουμε εγκαταστήσει το Visual Studio 2017 με υποστήριξη Xamarin, το ανοίγουμε και εκτελούμε τα εξής βήματα:

- Επιλέγουμε File→New Project
- Έπειτα επιλέγουμε από την αριστερή λίστα Cross-platform και από τη δεξιά λίστα Cross-platform App δίνοντας το όνομα της εφαρμογής στο πεδίο “Name:”.



- Έπειτα στο παράθυρο που θα εμφανιστεί επιλέγουμε “Blank App” και ως Code Sharing Strategy το Portable Class Library (PCL) που είναι και ο πιο συχνός τρόπος διαμοιρασμού κώδικα.

Η διαφορά τους είναι η εξής: Με το Shared project οργανώνουμε τον κώδικα χρησιμοποιώντας #if directives για να διαφοροποιήσουμε τον κώδικα ανά πλατφόρμα. Με το Portable Class Library (PCL) χρησιμοποιούμε Interfaces για να διαφοροποιήσουμε τον κώδικα ανάλογα με την κάθε πλατφόρμα. Για περισσότερες πληροφορίες δείτε εδώ: https://developer.xamarin.com/guides/cross-platform/application_fundamentals/code-sharing/.

Αφού φορτώσει το project μας, παρατηρούμε στο δεξί μέρος το Solution explorer. Περιέχει όλη τη δομή των αρχείων του project με τέσσερις ενότητες:

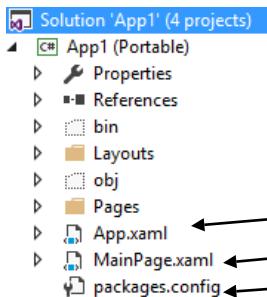
- App1 (Portable): Περιέχει τον κοινό κώδικα
- App1.Android: Ο κώδικας που θα χρησιμοποιηθεί στο Android

Tags: #github, #New_Project

1. Παρουσίαση του Xamarin.Forms: Εγκατάσταση, XAML, Pages, Layouts

- App.iOS: Ο κώδικας που θα χρησιμοποιηθεί στο iOS
- App.UWP: Ο κώδικας που θα χρησιμοποιηθεί στο Universal Windows Apps

Η δομή του project είναι η παρακάτω:



- 1.Το πρώτο αρχείο που φορτώνεται στην εφαρμογή
- 2.Το App φορτώνει την πρώτη σελίδα MainPage
Φιλοξενεί τα πρόσθετα πακέτα της εφαρμογής

Ανοίγουμε το MainPage κάνοντας διπλό κλικ και γράφουμε τον παρακάτω κώδικα:

```
<Label Text="Welcome to Xamarin Forms!"  
      VerticalOptions="Center"  
      HorizontalOptions="Center" />  
<Label Text="---PAGES---"  
      VerticalOptions="Start"  
      HorizontalOptions="StartAndExpand" />  
<Button Text="Pages" Clicked="Button_Clicked"  
      VerticalOptions="Start"  
      HorizontalOptions="StartAndExpand" />
```

Στην αρχή τοποθετούμε δύο ετικέτες κειμένου και μετά ένα κουμπί Button. Στο κουμπί θέτουμε την ιδιότητα κειμένου ως “Pages”. Η ιδιότητα Clicked παίρνει ως τιμή το όνομα της μεθόδου που χειρίζεται το συμβάν Πάτημα κουμπιού. Έτσι αν κάνουμε κλικ στο κουμπί θα εκτελεστεί η μέθοδος Button_Clicked:

```
private void Button_Clicked(object sender, EventArgs e)  
{  
    Navigation.PushAsync(new AContentPage());  
}
```

Η μέθοδος μας πλοηγεί σε μια νέα σελίδα. Παρακάτω θα μάθουμε περισσότερα για την πλοήγηση στο περιβάλλον της Xamarin.Forms.

1.9. Πλοήγηση στις σελίδες

Σπάνια έχουμε μια εφαρμογή που απασχολεί μία μόνο οθόνη. Συνήθως οι εφαρμογές περιέχουν πολλές οθόνες και ένα σύστημα πλοήγησης μεταξύ τους. Οπως είδαμε το App.xaml είναι το πρώτο στοιχείο που εκτελείται κατά την έναρξη της εφαρμογής. Ο πιο απλός τρόπος για να μεταβούμε από το App.xaml στη MainPage.xaml (ή όποια άλλη σελίδα επιθυμούμε) είναι ο παρακάτω κώδικας στον κατασκευαστή του App, αμέσως μετά τη γραμμή InitializeComponent(); :

```
MainPage = new NavigationPage(new MainPage());
```

Το NavigationPage είναι μια κλάση που χειρίζεται την πλοήγηση μεταξύ παραθύρων. Κάθε φορά που καλείται για να μας ανακατευθύνει σε μια νέα σελίδα, τοποθετεί σε στοίβα όλο το ιστορικό κλήσεων και μας επιτρέπει από κάθε σημείο να επιστρέψουμε πίσω στις προηγούμενες. Από τη MainPage και έπειτα μπορούμε να συνεχίσουμε τη μετάβαση σε άλλες σελίδες με την πιο απλή εντολή :

```
Navigation.PushAsync(new AContentPage()); //Δηλαδή βάλε στη στοίβα μια νέα κλάση της σελίδας AContentPage
```

Tags: #project_Structure, #Navigation

1.10. Page: Η συνέχεια

Έχοντας ανοίξει ένα νέο project και εκτελώντας πλέον πραγματικό κώδικα μπορούμε να δούμε στην πράξη τις διάφορες κλάσεις που υλοποιούν την κλάση Page.

1.10.1. ContentPage

Η ContentPage προβάλει μία μόνο όψη (View), η οποία συνήθως είναι ένα StackLayout ή ένα ScrollView. Στο project μας θα χρησιμοποιήσουμε ένα ContentPage για να φτιάξουμε ένα μενού επιλογών μετάβασης σε άλλες κλάσεις τύπου Page.

- Κάντε δεξί κλικ στο project App και επιλέξτε “Add new folder” και προσθέστε ένα φάκελο με το όνομα Pages. Μέσα στο φάκελο θα προσθέσουμε όλες τις σελίδες που υλοποιούν την κλάση Page.

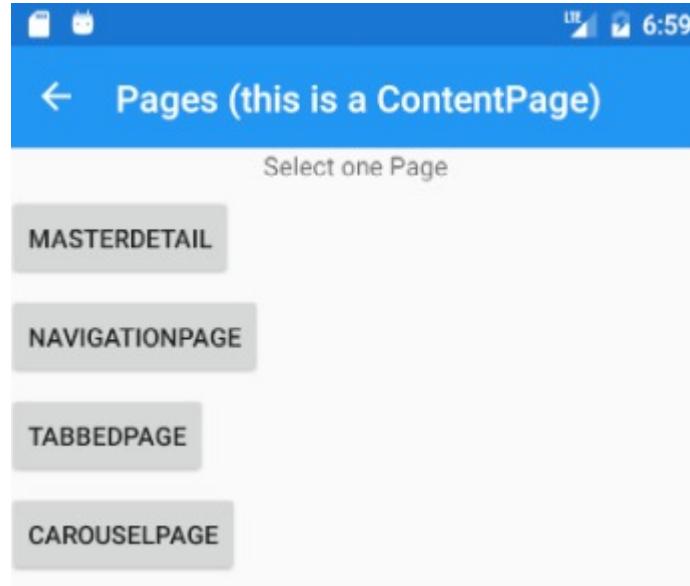
- Κάντε δεξί κλικ στο φάκελο Pages και επιλέξτε “Add new item”. Στο παράθυρο που θα προβληθεί επιλέξτε “Forms Blank Content Page Xaml” και δώστε το όνομα “AContentPage”. Εισάγετε στη xaml τον κώδικα:

```
<StackLayout>
<Label Text="Select one Page"
      VerticalOptions="Center"
      HorizontalOptions="Center" />
<Button Text="MasterDetail" Clicked="Button_Clicked"
      VerticalOptions="Start"
      HorizontalOptions="StartAndExpand" />
<Button Text="NavigationPage" Clicked="Button_Clicked_1"
      VerticalOptions="Center"
      HorizontalOptions="StartAndExpand" />
<Button Text="TabbedPage" Clicked="Button_Clicked_2"
      VerticalOptions="Center"
      HorizontalOptions="StartAndExpand" />
<Button Text="CarouselPage" Clicked="Button_Clicked_3"
      VerticalOptions="Center"
      HorizontalOptions="StartAndExpand" />
</StackLayout>
```

To StackLayout είναι μια δομή οργάνωσης στοιχείων (layout) που στοιβάζει τα αντικείμενα που περιέχει. Είναι κοινό να τοποθετούμε μέσα σε ένα ContentPage ένα μόνο στοιχείο StackLayout.

Πρώτα θέτουμε μία ετικέτα κειμένου και τη στοιχίζουμε στο κέντρο με την ιδιότητα VerticalOptions. Ακολούθως τοποθετούμε τέσσερα κουμπιά Button κάθε ένα από τα οποία οδηγεί σε διαφορετικό τύπο σελίδας.

Για την κάθετη τοποθέτηση χρησιμοποιούμε το VerticalOptions, το οποίο μπορεί να πάρει τιμές όπως Start, Center, End και Fill.



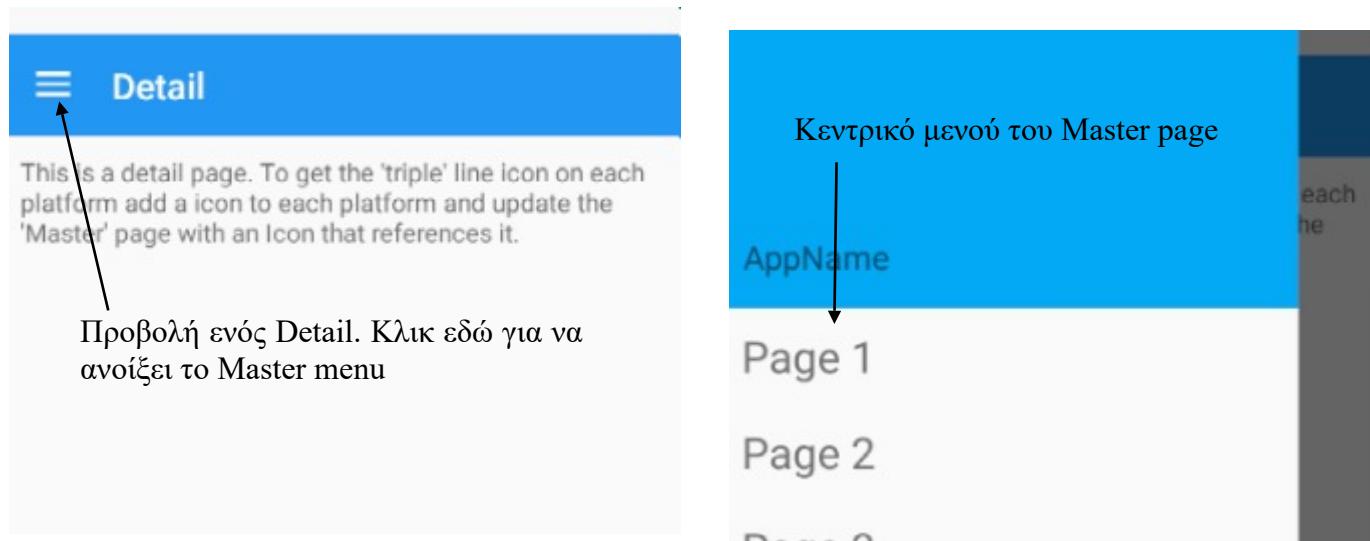
Tags: #ContentPage

1. Παρουσίαση του Xamarin.Forms: Εγκατάσταση, XAML, Pages, Layouts

1.10.2. MasterDetailPage

Το MasterDetailPage είναι μια κλάση που χρησιμοποιείται για τη διαχείριση δύο ειδών πληροφοριών: μια σελίδα Master που ενσωματώνει δεδομένα σε υψηλό επίπεδο (όπως μενού) και μια σελίδα Detail που προβάλει χαμηλού επιπέδου πληροφορίες σχετικά με το επιλεγμένο στοιχείο του Master.

- Για να δημιουργήσουμε ένα MasterDetailPage κάνουμε δεξί κλικ μέσα στο φάκελο Pages και επιλέγουμε Add new item.
- Έπειτα επιλέγουμε “Forms MasterDetail Page Xaml” και το Visual Studio δημιουργεί για εμάς όλα τα απαραίτητα αρχεία ενός MasterDetailPage.



1.10.3. TabbedPage

Το TabbedPage επιτρέπει την πλοήγηση σε διαφορετικά πάνελ πληροφοριών χρησιμοποιώντας tabs. Η πλοήγηση μπορεί να γίνει και σέρνοντας το δάχτυλό μας αριστερά ή δεξιά.

Για να δημιουργήσουμε ένα νέο TabbedPage ακολουθούμε τη γνωστή διαδικασία προσθήκης νέου αντικειμένου και έπειτα επιλέγουμε από τη λίστα προτύπων το “Forms Tabbed Page Xaml”.

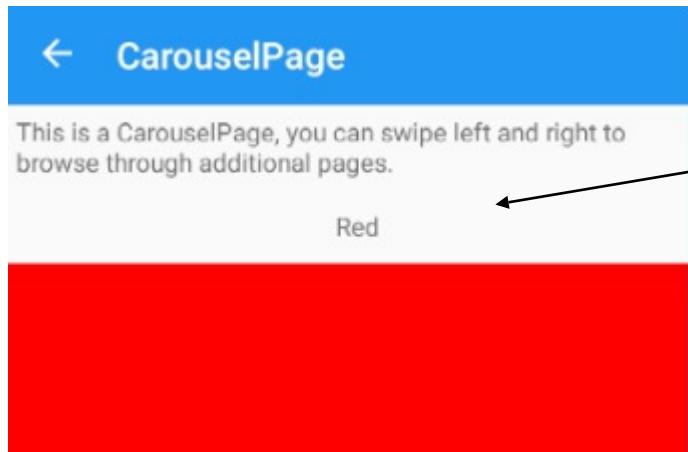


Tags: #MasterDetailPage, #TabbedPage

1.10.4. CarouselPage

Το CarouselPage είναι μία σελίδα που επιτρέπει την εναλλαγή υποσελίδων με ένα απλό σύρσιμο του δαχτύλου μας. Μοιάζει με μια συλλογή φωτογραφιών που προβάλλουμε στο κινητό μας.

Για να προσθέσουμε ένα CarouselPage χρησιμοποιούμε τη γνωστή διαδικασία προσθήκης, επιλέγοντας “Forms Carousel Page Xaml”.



Απλά σέρνουμε το δάχτυλο αριστερά ή δεξιά για να αλλάξουμε υποσελίδα

1.11. Layouts

Τα Layouts χρησιμοποιούνται για να οργανώσουν τα Controls σε λογικές δομές που βοηθούν το χρήστη να τα αξιοποιήσει παραγωγικά. Τα πιο γνωστά Layouts είναι τα εξής:

- StackLayout : Τοποθετεί τα αντικείμενα που περιέχει σε μια γραμμή, οριζόντια ή κατακόρυφα
- ContentView : Περιέχει ένα μόνο στοιχείο και χρησιμοποιείται κυρίως για ορισμένα από το χρήστη Controls
- Frame : Προβάλει ένα μόνο αντικείμενο με κάποια στοιχεία πλαισίωσης (πχ σκιά)
- ScrollView : Επιτρέπει τις μπάρες ολίσθησης αν το περιεχόμενο είναι πολύ μεγάλο
- TemplatedView : Προβάλει περιεχόμενα ενός πρότυπου Control
- AbsoluteLayout : Τοποθετεί τα περιεχόμενα σε ακριβείς θέσεις. Η θέση και το μέγεθος των Controls καθορίζεται από το χρήστη
- Grid : Τοποθετεί τα περιεχόμενα σε γραμμές και στήλες
- RelativeLayout : Χρησιμοποιεί περιορισμούς για να τοποθετήσει τα περιεχόμενα
- ContentPresenter : Είναι διαχειριστής περιεχομένου που χρησιμοποιείται για προτυποποιημένες όψεις.

Στο project μας δημιουργήστε ένα φάκελο με το όνομα Layouts. Για κάθε ένα από τα παραδείγματα που θα ακολουθήσουν δημιουργήστε μέσα στο φάκελο ένα αρχείο xaml τύπου ContentPage.

1.11.1. StackLayout

Το StackLayout τοποθετεί τα στοιχεία σε μια γραμμή, οριζόντια ή κατακόρυφα. Ο κώδικας που θα εισάγουμε στο νέο ContentPage που θα δημιουργήσουμε είναι ο εξής:

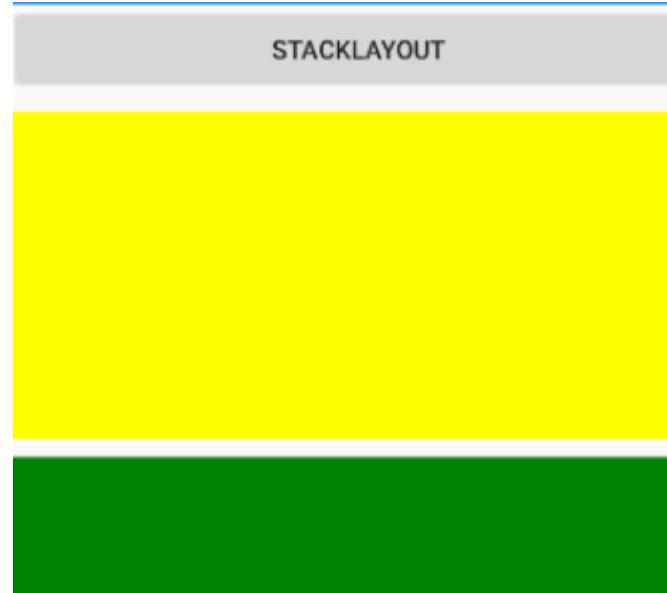
```
<StackLayout Spacing="10" x:Name="layout">
    <Button Text="StackLayout" VerticalOptions="Start"
    HorizontalOptions="FillAndExpand" />
    <BoxView Color="Yellow" VerticalOptions="FillAndExpand"
    HorizontalOptions="FillAndExpand" />
    <BoxView Color="Green" VerticalOptions="FillAndExpand"
    HorizontalOptions="FillAndExpand" />
    <BoxView HeightRequest="75" Color="Blue" VerticalOptions="End"
    HorizontalOptions="FillAndExpand" />
</StackLayout>
```

Tags: #CarouselPage, #Layouts, #StackLayout

Το StackLayout δηλώνει τη στοίχιση των περιεχομένων σε μια γραμμή. Τα αντικείμενα θα έχουν απόσταση 10 (Spacing="10"). Με την ιδιότητα x:Name="layout" δηλώνουμε το όνομα του στιγμιότυπου, το οποίο μπορούμε να χρησιμοποιήσουμε στη C# για να καλέσουμε τις ιδιότητές του.

Έπειτα τοποθετούμε ένα κουμπί στην έναρξη κάτι που δηλώνεται με την ιδιότητα VerticalOptions="Start". Το HorizontalOptions="FillAndExpand" σημαίνει ότι θέλουμε να γεμίσει σε πλάτος και να καλύψει τα όρια του StackLayout.

Τα υπόλοιπα αντικείμενα του StackLayout είναι τύπου BoxView τα οποία είναι απλά κουτιά που περιέχουν ένα χρώμα (δηλώνεται με το Color="Yellow" κτλ). Βάζοντας σε κάθε ένα το VerticalOptions="FillAndExpand" δηλώνουμε ότι επιθυμούμε ισόποσα να γεμίσουν οριζόντια τον υπόλοιπο χώρο του StackLayout.



1.11.2. AbsoluteLayout

Το AbsoluteLayout τοποθετεί τα αντικείμενα και ορίζει το μέγεθός τους δίνοντας αναλογικές ή απόλυτες τιμές. Ας δούμε το παρακάτω παράδειγμα του Aabsolutelayout που βρίσκεται στο φάκελο Layouts:

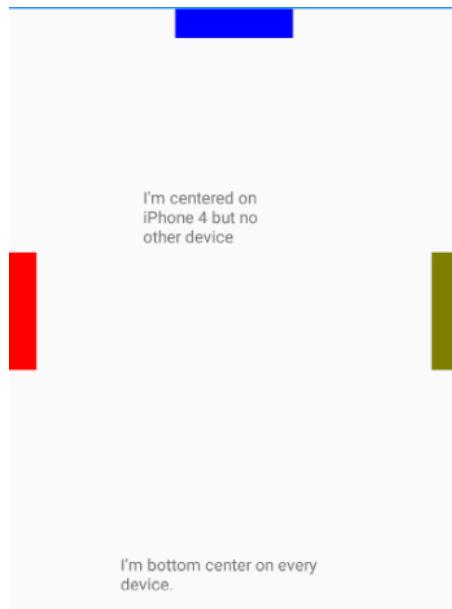
```
<AbsoluteLayout>
    <Label Text="I'm centered on iPhone 4 but no other device"
          AbsoluteLayout.LayoutBounds="115,150,100,100" LineBreakMode="WordWrap" />
    <Label Text="I'm bottom center on every device."
          AbsoluteLayout.LayoutBounds=".5,.1,.5,.1" AbsoluteLayout.LayoutFlags="All"
          LineBreakMode="WordWrap" />
    <BoxView Color="Olive" AbsoluteLayout.LayoutBounds="1,.5, 25, 100"
             AbsoluteLayout.LayoutFlags="PositionProportional" />
    <BoxView Color="Red" AbsoluteLayout.LayoutBounds="0,.5,25,100"
             AbsoluteLayout.LayoutFlags="PositionProportional" />
    <BoxView Color="Blue" AbsoluteLayout.LayoutBounds=".5,0,100,25"
             AbsoluteLayout.LayoutFlags="PositionProportional" />
</AbsoluteLayout>
```

Η ιδιότητα `AbsoluteLayout.LayoutBounds` ορίζει τα όρια και το μέγεθος του αντικειμένου με την εξής σειρά: X, Y, πλάτος, ύψος. Το `LineBreakMode="WordWrap"` σημαίνει ότι όταν το κείμενο του label ξεπεράσει τα όρια άλλαξε γραμμή.

Η ιδιότητα `AbsoluteLayout.LayoutFlags` ορίζει τον τρόπο ερμηνείας των τιμών. Η τιμή All σημαίνει ότι δεχόμαστε τα X,Y,πλάτος και ύψος ως αναλογικά και όχι ως απόλυτες τιμές ενώ η `PositionProportional` σημαίνει ότι το X,Y ερμηνεύονται αναλογικά ενώ το πλάτος/ύψος με απόλυτες τιμές. Πχ το .5,1 σημαίνει 50% πλάτος/100% ύψος.

Tags: #AbsoluteLayout

Ας δούμε το αποτέλεσμα του παραδείγματος:



1.11.3. RelativeLayout

Το RelativeLayout χρησιμοποιείται για να ορίσει τη θέση και το μέγεθος των αντικειμένων που περιέχει αναλογικά με τις ιδιότητες του ή τα κοντινά τους στοιχεία. Χρησιμοποιείται για να δημιουργήσουμε προβολές που προσαρμόζονται σε κάθε μέγεθος οθόνης.

Ας ανοίξουμε το αρχείο ARelativeLayout.xaml για να δούμε ένα παράδειγμα:

```
<RelativeLayout>
    <!-- requires x:Name so we can reference in RelativeLayout on Green box -->
    <BoxView Color="Red" x:Name="Red"
        WidthRequest="200"
        HeightRequest="200"
        RelativeLayout.XConstraint=
            "{ConstraintExpression Type=Constant,
                Constant=10}"
        RelativeLayout.YConstraint=
            "{ConstraintExpression Type=Constant,
                Constant=20}"/>
    <BoxView Color="Green"
        RelativeLayout.XConstraint=
            "{ConstraintExpression Type=RelativeToParent,
                Property=Width,
                Factor=0.5}"
        RelativeLayout.YConstraint=
            "{ConstraintExpression Type=RelativeToView,
                Property=Y,
                ElementName=Red,
                Constant=-5}"/>
    <BoxView Color="Yellow"
        WidthRequest="100"
        HeightRequest="100"
        RelativeLayout.XConstraint=
            "{ConstraintExpression Type=RelativeToParent,
                Property=Width,
                Factor=0.4}"
        RelativeLayout.YConstraint=
            "{ConstraintExpression Type=RelativeToParent,
                Property=Height,
                Factor=0.3}"/>
</RelativeLayout>
```

Tags: #RelativeLayout

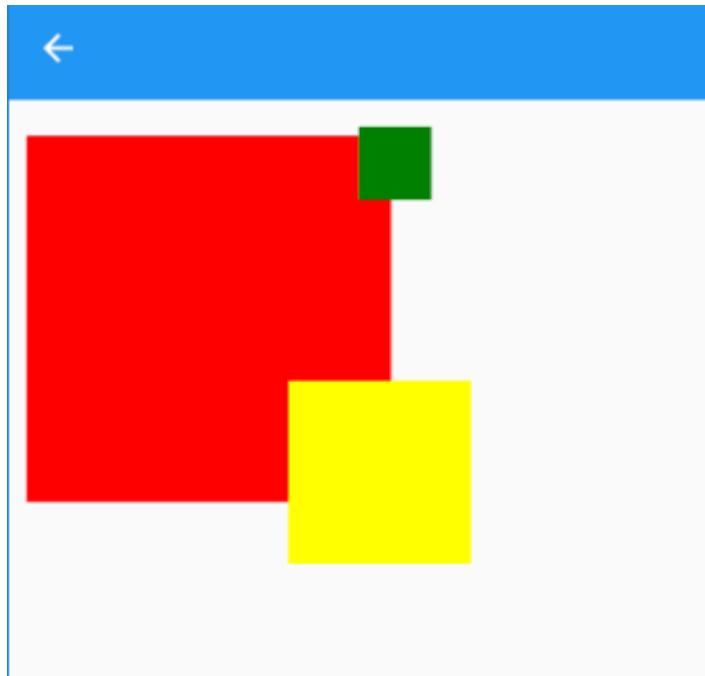
1. Παρουσίαση του Xamarin.Forms: Εγκατάσταση, XAML, Pages, Layouts

Στο πρώτο BoxView θέτουμε WidthRequest και HeightRequest="200", δηλαδή δημιουργούμε ένα τετράγωνο 200X200.

Στο κόκκινο κουτί "Red" το RelativeLayout.Xconstraint θέτει τους κανόνες που πρέπει να ακολουθούνται στον άξονα X με μια έκφραση που δηλώνεται με το ConstraintExpression. Ο τύπος είναι μια σταθερά Type=Constant με τιμή Constant=10, δηλαδή μετακίνησέ το 10 μονάδες δεξιά και 20 μονάδες κάτω (δηλώνεται με το Yconstraint). Στο Πράσινο κουτί δηλώνουμε ότι ο περιορισμός στον άξονα X είναι τύπου RelativeToParent δηλαδή θα βρίσκεται στον άξονα X στο 50% του μήκους του RelativeLayout (ακριβώς στη μέση). Επίσης στον άξονα Y, ο περιορισμός είναι τύπου RelativeToView έχοντας απόσταση από την κορυφή του άξονα Y -5 μονάδες από την απόσταση που έχει το στοιχείο Red. Άρα είναι -5 μονάδες πιο ψηλά από το Red.

Στο κίτρινο κουτί δηλώνουμε διαστάσεις 100X100. Στον άξονα X θέτουμε τον περιορισμό να είναι στο 40% του RelativeLayout, ενώ στον άξονα Y να είναι στο 30%.

Ας δούμε στην παρακάτω εικόνα το τελικό αποτέλεσμα.



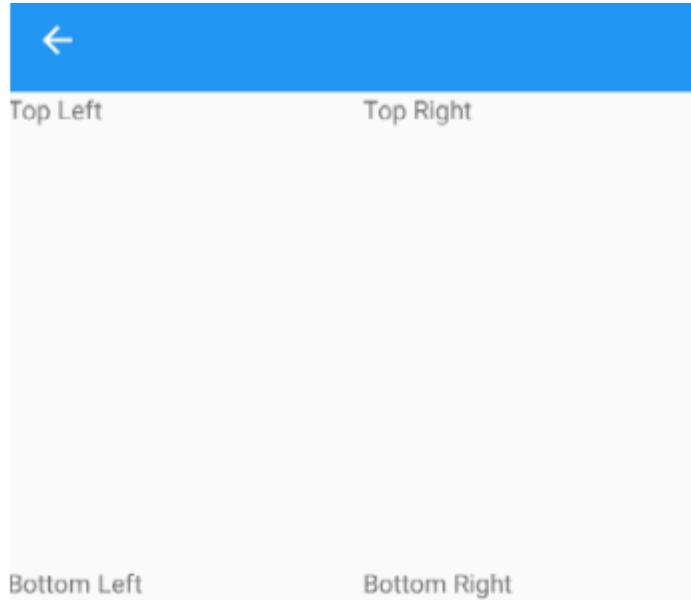
1.11.4. Grid Layout

Με το Grid layout προβάλλουμε τα στοιχεία σε πλέγμα, δηλαδή σε στήλες και γραμμές, όπως ένας πίνακας. Οι διαστάσεις των γραμμών και των στηλών μπορούν να έχουν αναλογικές ή απόλυτες τιμές. Ας μελετήσουμε τον κώδικα του αρχείου AGridLayout.xaml

```
<Grid VerticalOptions="FillAndExpand" HorizontalOptions="FillAndExpand">
    <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <Label Text="Top Left" Grid.Row="0" Grid.Column="0"/>
    <Label Text="Top Right" Grid.Row="0" Grid.Column="1"/>
    <Label Text="Bottom Left" Grid.Row="1" Grid.Column="0"/>
    <Label Text="Bottom Right" Grid.Row="1" Grid.Column="1"/>
</Grid>
```

Tags: #GridLayout

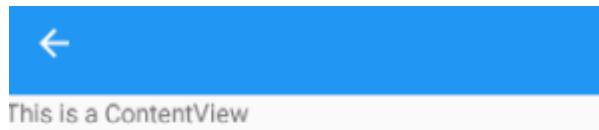
Μέσα στο Grid περιέχονται οι ορισμοί των γραμμών RowDefinitions και των στηλών ColumnDefinition. Παρατηρούμε ότι έχουμε βάλει δύο γραμμές (με το RowDefinition) και δύο στήλες (με το ColumDefinition). Το Height="*" δηλώνει “πάρε ισάξια με τις άλλες γραμμές που έχουν * τον ίδιο χώρο μέχρι να γεμίσει το layout”. Έπειτα, εισάγουμε τα Label και ορίζουμε τα κελιά στα οποία θα μπουν γράφοντας σε πια γραμμή Grid.Row και σε πια στήλη Grid.Column θα μπει το κάθε ένα. Στην παρακάτω εικόνα φαίνεται το τελικό αποτέλεσμα:



1.11.5. ContentView

Το ContentView είναι το πιο απλό layout καθώς περιλαμβάνει μόνο ένα στοιχείο (πχ μια ετικέτα). Είναι όμως πολύ χρήσιμο καθώς χρησιμοποιείται για τη δημιουργία custom controls του χρήστη. Ένα απλό παράδειγμα υλοποίησης φαίνεται παρακάτω (αρχείο AContentView.xaml) :

```
<ContentView>
    <Label Text="This is a ContentView"></Label>
</ContentView>
```



1.11.6. ScrollView

Το ScrollView χρησιμοποιείται για να προβάλλουμε περιεχόμενο που δεν χωράει στην οθόνη καθώς και να εξασφαλίσουμε χώρο για το πληκτρολόγιο. Πολύ απλά εμφανίζει μπάρες ολίσθησης οριζόντια/κάθετα και επιτρέπει με απλό σύρσιμο να προβάλλουμε το λοιπό περιεχόμενο. Ας δούμε τον κώδικα του αρχείου AScrollView.xaml.

```
<ScrollView>
    <StackLayout>
        <BoxView BackgroundColor="Red" HeightRequest="800" WidthRequest="350" />
    </StackLayout>
</ScrollView>
```

Μέσα στο ScrollView έχουμε βάλει ένα άλλο layout τύπου StackLayout, το οποίο περιέχει ένα κόκκινο κουτί μεγέθους 800X350, πολύ μεγαλύτερο από την οθόνη του κινητού. Παρατηρούμε ότι στις άκρες προβάλλονται μπάρες ολίσθησης και με ένα απλό σύρσιμο μπορούμε να προβάλλουμε το κρυφό περιεχόμενο.

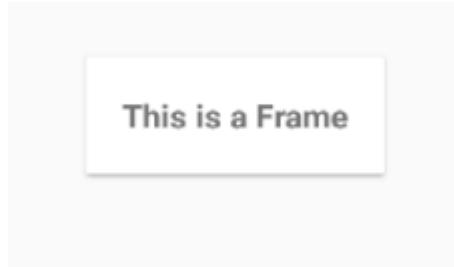
Tags: #ContentView, #ScrollView

1.11.7. Frame

To Frame είναι ένα πολύ απλό layout που μπορεί να προβάλει ένα αντικείμενο με δυνατότητες πλαισίωσης.

```
<Frame HasShadow="True" OutlineColor="Silver" VerticalOptions="CenterAndExpand"  
HorizontalOptions="Center">  
    <Label Text="This is a Frame" Font="Bold, Large" HorizontalOptions="Center" />  
</Frame>
```

Όπως φαίνεται στον κώδικα του αρχείου AFrame.xaml μπορούμε να εισάγουμε σκιά με το HasShadow καθώς και χρώμα περιγράμματος με το OutlineColor.



Tags: #Frame

2. Κontrols και χειρισμός συμβάντων

2.1. To project App1

Όλα τα παραδείγματα του παρόντος εγχειριδίου περιλαμβάνονται στο project με ονομασία App1. Πρόκειται για μια cross platform εφαρμογή που εκτελείται σε Windows 10, Android και iOS λειτουργικά συστήματα. Αφού κατεβάσετε το project από το github μεταβείτε στο φάκελο App1 και ανοίξτε το project στο Visual Studio 2017, κάνοντας διπλό κλικ στο αρχείο App1.sln.

Όνομα	Ημερομηνία τροποποίησης	Τύπος	Μέγεθος
.vs	4/4/2017 1:48 μμ	Φάκελος αρχείων	
App1	4/4/2017 1:49 μμ	Φάκελος αρχείων	
packages	4/4/2017 1:49 μμ	Φάκελος αρχείων	
UniBooks	21/7/2017 7:05 πμ	Φάκελος αρχείων	
App1.sln	1/4/2017 8:27 μμ	Microsoft Visual S...	19 KI

Αφού ανοίξετε το αρχείο solution θα δείτε τέσσερις φακέλους (projects):

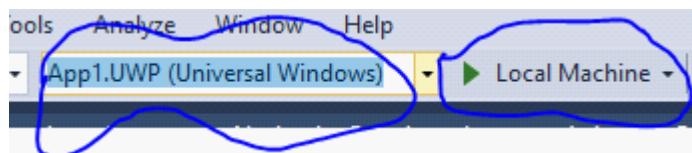
- App1 (Portable) : Περιέχει τον κοινό κώδικα για όλα τα λειτουργικά συστήματα.
- App1.Android : Περιέχει τον κώδικα που θα εκτελεστεί αποκλειστικά στο Android.
- App1.iOS : Περιέχει τον κώδικα που θα εκτελεστεί στο iOS
- App1.UWP: Περιέχει τον κώδικα που θα εκτελεστεί στα Windows 10.

Στο παρόν εγχειρίδιο θα εστιάσουμε πλήρως στο project App1 (Portable) χωρίς να χρειαστεί να ασχοληθούμε καθόλου με τον κώδικα που είναι προσαρμοσμένος σε κάθε λειτουργικό σύστημα. Σε περίπτωση που είχαμε διαφορετικές υλοποιήσεις για το χειρισμό ενός στοιχείου (πχ GPS) σε κάθε λειτουργικό σύστημα, τότε θα κινούμασταν ως εξής:

- Θα δημιουργούσαμε ένα κοινό interface στο project App1 (Portable)
- Και θα υλοποιούσαμε αυτό το interface στο project του κάθε λειτουργικού συστήματος.

Εκτέλεση του App1 στα Windows 10

Για να εκτελέσουμε το project στα Windows 10, μεταβαίνουμε στο project App1.UWP και με δεξί κλικ πάνω στο App1.UWP επιλέγουμε “Deploy”. Επειτα, εκτελούμε το πρόγραμμα επιλέγοντας στην πάνω μπάρα του Visual Studio το “App1.UWP (Universal Windows)” και πατώντας το πράσινο βελάκι (Local Machine).



2. Controls και χειρισμός συμβάντων

Μετά την εκκίνηση του App1 θα μεταφερθούμε στην κύρια οθόνη (MainPage), όπου και θα πρέπει να επιλέξουμε το παράδειγμα που θέλουμε να προβάλλουμε. Για να δούμε την υλοποίηση των Controls επιλέγουμε το κουμπί Controls και μεταφερόμαστε στο παράθυρο των Controls που υλοποιείται από το αρχείο AControlsPage.xaml.

- ▷ Bindings
- ◀ Controls
- ▷ AControlsPage.xaml
- ▷ Layouts

Label

To Label χρησιμοποιείται για να προβάλλουμε κείμενο για ανάγνωση από το χρήστη και είναι ίσως το control που θα χρησιμοποιούμε πιο συχνά.

test value

Σε ένα label μπορούμε να αλλάξουμε ένα πλήθος από ιδιότητες, όπως το χρώμα, το μέγεθος και το στυλ του κειμένου. Για παράδειγμα αν θέλουμε πλάγιο κείμενο απλά προσθέτουμε τον κώδικα *FontAttributes="Italic"*.

Για να εισάγουμε γενικά μια ετικέτα κειμένου Label χρησιμοποιούμε τον παρακάτω κώδικα στη xaml:

```
<Label x:Name="LabelText" FontAttributes="Italic" />
```

Για να θέσουμε το κείμενο της ετικέτας είτε προσθέτουμε τον απλό κώδικα xaml *Text="test value"* είτε προσθέτουμε τον κώδικα c# *LabelText.Text="test value"*; στον κατασκευαστή της σελίδας AControlsPage.xaml.cs.

ActivityIndicator

To ActivityIndicator χρησιμοποιείται όταν θέλουμε να ενημερώσουμε το χρήστη ότι πρέπει να περιμένει μέχρι να ολοκληρωθεί μια χρονοβόρα διαδικασία, όπως το άνοιγμα ενός αρχείου κτλ.

• • • ..

Η πιο συχνή ιδιότητα που αλλάζουμε είναι το χρώμα (Color) και το IsRunning (ενεργοποίηση με true). Για να εισάγουμε ένα ActivityIndicator χρησιμοποιούμε τον παρακάτω κώδικα xaml:

```
<ActivityIndicator Color="Red" IsRunning="true" ></ActivityIndicator>
```

BoxView

To BoxView είναι ένα απλό ορθογώνιο σχήμα που χρησιμοποιείται πολύ συχνά για τη βελτίωση του γραφικού περιβάλλοντος της εφαρμογής.



Η πιο συχνές ιδιότητες είναι το χρώμα (Color) και οι διαστάσεις (WidthRequest/ HeightRequest).

Για να εισάγουμε ένα BoxView χρησιμοποιούμε τον κάτωθι κώδικα xaml:

```
<BoxView Color="Red" WidthRequest="50" HeightRequest="50"></BoxView>
```

2. Controls και χειρισμός συμβάντων

Button

Όπως η ετικέτα Label έτσι και το κουμπί είναι από τα πιο δημοφιλή στοιχεία διεπαφής με το χρήστη. Πατώντας το εκτελείται ένα συμβάν με όνομα Clicked, το οποίο καλεί μια συνάρτηση χειρισμού του συμβάντος.



Η βασική και πιο απλή υλοποίηση ενός κουμπιού περιλαμβάνει τις ιδιότητες του κειμένου Text και του συμβάντος Clicked.

Για να εισάγουμε ένα Button χρησιμοποιούμε τον κάτωθι κώδικα xaml:

```
<Button Clicked="Button_Clicked" Text="Click me!"></Button>
```

Στον κώδικα διακρίνουμε το Clicked="Button_Clicked". Αυτό μεταφράζεται ως εξής: αν γίνει κλικ στο κουμπί βρες τη συνάρτηση Button_Clicked και εκτέλεσέ την. Αν ανοίξουμε τον κώδικα c# του αρχείου AControlsPage.xaml.cs θα δούμε την υλοποίηση της συνάρτησης ως εξής:

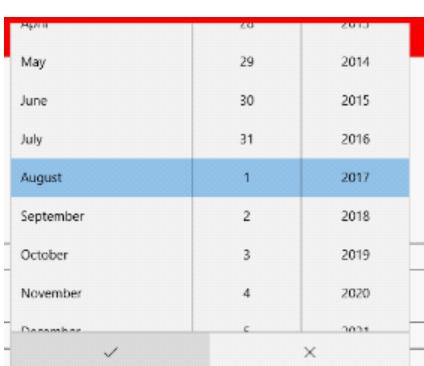
```
private void Button_Clicked(object sender, EventArgs e)
{
    ButtonLabel.Text = "Thank you";
}
```

Η συνάρτηση μας λέει ότι στην ιδιότητα Text (κείμενο που προβάλλεται) του Label ButtonLabel βάλε την τιμή “Thank you”.

Έτσι αν κάνουμε κλικ στο κουμπί θα δούμε από κάτω να αλλάξει η τιμή της ετικέτας σε “Thank you”.

DatePicker

To DatePicker είναι ένα control που διευκολύνει την προβολή αλλά και την επιλογή ημερομηνίας από το χρήστη.



Οι πιο σημαντικές ιδιότητες του DatePicker είναι το Date και το συμβάν DateSelected.

Για να εισάγουμε ένα DatePicker χρησιμοποιούμε τον κάτωθι κώδικα xaml:

```
<DatePicker x:Name="DateCtrl" DateSelected="DateCtrl_DateSelected"></DatePicker>
```

Αφού προσδιορίσουμε το όνομα του DatePicker με την ιδιότητα x:Name="DateCtrl" (άρα θα το καλούμε με το όνομα DateCtrl), για να θέσουμε την προεπιλεγμένη ημερομηνία, μπορούμε να εισάγουμε τον παρακάτω κώδικα c# στη συνάρτηση κατασκευαστή του αρχείου AControlsPage.xaml.cs:

```
DateCtrl.Date = DateTime.Now;
```

Αυτό σημαίνει ότι όταν φορτώνεις τη σελίδα βάλε στο DatePicker με όνομα DateCtrl ως προεπιλεγμένη ημερομηνία την ημερομηνία που έχουμε Τώρα (DateTime.Now).

Επίσης, για να χειριστούμε τα συμβάντα επιλογής ημερομηνίας από το χρήστη τροποποιούμε την ιδιότητα DateSelected θέτοντας ως συνάρτηση χειρισμού τη συνάρτηση με όνομα “DateCtrl_DateSelected”. Η υλοποίηση της συνάρτησης σε c# φαίνεται παρακάτω:

2. Controls και χειρισμός συμβάντων

```
private void DateCtrl_DateSelected(object sender, DateChangedEventArgs e)
{
    DateSelected.Text = e.NewDate.ToString();
}
```

Η συνάρτηση μας λέει ότι όταν αλλάξει η ημερομηνία από το χρήστη ενημέρωσε το κείμενο της ετικέτας με όνομα *DateSelected* με την τιμή της επιλεγμένης ημερομηνίας (*e.NewDate.ToString()*).

Editor

Ο Editor χρησιμοποιείται για να εισάγει ο χρήστης τουλάχιστον μία γραμμή κειμένου.

Editor

```
sdjkdhdssd\
fsdfksjksdlj
\fsdfsdf
```

Οι πιο συχνά χρησιμοποιούμενες ιδιότητες είναι το ύψος του Editor (HeightRequest) καθώς και το συμβάν αλλαγής του κειμένου από το χρήστη (TextChanged).

Για να εισάγουμε έναν Editor γράφουμε τον παρακάτω κώδικα xaml:

```
<Editor x:Name="EditorCtr" TextChanged="EditorCtr_TextChanged" HeightRequest="100" ></Editor>
```

Ο κώδικας θέτει το όνομα *EditorCtr*, ύψος 100 μονάδες και σε περίπτωση αλλαγής του κειμένου από το χρήστη (έστω και σε ένα γράμμα) ενεργοποιεί το συμβάν *TextChanged* καλώντας τη συνάρτηση c# με όνομα *EditorCtr_TextChanged*. Η υλοποίησή της στον κώδικα c# φαίνεται παρακάτω:

```
private void EditorCtr_TextChanged(object sender, TextChangedEventArgs e)
{
    EditorText.Text = e.NewTextValue;
}
```

Ετσι, όταν αλλάξει το κείμενο του Editor, το κείμενο του Label με όνομα *EditorText* θα πάρει την νέα τιμή κειμένου του Editor (*e.NewTextValue*).

Entry

To Entry είναι όμοιο με τον Editor, με τη διαφορά ότι υποστηρίζει σύντομο κείμενο μίας γραμμής.

```
σδξκλησδξκσ
```

Για να εισάγουμε ένα Entry χρησιμοποιούμε τον παρακάτω κώδικα xaml:

```
<Entry x:Name="EntryCtr" TextChanged="EntryCtr_TextChanged"></Entry>
```

Σε περίπτωση αλλαγής του κειμένου από το χρήστη ενεργοποιείται (όπως και στον Editor) το συμβάν *TextChanged* με την κλήση της συνάρτησης c# *EntryCtr_TextChanged*:

```
private void EntryCtr_TextChanged(object sender, TextChangedEventArgs e)
{
    EntryText.Text = e.NewTextValue;
}
```

Ετσι, όταν αλλάξει το κείμενο του Entry, το κείμενο του Label με όνομα *EntryText* θα πάρει την νέα τιμή κειμένου του Editor (*e.NewTextValue*).

2. Κontrols και χειρισμός συμβάντων

Image

To Image χρησιμοποιείται για την προβολή εικόνων.



Για να εισάγουμε μια εικόνα Image χρησιμοποιούμε τον παρακάτω κώδικα xaml:

```
<Image x:Name="ImageCtr" ></Image>
```

Για να φορτώσουμε μια εικόνα χρησιμοποιούμε τον παρακάτω κώδικα c#:

```
ImageCtr.Source = ImageSource.FromUri(new Uri("http://www.uom.gr/themes/UOM3/images/pamak-front-ell-header.jpg"));
```

Ο παραπάνω κώδικας λαμβάνει την εικόνα με το σήμα του πανεπιστημίου Μακεδονίας από το διαδίκτυο.
Για να θέσουμε μια εικόνα που είναι τοπικά αποθηκευμένη σε κάθε project λειτουργικού συστήματος (Android/iOS/Win10) απλά γράφουμε τον παρακάτω κώδικα xaml που περιέχει το όνομα του αρχείου εικόνας:

```
<Image Source="pamak.jpg" />
```

ListView

To ListView χρησιμοποιείται για να προβάλλει πλήθος εγγραφών δεδομένων με προσαρμοσμένη μορφοποίηση.

ListView

Text1	Val1
Text2	Val2
Text3	Val3

Selected item=Text2 Val2

2. Controls και χειρισμός συμβάντων

Για να εισάγουμε ένα ListView χρησιμοποιούμε μια δομή όπως ο παρακάτω κώδικας xaml:

```
<ListView x:Name="ListViewCtr" ItemSelected="ListViewCtr_ItemSelected" ItemsSource="{Binding ListValues}">
    <ListView.ItemTemplate>
        <DataTemplate>
            <ViewCell>
                <ViewCell.View>
                    <Grid>
                        <Grid.ColumnDefinitions>
                            <ColumnDefinition Width="1*"/>
                            <ColumnDefinition Width="5*"/>
                        </Grid.ColumnDefinitions>
                        <Grid.RowDefinitions>
                            <RowDefinition Height="auto"/></RowDefinition>
                        </Grid.RowDefinitions>
                        <Label Grid.Column="0" Text="{Binding Text}"></Label>
                        <Label Grid.Column="1" Text="{Binding Val}"></Label>
                    </Grid>
                </ViewCell.View>
            </ViewCell>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

Ξεκινώντας από τη δήλωση του ListView, συναντάμε τη δήλωση του ονόματος της λίστας (ListViewCtr), το χειρισμό του συμβάντος επιλογής εγγραφής από το χρήστη (ItemSelected) και τη δήλωση της πηγής από όπου θα αντλήσει τα δεδομένα (ItemsSource).

Η γραμμή ListView.ItemTemplate (και τα στοιχεία <DataTemplate> <ViewCell> <ViewCell.View>) δηλώνουν ότι εκεί ξεκινάει το πρότυπο μορφοποίησης κάθε γραμμής (δηλαδή πως θα φαίνεται στο χρήστη). Βλέπουμε ότι κάθε εγγραφή προβάλλεται μέσα σε ένα Grid (πίνακα) με δύο στήλες (ColumnDefinition) και μια γραμμή (RowDefinition). Στην πρώτη στήλη έχουμε ένα Label που αποτυπώνει την τιμή Text ενώ στη δεύτερη στήλη έχουμε ένα Label που αποτυπώνει την τιμή Val.

Για να φορτώσουμε δεδομένα σε μια λίστα χρησιμοποιούμε κώδικα c#. Αρχικά δημιουργούμε τοπικά μια κλάση που θα φέρει τη δομή της κάθε εγγραφής, όπως η ListValue στο παράδειγμά μας:

```
public class ListValue
```

```
{
    public string Text { get; set; }
    public string Val { get; set; }
}
```

Κάθε μια εγγραφή ListValue πρέπει να μπει σε μια λίστα (συλλογή) που τελικά θα φορτώσουμε και θα εισάγουμε στη ListView:

```
public List<ListValue> ListValues { get; set; }
```

Στον κατασκευαστή της σελίδας μας, φορτώνουμε την λίστα με απλές τιμές, όπως παρακάτω:

```
ListValues = new List<ListValue>
```

```
{
    new ListValue{ Text="Text1",Val="Val1" },
    new ListValue{ Text="Text2",Val="Val2" },
    new ListValue{ Text="Text3",Val="Val3" }
};
```

2. Controls και χειρισμός συμβάντων

Επειτα, για να συνδέσουμε τη λίστα δεδομένων με τη ListView είτε χρησιμοποιούμε τον κώδικα xaml: `ItemsSource="{Binding ListValues}"`, είτε τον κώδικα c# : `ListViewCtr.ItemsSource = ListValues;` .

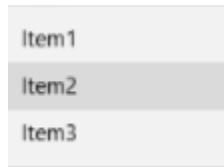
Για να χειριστούμε το συμβάν επιλογής εγγραφής από το χρήστη, χρησιμοποιούμε το συμβάν `ItemSelected="ListViewCtr_ItemSelected"` με το οποίο μετά την επιλογή καλούμε τη συνάρτηση `ListViewCtr_ItemSelected`. Παρακάτω βλέπουμε την υλοποίηση της συνάρτησης σε c# :

```
private void ListViewCtr_ItemSelected(object sender, SelectedItemChangedEventArgs e)
{
    var itm = e.SelectedItem as ListValue;
    ListViewText.Text = "Selected item=" + itm.Text + " " + itm.Val;
}
```

Αρχικά μετατρέπουμε την επιλεγμένη εγγραφή σε τύπο `ListValue` (`e.SelectedItem as ListValue`) και έπειτα θέτουμε το κείμενο του Label `ListViewText` με την τιμή του Text και του Val.

Picker

Το Picker επιτρέπει το χρήστη να επιλέξει μια τιμή μέσα από μια λίστα πολλαπλών τιμών.



Μπορούμε να φορτώσουμε τις τιμές της λίστας του Picker είτε μέσω κώδικα c# είτε με κώδικα xaml, όπως παρακάτω:

```
<Picker x:Name="PickerCtr" Title="Select an Item"
SelectedIndexChanged="PickerCtr_SelectedIndexChanged">
<Picker.Items>
    <x:String>Item1</x:String>
    <x:String>Item2</x:String>
    <x:String>Item3</x:String>
</Picker.Items>
</Picker>
```

Αρχικά, θέτουμε το όνομα του Picker, ένα τίτλο που θα προβάλλεται μετά τη φόρτωση του Picker και το συμβάν επιλογής τιμής από το χρήστη `SelectedIndexChanged`.

Εντός του Picker, εντός της ετικέτας `<Picker.Items>` θέτουμε τη λίστα τιμών με τύπο String (κειμένου):

```
<x:String>Item1</x:String>
```

Για να χειριστούμε ένα συμβάν επιλογής εγγραφής από το χρήστη υλοποιούμε τη συνάρτηση c# `PickerCtr_SelectedIndexChanged` ως εξής:

```
private void PickerCtr_SelectedIndexChanged(object sender, EventArgs e)
{
    PickerText.Text = PickerCtr.Items[PickerCtr.SelectedIndex];
}
```

Ο ανωτέρω κώδικας ενημερώνει το κείμενο του Label `PickerText` με το κείμενο της επιλεγμένης εγγραφής. Ουσιαστικά ο κώδικας `PickerCtr.Items[PickerCtr.SelectedIndex]` σημαίνει: φέρε από τον πίνακα τιμών `Items` του Picker με όνομα `PickerCtr` την εγγραφή που βρίσκεται στη θέση `PickerCtr.SelectedIndex` (θέση επιλεγμένης εγγραφής).

2. Controls και χειρισμός συμβάντων

ProgressBar

Το ProgressBar είναι ένα πολύ απλό control που ενημερώνει το χρήστη για την πρόοδο μιας χρονοβόρας ενέργειας, όπως η ανάγνωση ενός αρχείου.

ProgressBar

Για να υλοποιήσουμε το ProgressBar με κώδικα xaml γράφουμε:

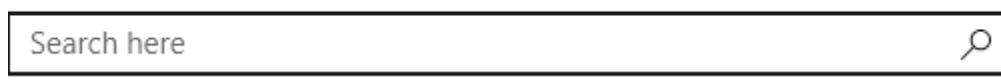
```
<ProgressBar x:Name="ProgressBarCtr" Progress=".2"></ProgressBar>
```

Οπως βλέπουμε στον κώδικα, βασική ιδιότητα είναι η τιμή της προόδου *Progress=".2"*, η οποία ξεκινάει από το 0% (0) μέχρι την τιμή 100% (1).

SearchBar

Το SearchBar υλοποιεί μια απαίτηση που οι προγραμματιστές συναντούν συχνά στις εφαρμογές, το γραφικό περιβάλλον της αναζήτησης πληροφοριών στην εφαρμογή.

SearchBar



Για να εισάγουμε το SearchBar προσθέτουμε τον παρακάτω κώδικα xaml:

```
<SearchBar x:Name="SearchBarCtr" Placeholder="Search here"></SearchBar>
```

Η ιδιότητα *Placeholder="Search here"* θέτει το κείμενο που θα υπάρχει όταν το πεδίο κειμένου είναι κενό.

Slider

Το Slider επιτρέπει το χρήστη να επιλέξει μια αριθμητική τιμή μέσα από ένα εύρος τιμών, σέρνοντας το ποντίκι/δάχτυλο.

Slider



0.6

Για να εισάγουμε το Slider χρησιμοποιούμε τον παρακάτω κώδικα xaml:

```
<Slider x:Name="SliderCtr" Value=".4" ValueChanged="SliderCtr_ValueChanged"></Slider>
```

Το *Value=".4"* θέτει την αρχική τιμή του Slider ενώ για να χειριστούμε το συμβάν της αλλαγής της τιμής από το χρήστη χρησιμοποιούμε το *ValueChanged="SliderCtr_ValueChanged"*. Έτσι, όταν ο χρήστης σύρει το δάχτυλο και αλλάξει την τιμή θα γίνει κλήση της συνάρτησης c# *SliderCtr_ValueChanged*, η οποία υλοποιείται με τον παρακάτω κώδικα:

```
private void SliderCtr_ValueChanged(object sender, ValueChangedEventArgs e)
{
    SliderText.Text = e.NewValue.ToString();
}
```

Ο ανωτέρω κώδικας θέτει το κείμενο του Label *SliderText* με τη νέα αριθμητική τιμή (που μετατρέπεται σε κείμενο με το *e.NewValue.ToString()*).

2. Κontrols και χειρισμός συμβάντων

Stepper

To Stepper χρησιμοποιείται για να εισάγει ο χρήστης αριθμητική τιμή με μοναδιαία αύξηση/μείωση.

Stepper



7

Για να εισάγουμε το Stepper χρησιμοποιούμε τον παρακάτω κώδικα xaml:

<Stepper x:Name="StepperCtr" Value="6" ValueChanged="StepperCtr_ValueChanged"></Stepper>
Η ιδιότητα *Value* = "6" θέτει την αρχική τιμή του Stepper. Έτσι αν πατήσουμε το + η τιμή θα γίνει 7. Για να

χειριστούμε το συμβάν της αυξομείωσης τιμής από το χρήστη αξιοποιούμε το συμβάν *ValueChanged*, το

οποίο ενεργοποιεί τη c# συνάρτηση *StepperCtr_ValueChanged*, η οποία υλοποιείται ως εξής:

```
private void StepperCtr_ValueChanged(object sender, ValueChangedEventArgs e)
{
    StepperText.Text = e.NewValue.ToString();
}
```

Η ανωτέρω συνάρτηση ενημερώνει το κείμενο του Label *StepperText* με τη νέα τιμή *e.NewValue.ToString()*.

Switch

To Switch είναι ένας διακόπτης On/Off και χρησιμοποιείται για είσοδο τύπου Ναι/Όχι από το χρήστη.

Switch



On

True

Για να υλοποιήσουμε το Switch χρησιμοποιούμε τον κώδικα xaml:

<Switch x:Name="SwitchCtr" IsToggled="True" Toggled="SwitchCtr_Toggled"></Switch>

Βασική ιδιότητα είναι το *IsToggled="True"*, το οποίο θέτει το διακόπτη σε *On* (*True*) ή *Off* (*False*). Για να χειριστούμε το συμβάν αλλαγής του διακόπτη χρησιμοποιούμε το συμβάν *Toggled* το οποίο κατευθύνει στη συνάρτηση c# *SwitchCtr_Toggled* η οποία υλοποιείται ως εξής:

```
private void SwitchCtr_Toggled(object sender, ToggledEventArgs e)
{
    SwitchText.Text = e.Value.ToString();
}
```

Η ανωτέρω συνάρτηση θέτει το κείμενο του Label *SwitchText* με την νέα τιμή *True/False* που λαμβάνει από το *e.Value.ToString()*.

TableView

To TableView χρησιμοποιείται για την εισαγωγή επιλογών/ρυθμίσεων του χρήστη σε μια λίστα που φέρει διαφορετική μορφοποίηση σε κάθε γραμμή (πχ φόρμες/ρυθμίσεις εφαρμογής κτλ).

2. Controls και χειρισμός συμβάντων

TableView



Για να εισάγουμε το TableView του παραδείγματος χρησιμοποιούμε τον παρακάτω κώδικα xaml:

```
<TableView x:Name="TableViewCtr">
    <TableRoot>
        <TableSection Title="Title1">
            <TextCell Text="Text1" Detail="Detail1"></TextCell>
            <EntryCell Label="Entry1:" Placeholder="Type here" ></EntryCell>
        </TableSection>
        <TableSection Title="Title2">
            <TextCell Text="Text2" Detail="Detail2"></TextCell>
            <EntryCell Label="Entry2:" Placeholder="Type here" ></EntryCell>
            <SwitchCell Text="SwitchCell1:"></SwitchCell>
        </TableSection>
    </TableRoot>
</TableView>
```

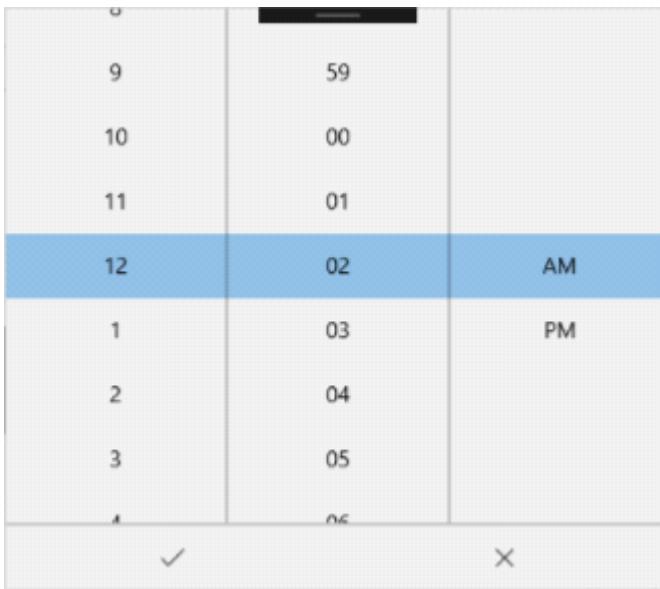
To TableView περιέχει ένα TableRoot (ρίζα) στο οποίο ανήκουν πολλά TableSection (ενότητες ρυθμίσεων). Κάθε TableSection έχει έναν τίτλο (πχ title1, title2...). Μέσα στο TableSection βάζουμε τα controls που επιθυμούμε, όπως:

- Το TextCell που είναι ένα κελί κειμένου που φέρει τίτλο (με έντονα γράμματα) και λεπτομέρειες (από κάτω με μικρότερα γράμματα).
- Το EntryCell που είναι ένα κελί τύπου Entry που υποστηρίζει Label (από πάνω, πχ *Label="Entry1:"*) και Placeholder.
- Το SwitchCell που είναι ένα κελί-διακόπτης με την ιδιότητα Text που προβάλει το αντίστοιχο Label από μπροστά.

TimePicker

To TimePicker είναι ένα control που επιτρέπει το χρήστη να επιλέξει ώρα.

2. Controls και χειρισμός συμβάντων



Για να δημιουργήσουμε ένα TimePicker χρησιμοποιούμε τον παρακάτω κώδικα xaml:

```
<TimePicker x:Name="TimePickerCtr" ></TimePicker>
```

WebView

To WebView είναι ένα control που επιτρέπει το χρήστη να προβάλει μια ιστοσελίδα εντός της εφαρμογής.

WebView

Xamarin.Forms Class

A [View](#) that presents HTML content.

See Also: [WebView](#)

Για να εισάγουμε ένα WebView χρησιμοποιούμε τον παρακάτω κώδικα xaml:

```
<WebView x:Name="WebViewCtr" HeightRequest="400" ></WebView>
```

Θέτοντας `HeightRequest="400"` δηλώνουμε το επιθυμητό ύψος του control.

Για να φορτώσουμε την επιθυμητή ιστοσελίδα, χρησιμοποιούμε τον παρακάτω κώδικα c# στη συνάρτηση κατασκευαστή της σελίδας μας:

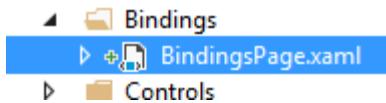
```
UrlWebViewSource u = new UrlWebViewSource();
u.Url = "https://developer.xamarin.com/api/type/Xamarin.Forms.WebView/";
WebViewCtr.Source = u;
```

Δηλαδή, δημιουργούμε ένα αντικείμενο `UrlWebViewSource` στο οποίο θέτουμε ως `Url` τη διεύθυνση της ιστοσελίδας. Έπειτα, το θέτουμε ως `Source` στο `WebView`.

3. Διασύνδεση δεδομένων με γραφικό περιβάλλον

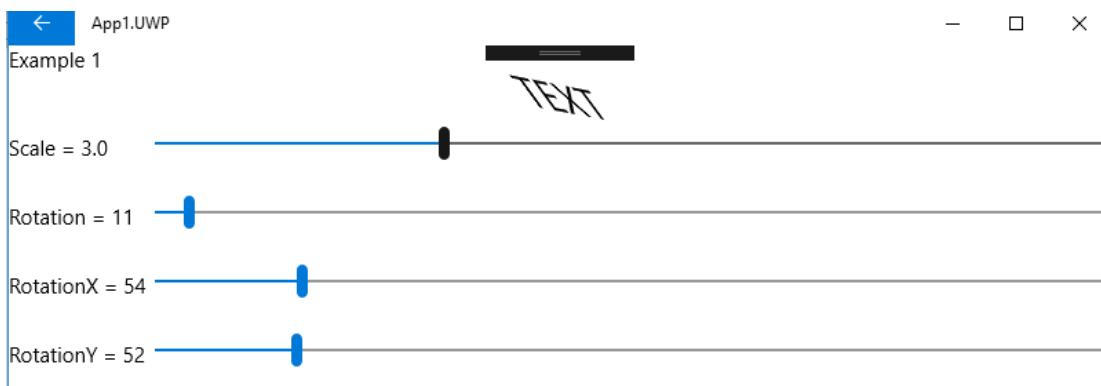
Το xamarin.Forms μας προσφέρει ένα μηχανισμό διασύνδεσης του γραφικού περιβάλλοντος χρήστη με τα δεδομένα. Με αυτό το μηχανισμό, όταν τροποποιούνται τα δεδομένα ενημερώνεται αυτόματα το γραφικό περιβάλλον και αντιστρόφως, όταν τροποποιείται το γραφικό περιβάλλον ενημερώνονται αυτόματα τα δεδομένα.

Για να δούμε στην πράξη τα Bindings θα μελετήσουμε την επιλογή Bindings στο project App1. Το παράδειγμα υλοποιείται σε xaml στο αρχείο BindingsPage.xaml.



Έχοντας μια πρώτη εξοικείωση με τα Controls και τα Events από το προηγούμενο κεφάλαιο, στο πρώτο παράδειγμα (example 1) θα δημιουργήσουμε ένα Label και από κάτω τέσσερα Slider. Όταν μεταβάλλεται η τιμή του κάθε Slider θα τροποποιεί και μια ιδιότητα του Label.

Στην παρακάτω εικόνα θα δείτε το τελικό αποτέλεσμα του παραδείγματος:



Παρακάτω θα βρείτε τον κωδικό xaml του παραδείγματος:

```
<StackLayout Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="2">
    <!-- Scaled and rotated Label -->
    <Label x:Name="label"
        Text="TEXT"
        HorizontalOptions="Center"
        VerticalOptions="CenterAndExpand" />
</StackLayout>

    <!-- Slider and identifying Label for Scale -->
    <Slider x:Name="scaleSlider"
        BindingContext="{x:Reference label}"
        Grid.Row="1" Grid.Column="1"
        Maximum="10"
        Value="{Binding Scale, Mode=TwoWay}" />

    <Label BindingContext="{x:Reference scaleSlider}"
        Text="{Binding Value, StringFormat='Scale = {0:F1}'}"
        Grid.Row="1" Grid.Column="0"
        VerticalTextAlignment="Center" />
```

3. Διασύνδεση δεδομένων με γραφικό περιβάλλον

```
<!-- Slider and identifying Label for Rotation -->
<Slider x:Name="rotationSlider"
BindingContext="{x:Reference label}"
Grid.Row="2" Grid.Column="1"
Maximum="360"
Value="{Binding Rotation, Mode=OneWayToSource}" />

<Label BindingContext="{x:Reference rotationSlider}"
Text="{Binding Value, StringFormat='Rotation = {0:F0}'}"
Grid.Row="2" Grid.Column="0"
VerticalTextAlignment="Center" />

<!-- Slider and identifying Label for RotationX -->
<Slider x:Name="rotationXSlider"
BindingContext="{x:Reference label}"
Grid.Row="3" Grid.Column="1"
Maximum="360"
Value="{Binding RotationX, Mode=OneWayToSource}" />

<Label BindingContext="{x:Reference rotationXSlider}"
Text="{Binding Value, StringFormat='RotationX = {0:F0}'}"
Grid.Row="3" Grid.Column="0"
VerticalTextAlignment="Center" />

<!-- Slider and identifying Label for RotationY -->
<Slider x:Name="rotationYSlider"
BindingContext="{x:Reference label}"
Grid.Row="4" Grid.Column="1"
Maximum="360"
Value="{Binding RotationY, Mode=OneWayToSource}" />

<Label BindingContext="{x:Reference rotationYSlider}"
Text="{Binding Value, StringFormat='RotationY = {0:F0}'}"
Grid.Row="4" Grid.Column="0"
VerticalTextAlignment="Center" />
```

Αρχικά δηλώνεται το Label, οι ιδιότητες του οποίου θα μεταβάλλονται, με όνομα label και κείμενο “TEXT”.

Έπειτα δηλώνεται ένα Slider με όνομα scaleSlider, με μέγιστη τιμή 10 (*Maximum="10"*). Η ιδιότητα *BindingContext="{x:Reference label}"* δηλώνει ότι το control με το οποίο θα διασυνδεθεί έχει όνομα “label”. Η ιδιότητα *Value="{Binding Scale, Mode=TwoWay}"* δηλώνει ότι η τιμή του Slider θα είναι διασυνδεδεμένη με την ιδιότητα Scale του BindingContext (δηλαδή του label) με τον τρόπο *Mode=TwoWay*. Αυτός ο τρόπος ορίζει ότι η τιμή Value του Slider θα τροποποιεί την τιμή Scale του label και αντιστρόφως. Ας δούμε παρακάτω τα διάφορα Binding modes:

- **OneWay:** Η Binding ιδιότητα του BindingContext τροποποιεί την επίμαχη ιδιότητα του Control
- **OneWayToSource:** Η Binding ιδιότητα του BindingContext τροποποιείται από την επίμαχη ιδιότητα του Control
- **TwoWay:** Η Binding ιδιότητα του BindingContext τροποποιεί και τροποποιείται από την επίμαχη ιδιότητα του Control

Αριστερά του Slider scaleSlider υπάρχει ένα Label που λαμβάνει με την τιμή Text με OneWay που δεν χρειάζεται να δηλώσουμε καθώς είναι η default τιμή του Binding.

3. Διασύνδεση δεδομένων με γραφικό περιβάλλον

- TwoWay: Η Binding ιδιότητα του BindingContext τροποποιεί και τροποποιείται από την επίμαχη ιδιότητα του Control

Αριστερά του Slider `scaleSlider` υπάρχει ένα Label που λαμβάνει με την τιμή Text με OneWay που δεν χρειάζεται να δηλώσουμε καθώς είναι η default τιμή του Binding.

Πλέον έχοντας κατανοήσει τον τρόπο διασύνδεσης των δεδομένων τα υπόλοιπα Sliders είναι απλά.

Το Slider `rotationSlider` τροποποιεί την ιδιότητα Rotation του label, με την οποία ορίζεται ο βαθμός περιστροφής του από το κέντρο του Label.

Το Slider `rotationXSlider` τροποποιεί την ιδιότητα RotationX του label, με την οποία ορίζεται ο βαθμός περιστροφής στον άξονα X.

Το Slider `rotationYSlider` τροποποιεί την ιδιότητα RotationY του label, με την οποία ορίζεται ο βαθμός περιστροφής στον άξονα Y.

Παράδειγμα 2

Έχοντας μια μικρή εξοικείωση από το παράδειγμα ένα θα μελετήσουμε τη διασύνδεση δύο controls τύπου Entry. Η τροποποίηση της τιμής του ενός θα καθρεφτίζεται στο άλλο.

Example 2

Copy from here:

To here:

Παρακάτω παρατίθεται ο κώδικας υλοποίησης σε xaml:

```
<StackLayout Grid.Row="0" Orientation="Horizontal">
    <Label Text="Copy from here: "></Label>
    <Entry x:Name="Text1" WidthRequest="200" Text="Type here"></Entry>
</StackLayout>
<StackLayout Grid.Row="1" Orientation="Horizontal">
    <Label Text="To here: "></Label>
    <Entry x:Name="Text2" WidthRequest="200" BindingContext="{x:Reference Name=Text1}"
Text="{Binding Path=Text }"></Entry>
</StackLayout>
```

Αρχικά δημιουργούμε ένα Entry με όνομα `Text1`, το οποίο θα δέχεται τιμές από το χρήστη.

Επειτα, δημιουργούμε ένα άλλο Entry με όνομα `Text2`, στο οποίο θέτουμε ως BindingContext το control `Text1`. Στην ιδιότητα `Text` θέτουμε ένα binding με path την ιδιότητα `Text` του `Text1` (δεν απαιτείται να γράψουμε mode, καθώς μας εξυπηρετεί το default mode=OneWay).

Πλέον, όταν τροποποιούμε την τιμή του `Text1`, ενημερώνεται αυτόματα η τιμή του `Text2`.

Παράδειγμα 3

Στο τρίτο παράδειγμα έχουμε ένα απλό Label το οποίο ενημερώνει την ιδιότητα `Text` με Binding σε μια τιμή που δηλώνεται σε κώδικα c#.

Example 3

A String from code behing

3. Διασύνδεση δεδομένων με γραφικό περιβάλλον

Παρακάτω παρατίθεται ο κώδικας σε xaml:

```
<Label Text="{Binding BindingString}"></Label>
```

To BindingString ουσιαστικά είναι μια μεταβλητή τύπου string που δηλώνεται στο αρχείο c# BindingsPage.xaml.cs:

```
public partial class BindingsPage : ContentPage
{
    public string BindingString { get; set; }
    public BindingsPage ()
    {
        InitializeComponent ();
        BindingString = "A String from code behing";
        BindingContext = this;
    }
}
```

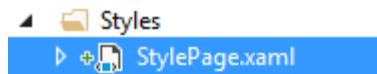
Αφού ορίσουμε τη μεταβλητή BindingString ως public, την αρχικοποιούμε στον κατασκευαστή της σελίδας και θέτουμε ως BindingContext όλης της σελίδας την κλάση BindingsPage.

4. Styles

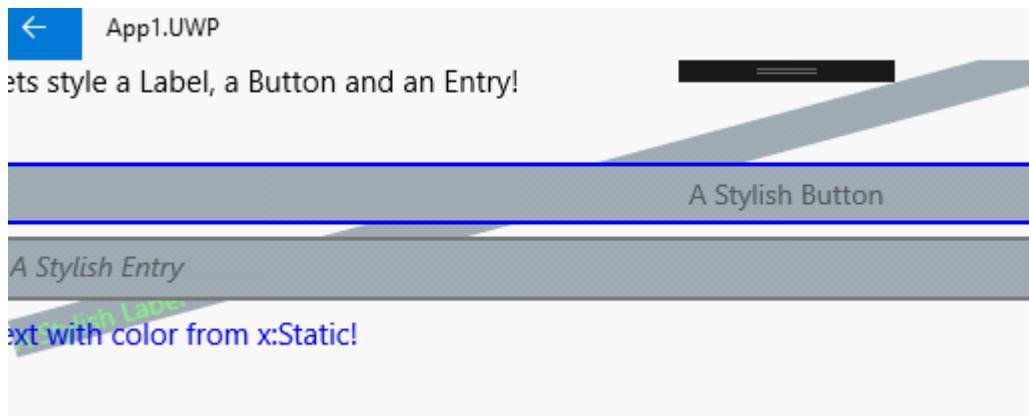
Styles

Η λογική των Styles είναι πολύ απλή: αντί να μορφοποιούμε την εμφάνιση (χρώμα, μέγεθος κτλ) κάθε control ξεχωριστά, δημιουργούμε γενικούς κανόνες μορφοποίησης τους οποίους αναθέτουμε σε πολλά controls.

Στη σελίδα StylePage.xaml υπάρχει ένα σχετικό παράδειγμα.



Τρέχοντας την εφαρμογή και ανοίγοντας τη σελίδα Styles από το κεντρικό μενού προβάλλεται η παρακάτω θόνη:



Ο κώδικας xaml του παραδείγματος έχει ως εξής:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:App1;assembly=App1"
    xmlns:sys="clr-namespace:System;assembly=mscorlib"
    x:Class="App1.Styles.StylePage" >
<ContentPage.Resources>
<ResourceDictionary>
    <Style Class="LabelTemplate" TargetType="Label">
        <Setter Property="TextColor" Value="#575e62" />
        <Setter Property="BackgroundColor" Value="#9facb3" />
        <Setter Property="FontSize" Value="14" />
        <Setter Property="Text" Value="A Stylish Label" />
    </Style>
    <Style Class="ButtonTemplate" TargetType="Button">
        <Setter Property="TextColor" Value="#575e62" />
        <Setter Property="BackgroundColor" Value="#9facb3" />
        <Setter Property="FontSize" Value="14" />
        <Setter Property="Text" Value="A Stylish Button" />
        <Setter Property="BorderColor" Value="Blue" />
        <Setter Property="BorderRadius" Value="5" />
        <Setter Property="BorderWidth" Value="2" />
    </Style>
```

4. Styles

```
<x:Double x:Key="rotationAngle">-15</x:Double>
    <Color x:Key="textColor1">Red</Color>
    <OnPlatform x:Key="textColor"
        x:TypeArguments="Color"
        iOS="Red"
        Android="Aqua"
        WinPhone="#80FF80" />
    </ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
    <StackLayout>
        <Label Text="Lets style a Label, a Button and an Entry!" />
        <Label StyleClass="LabelTemplate"
            Rotation="{StaticResource rotationAngle}" TextColor="{StaticResource textColor}"
        ></Label>
        <Button StyleClass="ButtonTemplate" /></Button>
        <Entry StyleClass="EntryTemplate" /></Entry>
        <Label Text="Text with color from x:Static!" TextColor="{x:Static
local:ColorsList.ForegroundColor}" />
    </StackLayout>
</ContentPage.Content>
</ContentPage>
```

Αν παρατηρήσουμε στην αρχή της σελίδας (τύπου ContentPage) υπάρχει η δήλωση

<ContentPage.Resources> που σημαίνει ότι εδώ δηλώνονται οι πόροι (Resources) που θα χρησιμοποιηθούν από τα στοιχεία της σελίδας. Μέσα σε αυτή περιέχεται η δήλωση <ResourceDictionary> που σημαίνει ότι εδώ δηλώνεται το λεξικό των πόρων, δηλαδή οι ίδιοι οι πόροι, όπως οι μορφοποιήσεις, οι στατικοί αριθμοί κτλ.

Αρχικά υπάρχει ένα Label που δεν περιέχει καμία μορφοποίηση και φέρει το κείμενο: “ *Lets style a Label, a Button and an Entry!*”.

Επειτα υπάρχει ένα Label που φέρει μεταξύ άλλων την ιδιότητα **StyleClass** . Η ιδιότητα αυτή σημαίνει ότι πρέπει να πάρεις το στυλ της μορφοποίησής σου από μια κλάση Style (μορφοποίησης) που έχει το όνομα *“LabelTemplate”* . Ανατρέχουμε μέσα στο ResourceDictionary και βλέπουμε την κλάση Style με όνομα *LabelTemplate*.

```
<Style Class="LabelTemplate" TargetType="Label">
    <Setter Property="TextColor" Value="#575e62" />
    <Setter Property="BackgroundColor" Value="#9facb3" />
    <Setter Property="FontSize" Value="14" />
    <Setter Property="Text" Value="A Stylish Label" />
</Style>
```

Η κλάση Style δηλώνεται με το <Style... και περιέχει το όνομα της κλάσης Class=“LabelTemplate” και το control το οποίο αφορά TargetType=“Label” (αυτή η κλάση δηλαδή εφαρμόζεται σε Label).

Έπειτα, μέσα στο Style υπάρχουν πολλές δηλώσεις τύπου Setter, η οποίες αναθέτουν μια τιμή σε μια ιδιότητα. Η πρώτη Setter <Setter Property="TextColor" Value="#575e62" /> μεταφράζεται ως εξής: Βάλε στην ιδιότητα *TextColor* του Label που θα χρησιμοποιηθεί αυτό το Style την τιμή #575e62 (HEX color). Ομοίως οι υπόλοιπες δηλώσεις Setter τροποποιούν άλλες ιδιότητες του Label, όπως *BackgroundColor*, *FontSize*, μέχρι και το *Text* (κείμενο).

4. Styles

Αφού εφαρμόσαμε το πρώτο μας Style στο Label μπορούμε να τροποποιήσουμε και άλλες ιδιότητες σε αυτό με τη χρήση του StaticResource. Το StaticResource δηλώνει τη σύνδεση με μια στατική τιμή διάφορων τύπων (Color, Double κτλ) που δηλώνονται μέσα στο ResourceDictionary. Στο παράδειγμά μας βλέπουμε ότι η ιδιότητα *Rotation* (περιστροφής κειμένου) συνδέεται με τη χρήση του StaticResource (δήλωση {StaticResource ..ονομα...}) με το όνομα στατικού πόρου rotationAngle που αντιστοιχεί στη δήλωση <x:Double x:Key="rotationAngle">-15</x:Double>. Δηλαδή, η περιστροφή θα συνδεθεί με έναν αριθμό τύπου double με τιμή -15.

Ομοίως η ιδιότητα *TextColor* συνδέεται με το *textColor* που δηλώνεται στο ResourceDictionary ως εξής:

```
<OnPlatform x:Key="textColor"
    x:TypeArguments="Color"
    iOS="Red"
    Android="Aqua"
    WinPhone="#80FF80" />
```

Το ενδιαφέρον αυτής της περίπτωσης είναι ότι μπορούμε να προσαρμόσουμε τις τιμές ανάλογα με τον τύπο συσκευής που τρέχει την εφαρμογή μας. Έτσι με τη δήλωση <OnPlatform λέμε ότι ανάλογα με την πλατφόρμα (iOS, Android, WinPhone) βάλε στο κλειδί *textColor* τύπου x:TypeArguments="Color" τις αντίστοιχες τιμές (πχ αν τρέχεις σε iOS βάλε κόκκινο χρώμα κτλ).

Το επόμενο control που συναντάμε είναι το Button που φέρει *StyleClass* με όνομα *ButtonTemplate*.

Βλέποντας την κλάση *style* *ButtonTemplate* στο ResourceDictionary βλέπουμε ότι μορφοποιεί πολλές τιμές, όπως το *TextColor*, *FontSize*, *BorderColor*, *BorderRadius* κτλ.

Το επόμενο control είναι ένα *Entry* που φέρει το *Style* με όνομα *EntryTemplate*. Παρατηρώντας το ResourceDictionary δεν μπορούμε να εντοπίσουμε το συγκεκριμένο *Style*. Η Xamarin.Forms δίνει τη δυνατότητα στον προγραμματιστή να ορίσει *Style* όχι μόνο τοπικά σε μια σελίδα, αλλά και καθολικά *Style* που να ισχύουν σε όλες τις σελίδες της εφαρμογής. Για να ορίσουμε ένα καθολικό *Style* ανοίγουμε το App.xaml και βλέπουμε τον παρακάτω κώδικα:

```
<Application.Resources>
    <ResourceDictionary>
        <Style Class="EntryTemplate" TargetType="Entry">
            <Setter Property="TextColor" Value="#575e62" />
            <Setter Property="BackgroundColor" Value="#9facb3" />
            <Setter Property="FontSize" Value="14" />
            <Setter Property="Text" Value="A Stylish Entry" />
            <Setter Property="FontAttributes" Value="Italic" />
        </Style>
    </ResourceDictionary>
</Application.Resources>
```

Με τον ίδιο τρόπο που ορίζουμε *Style* σε μια σελίδα, ορίζουμε και τα καθολικά *Style* με τη μόνη διαφορά ότι το ResourceDictionary στο οποίο ανήκουν βρίσκεται μέσα στο Application.Resources.

Το τελευταίο control που παρατηρούμε έχει *Text* "Text with color from x:Static!" και συνδέει την ιδιότητα *TextColor* με την τιμή "{x:Static local:ColorsList.ForegroundColor}".

Η τιμή x:Static χρησιμοποιείται για να δηλώσει τη σύνδεση με μία μεταβλητή τύπου static που βρίσκεται στη διαδρομή local:ColorsList.ForegroundColor. Η τιμή local ορίζεται μέσα στη δήλωση του ContentPage ως εξής:

```
xmlns:local="clr-namespace:App1;assembly=App1"
xmlns:sys="clr-namespace:System;assembly=mscorlib"
```

Η δήλωση xmlns:local σημαίνει : το namespace App1 και την assembly (Όνομα προγράμματος) App1 μέσα στο πρόγραμμα μπορείς να τα αποκαλείς ως local.

Στη δεύτερη γραμμή, η τιμή xmlns:sys σημαίνει: το namespace System και την assembly mscorelib μπορείς να την αποκαλείς μέσα στο πρόγραμμα ως sys. Επομένως με το sys μπορούμε να καλούμε βιβλιοθήκες, όπως τη Math κτλ.

4. Styles

Γνωρίζοντας πλέον ότι η local αναφέρεται στο πρόγραμμά μας, η υπόλοιπη διαδρομή που περιέχει την απαιτούμενη τιμή είναι η *ColorsList.ForegroundColor*. Το ColorList είναι ένα αρχείο .cs που δημιουργήσαμε και περιέχει τη static κλάση ColorsList και αυτή τη static μεταβλητή ColorsList. Παρακάτω παρατίθεται ο σχετικός κώδικας c#:

```
public static class ColorsList
{
    public static readonly Color ForegroundColor =
        Device.OnPlatform(Color.Black, Color.White, Color.Blue);
}
```

Με την Device.OnPlatform αποδίδουμε ξεχωριστή τιμή, τύπου Color, ανάλογα με την πλατφόρμα. Έτσι για παράδειγμα, στα Windows 10 το ForegroundColor θα είναι Μπλε.

5. Animation

Animation

Με το Xamarin.Forms μπορούμε να εφαρμόσουμε διάφορα Animation στα controls ώστε να βελτιώσουμε την εμπειρία χρήστη και το περιβάλλον διεπαφής.

Ανοίγουμε από το κεντρικό μενού το παράδειγμα Animations και προβάλλεται η παρακάτω οθόνη:



Ο κώδικας του παραδείγματος του Animation βρίσκεται στο αρχείο AnimationPage.xaml.

```
<StackLayout>
    <Button Text="RotateTo" Clicked="Button_Clicked"></Button>
    <Button Text="RotateXTo" Clicked="Button_Clicked_7"></Button>
    <Button Text="RelRotateTo" Clicked="Button_Clicked_1"></Button>
    <Button Text="ScaleTo" Clicked="Button_Clicked_2"></Button>
    <Button Text="RelScaleTo" Clicked="Button_Clicked_3"></Button>
    <Button Text="Scaling and Rotation with Anchors" Clicked="Button_Clicked_4"></Button>
    <Button Text="TranslateTo" Clicked="Button_Clicked_5"></Button>
    <Button Text="Fading" Clicked="Button_Clicked_6"></Button>

    <Label x:Name="AnimationLabel" HorizontalTextAlignment="Center" Text="Welcome to Xamarin Forms!" />
</StackLayout>
```

Παρατηρούμε ότι ο κώδικας είναι πολύ απλός, αφού περιέχει ένα StackLayout μέσα στο οποίο τοποθετούνται κατακόρυφα τα στοιχεία Button και ένα Label. Κάθε Button εφαρμόζει ένα Animation στο Label.

Το πρώτο Button με Text “RotateTo” περιστρέφεται γύρω από τον άξονά του. Για να γίνει αυτό γράφουμε τον c# κώδικα:

```
private async void Button_Clicked(object sender, EventArgs e)
{
    AnimationLabel.RotateTo(360, duration);
}
```

Ο κώδικας δηλώνει τη συνάρτηση RotateTo στο AnimationLabel με 360 μοίρες και duration 2 sec. Η RotateTo περιστρέφει το control κατά X μοίρες σε Y δευτερόλεπτα.

5. Animation

Το δεύτερο Button με Text RotateXTo περιστρέφει το Label κατά 360 μοίρες, 251 φορές σε 2 sec χρησιμοποιώντας τον παρακάτω κώδικα:

```
AnimationLabel.RotateXTo(251 * 360, duration);
```

Το τρίτο Button με Text RelRotateTo περιστρέφει το Label σε σχέση με τη θέση του κατά 360 μοίρες σε 2 sec χρησιμοποιώντας τον παρακάτω κώδικα:

```
AnimationLabel.RelRotateTo(360, duration);
```

Το Button ScaleTo αλλάζει το μέγεθος του Label κατά 2 μονάδες σε 2 sec χρησιμοποιώντας τον παρακάτω κώδικα:

```
AnimationLabel.ScaleTo(2, duration);
```

Το Button RelScaleTo τροποποιεί με όμοιο τρόπο το μέγεθος αλλά σε σχέση με τη θέση του.

```
AnimationLabel.RelScaleTo(2, duration);
```

Το Button με Text “Scaling and Rotation with Anchors” περιστρέφει και αλλάζει το μέγεθος του Label ταυτόχρονα παίρνοντας ως σημείο αναφορά ένα συγκεκριμένο σημείο:

```
AnimationLabel.AnchorY = (Math.Min(this.Width, this.Height) / 2) / AnimationLabel.Height;
```

```
AnimationLabel.RotateTo(360, duration);
```

Το Button με Text TranslateTo αλλάζει τη θέση του Label προς το σημείο που απέχει -100 από το X και -100 από το Y σε 2 sec, χρησιμοποιώντας τον παρακάτω κώδικα:

```
AnimationLabel.TranslateTo(-100, -100, duration);
```

Το Button με Text Fading αλλάζει την ορατότητα του Label σε 0, δηλαδή το εξαφανίζει (τιμές 0-1) σε χρόνο 4 sec, χρησιμοποιώντας τον παρακάτω κώδικα:

```
AnimationLabel.FadeTo(0, 4000);
```

6. Σύστημα Πλοήγησης (Navigation)

Navigation

Ένα από τα πιο σημαντικά θέματα στην ανάπτυξη κινητών εφαρμογών είναι η δημιουργία πλοήγησης μεταξύ των παραθύρων.

Για να κατανοήσουμε τη διαδικασία πλοήγησης πατάμε το κουμπί Navigation στο κεντρικό μενού και προβάλλουμε το παράθυρο του παραδείγματος.

Δημιουργία πλοήγησης

--NAVIGATION---

Navigation

Βλέπουμε ότι με το πάτημα του Button στο κεντρικό μενού μεταβήκαμε σε μια νέα σελίδα. Αυτό είναι ένα παράδειγμα Navigation. Ας δούμε τον κώδικα xaml που εκτελείται στην εκκίνηση του λογισμικού στο αρχείο App.xaml.cs:

```
public App()
{
    InitializeComponent();

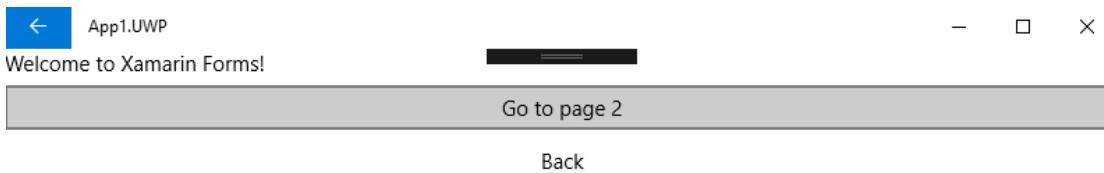
    MainPage = new NavigationPage(new MainPage());
}
```

Βλέπουμε ότι ορίζεται ως MainPage (αρχική σελίδα) ένα νέο στιγμιότυπο της κλάσης NavigationPage που δέχεται ως όρισμα μια νέα σελίδα τύπου MainPage (θα μπορούσε να είναι οποιαδήποτε άλλη σελίδα). Τι είναι όμως ένα NavigationPage; Είναι ένα είδος σελίδας που χειρίζεται την πλοήγηση και την εμπειρία χρήστη σε μία στοίβα σελίδων. Κάθε φορά που πλοηγούμαστε σε μια σελίδα προσθέτουμε στη στοίβα, ενώ όταν μεταβαίνουμε ένα βήμα πίσω τότε αφαιρείται μια σελίδα από τη στοίβα.

Μόλις ανοίξει η σελίδα MainPage (ενσωματωμένη μέσα σε μια NavigationPage) τότε προστίθεται μέσα στη στοίβα η πρώτη σελίδα (MainPage). Πατώντας το κουμπί Navigation μέσα στη MainPage εκτελείται ο παρακάτω κώδικας c#:

```
Navigation.PushAsync(new NavigationPage1());
```

Με τον κώδικα, προσθέτουμε στη στοίβα (Navigation.PushAsync) τη σελίδα NavigationPage1. Επομένως προστίθεται μέσα στη στοίβα η δεύτερη σελίδα τύπου NavigationPage1.



Για να επιστρέψουμε πίσω, πρέπει να πατήσουμε το κουμπί Back, το οποίο θα αφαιρέσει την τελευταία σελίδα από τη στοίβα:

```
await Navigation.PopAsync();
```

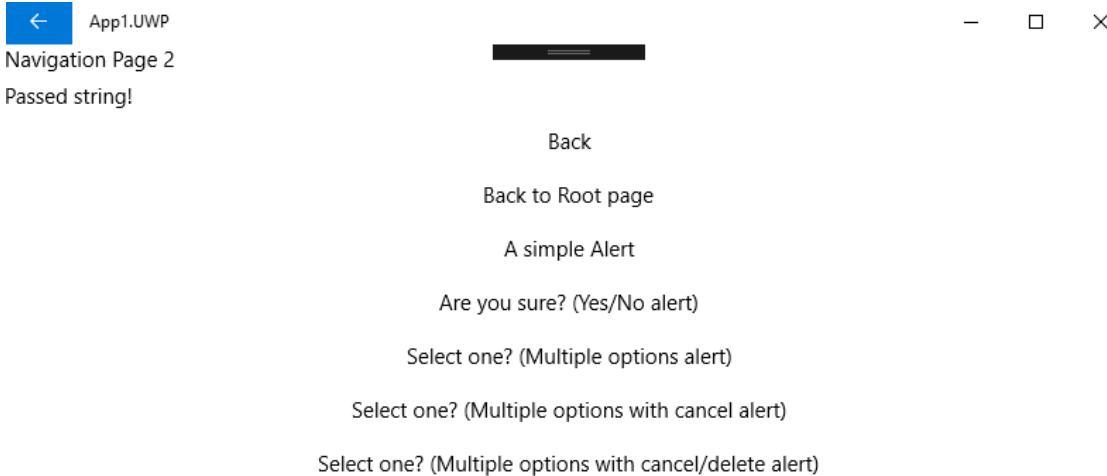
Με την εντολή Navigation.PopAsync() αφαιρείται από τη στοίβα η τελευταία NavigationPage και επιστρέφουμε πίσω.

Για να μεταβούμε στη σελίδα 2 πρέπει να πατήσουμε το κουμπί "Go to page 2" το οποίο εκτελεί το c# κώδικα:

6. Σύστημα Πλοήγησης (Navigation)

```
await Navigation.PushAsync(new NavigationPage2("Passed string!));
```

Με το Navigation.PushAsync προσθέτουμε στη στοίβα πλοήγησης μια νέα σελίδα τύπου NavigationPage2 περνώντας στον κατασκευαστή την τιμή "Passed string!".



```
<StackLayout>
    <Label Text="Navigation Page 2" />
    <Label x:Name="PassedValue" Text="Navigation Page 2" />
    <Button x:Name="GoToPrevious" Text="Back"
Clicked="OnPreviousPageButtonClicked"></Button>
    <Button x:Name="GoToRoot" Text="Back to Root page"
Clicked="OnRootPageButtonClicked"></Button>
    <Button x:Name="AlertBtn" Text="A simple Alert" Clicked="AlertBtn_Clicked"></Button>
    <Button x:Name="AlertYesNo" Text="Are you sure? (Yes/No alert)"
Clicked="OnAlertYesNoClicked"></Button>
    <Button x:Name="AlertMultipleChoice" Text="Select one? (Multiple options alert)"
Clicked="OnActionSheetSimpleClicked"></Button>
    <Button x:Name="AlertWithCancel" Text="Select one? (Multiple options with cancel alert)"
Clicked="OnActionSheetSimpleClicked"></Button>
    <Label x:Name="AlertAnswer" Text="" />
</StackLayout>
```

Όταν ανοίξει η σελίδα NavigationPage2 εκτελείται ο κατασκευαστής με τον κώδικα:

```
public NavigationPage2 ()
{
    InitializeComponent ();
}
public NavigationPage2(string passedValue)
{
    InitializeComponent();
    PassedValue.Text = passedValue;
}
```

Βλέπουμε ότι υπάρχουν δύο κατασκευαστές, ο ένας χωρίς παράμετρο και ο άλλος με παράμετρο το passedValue τύπου string. Αφού φορτωθεί το γραφικό περιβάλλον με την εντολή InitializeComponent(); ακολουθεί η ανάθεση της τιμής στην ιδιότητα Text του Label PassedValue.

6. Σύστημα Πλοήγησης (Navigation)

Το Button με κείμενο Back μας επιστρέφει στην προηγούμενη σελίδα, αφαιρώντας από τη στοίβα την τελευταία σελίδα με τον κώδικα:

```
await Navigation.PopAsync();
```

Πατώντας το κουμπί “Back to root page” μεταβαίνουμε στην πρώτη σελίδα της στοίβας με τον κώδικα:

```
await Navigation.PopToRootAsync(true);
```

Το Navigation.PopToRootAsync αφαιρεί όλες τις σελίδες από τη στοίβα αφήνοντας μόνο την αρχική. Για να χρησιμοποιήσουμε animation κατά την αλλαγή σελίδας περνάμε ως όρισμα το true.

Το κουμπί με όνομα AlertBtn ανοίγει ένα Alert. Τα Alert είναι μικρές ειδοποιήσεις που ενημερώνουν το χρήστη ή λαμβάνουν μια σύντομη επιλογή του μέσα από μια λίστα επιλογών. Ο κώδικας που εκτελείται για:

```
DisplayAlert("Alert", "You have been alerted", "OK");
```

Το DisplayAlert ανοίγει μια σύντομη ειδοποίηση στο χρήστη. Το πρώτο όρισμα είναι ο Τίτλος, το δεύτερο το κείμενο και το τρίτο το όνομα του κουμπιού κλεισίματος.

Alert

Alert

You have been alerted

OK

Το κουμπί με όνομα AlertYesNo ανοίγει ένα παράθυρο ειδοποίησης επιτρέποντας να λάβουμε μια αντίδραση τύπου Ναι/Όχι από το χρήστη.

Question?

Question?

Would you like to play a game

Yes

No

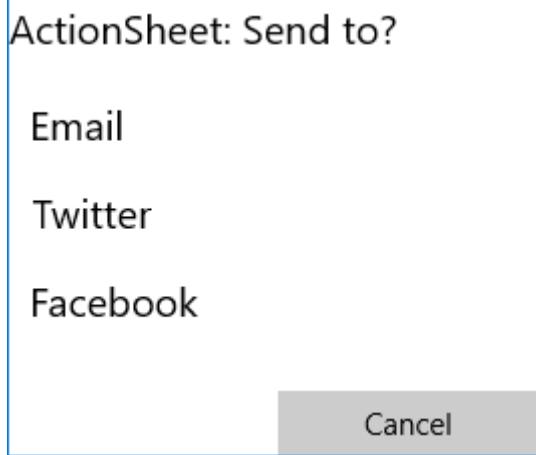
Ο κώδικας xaml c# που εκτελείται όταν πατάμε το κουμπί είναι:

```
var answer = await DisplayAlert("Question?", "Would you like to play a game", "Yes", "No");
AlertAnswer.Text="Answer: " + answer;
```

Το DisplayAlert έχει ως τελευταίες παραμέτρους τα κουμπιά Yes και No και επιστρέφει την τιμή επιλογής στη μεταβλητή answer. Τέλος, αναθέτουμε την τιμή answer στο κείμενο του Label AlertAnswer.

Με το κουμπί AlertMultipleChoice μπορούμε να προβάλλουμε ένα Alert με μια λίστα πολλαπλών επιλογών και να λάβουμε την επιλογή του χρήστη.

6. Σύστημα Πλοήγησης (Navigation)



Ο κώδικας xaml είναι ο εξής:

```
var action = await DisplayActionSheet("ActionSheet: Send to?", "Cancel", null, "Email", "Twitter",  
"Facebook");  
AlertAnswer.Text = "Action: " + action;
```

Για να μπορέσουμε να προβάλλουμε τη λίστα επιλογών χρησιμοποιούμε το DisplayActionSheet με ορίσματα: το κείμενο, το κείμενο του κουμπιού ακύρωσης, το κείμενο του κουμπιού καταστροφής (null στην περίπτωσή μας) και τα διαφορετικά strings με τις επιλογές. Όταν λάβουμε την απάντηση, την αναθέτουμε στη μεταβλητή action που με τη σειρά της την προβάλλουμε στο Label AlertAnswer. Τα τελευταία κουμπιά AlertWithCancel και AlertWithCancelDelete επιτρέπουν την προβολή πολλαπλών επιλογών με δυνατότητα επιλογής ακύρωσης και διαγραφής ανάλογα με το αν θα περάσουμε το null ή τιμή στις αντίστοιχες παραμέτρους:

With Cancel only

```
var action = await DisplayActionSheet("ActionSheet: Send to?", "Cancel", null, "Email", "Twitter",  
"Facebook");  
AlertAnswer.Text = "Action: " + action;
```

With Cancel and Delete

```
var action = await DisplayActionSheet("ActionSheet: SavePhoto?", "Cancel", "Delete", "Photo Roll",  
"Email");  
AlertAnswer.Text = "Action: " + action;
```

7. Database

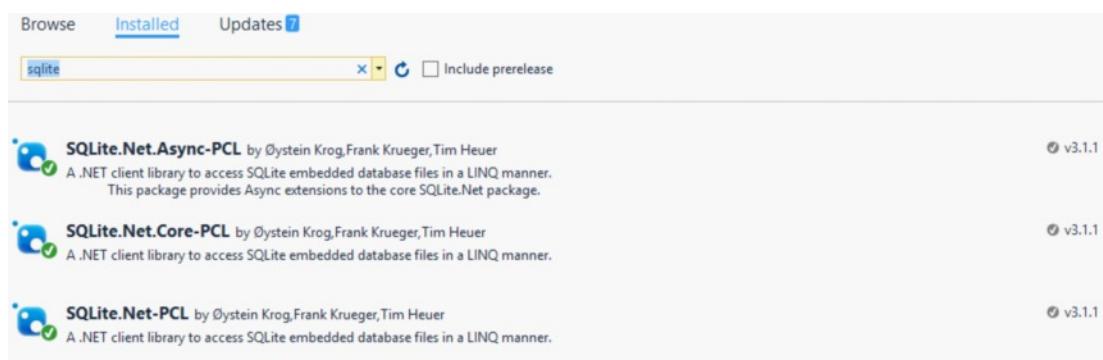
Database

Ένα από τα πιο συχνά προβλήματα που προκύπτουν κατά την ανάπτυξη μια εφαρμογής είναι η αποθήκευση δεδομένων τοπικά ώστε να είναι προσβάσιμα ακόμα και χωρίς σύνδεση στο Internet. Ο πιο συχνός τρόπος αποθήκευσης δεδομένων είναι η χρήση Βάσης Δεδομένων.

Για να μελετήσουμε τον τρόπο διαχείρισης μιας Βάσης Δεδομένων θα ανοίξουμε το solution UniBooks.

Το πρώτο βήμα είναι να προσθέσουμε τις κατάλληλες βιβλιοθήκες της SQLite. Για να γίνει αυτό κάνουμε δεξί κλικ στο project UniBooks και επιλέγουμε “Manage NuGet Packages”.

Στην καρτέλα Browse γράφουμε “SQLite.Net-PCL” και πατάμε Install . Επαναλαμβάνουμε τη διαδικασία για κάθε project ξεχωριστά (Android,iOS και UWP).



Το δεύτερο βήμα αφού έχουμε εγκαταστήσει σε όλα τα project την SQLite, είναι να προσθέσουμε μια κλάση που θα υλοποιεί το Interface ISqlitePlatformProvider, αφού κάθε λειτουργικό σύστημα έχει διαφορετικές λειτουργίες ως προς την πρόσβαση στην SQLite (διαφορετική διαδρομή αποθήκευσης κτλ.).

namespace UniBooks.Data

```
{  
    public interface ISqlitePlatformProvider  
    {  
        ISQLitePlatform GetPlatform();  
    }  
}
```

Για το project **UWP** δημιουργούμε μια κλάση στο αρχείο SqlitePlatformProvider.cs που περιέχει τον κώδικα c#:

```
[assembly: Dependency(typeof(SqlitePlatformProvider))]  
namespace UniBooks.UWP  
{  
    public class SqlitePlatformProvider : ISqlitePlatformProvider  
    {  
        public ISQLitePlatform GetPlatform()  
        {  
            return new SQLitePlatformWinRT();  
        }  
    }  
}
```

Με την γραμμή [assembly: Dependency(typeof(SqlitePlatformProvider))] δηλώνεται στην DependencyService η υλοποίηση του Interface ISqlitePlatformProvider από το UWP. Το DependencyService επιτρέπει στην εφαρμογή να κάνει κλήσεις λειτουργιών ενός λειτουργικού συστήματος μέσα από τον κοινό κώδικα. Αυτό επιτρέπει την εφαρμογή να έχει πρόσβαση σε όλες τις λειτουργίες του λειτουργικού (πχ του Android). Στην πράξη ορίζουμε το Interface στον κοινό μας κώδικα (πχ ISqlitePlatformProvider) και το DependencyService βρίσκει την υλοποίησή που είναι κατάλληλη για τη συσκευή που εκτελεί το app μέσα από τις υλοποιήσεις στα διάφορα project των λειτουργικών συστημάτων.

7. Database

Το μόνο που χρειάζεται να υλοποιήσουμε είναι η συνάρτηση GetPlatform που επιστρέφει ένα Interface ISQLitePlatform ως “new SQLitePlatformWinRT();” Δηλαδή SQLitePlatform ειδικά για τα Windows 10.

Για το project Android ο αντίστοιχος κώδικας c# θα είναι:

```
[assembly: Xamarin.Forms.Dependency(typeof(SqlitePlatformProvider))]
namespace UniBooks.Droid
{
    public class SqlitePlatformProvider : ISqlitePlatformProvider
    {
        public ISQLitePlatform GetPlatform()
        {
            return new SQLitePlatformAndroid();
        }
    }
}
```

Για το project iOS ο αντίστοιχος κώδικας c# θα είναι :

```
[assembly: Xamarin.Forms.Dependency(typeof(SqlitePlatformProvider))]
namespace UniBooks.iOS
{
    public class SqlitePlatformProvider : ISqlitePlatformProvider
    {
        public ISQLitePlatform GetPlatform()
        {
            return new SQLitePlatformIOS();
        }
    }
}
```

Το τρίτο βήμα είναι με όμοιο τρόπο (δηλαδή χρησιμοποιώντας το DependencyService) να δηλώσουμε ένα Interface με όνομα ILocalFilePathProvider

```
namespace UniBooks.Data
{
    public interface ILocalFilePathProvider
    {
        string GetLocalPathToFile(string fileName);
    }
}
```

Η υλοποίηση του Interface δέχεται το όνομα της Β.Δ. και επιστρέφει τη διαδρομή αποθήκευσής της στο τρέχων λειτουργικό σύστημα.

To Interface υλοποιείται στο Android με τον παρακάτω κώδικα:

```
[assembly: Xamarin.Forms.Dependency(typeof(LocalFilePathProvider))]
namespace UniBooks.Droid
{
    public class LocalFilePathProvider : ILocalFilePathProvider
    {
        public string GetLocalPathToFile(string fileName)
        {
            return
Path.Combine(System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal), fileName);
        }
    }
}
```

7. Database

Η υλοποίηση στο iOS έχει ως εξής:

```
[assembly: Xamarin.Forms.Dependency(typeof(LocalFilePathProvider))]
namespace UniBooks.iOS
{
    public class LocalFilePathProvider : ILocalFilePathProvider
    {
        public string GetLocalPathToFile(string fileName)
        {
            return Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Personal),
fileName);
        }
    }
}
```

Τέλος, η υλοποίηση στο Windows 10 (UWP) έχει ως εξής:

```
[assembly: Xamarin.Forms.Dependency(typeof(LocalFilePathProvider))]
```

```
namespace UniBooks.UWP
{
    public class LocalFilePathProvider : ILocalFilePathProvider
    {
        public string GetLocalPathToFile(string fileName)
        {
            return Path.Combine(ApplicationData.Current.LocalFolder.Path, fileName);
        }
    }
}
```

Στο τέταρτο βήμα αφού έχουμε τελειώσει με τον τρόπο απόκτησης πρόσβασης σε κάθε λειτουργικό σύστημα ζεχωριστά θα μελετήσουμε τον τρόπο υλοποίηση της κλάσης διαχείρισης της Β.Δ. μέσα στον κοινό κώδικα.

Δημιουργούμε μια νέα κλάση με όνομα BooksDataAccess . Παρακάτω παρατίθεται ο κώδικας c#:

```
public class BooksDataAccess
{
    private static object collisionLock = new object();
    public ObservableCollection<ABook> Books { get; set; }
    SQLiteAsyncConnection _connection;
    public BooksDataAccess(string databaseName)
    {
        var pathToDatabaseFile =
DependencyService.Get<ILocalFilePathProvider>().GetLocalPathToFile(databaseName);
        var platform = DependencyService.Get<ISqlitePlatformProvider>().GetPlatform();
        _connection = new SQLiteAsyncConnection(() =>
            new SQLiteConnectionString(pathToDatabaseFile, false)));
        _connection.CreateTableAsync<ABook>();
    }

    public async Task<List<ABook>> GetAllBooks()
    {
        var s=await _connection.Table<ABook>().ToListAsync();
        return s;
    }
}
```

7. Database

```
public Task<List<ABook>> GetFilteredBooks(string bookName)
{
    lock (collisionLock)
    {
        var query = from cust in _connection.Table<ABook>()
                    where cust.title == bookName
                    select cust;
        return query.ToListAsync();
    }
}
public Task<List<ABook>> GetFilteredBooks()
{
    lock (collisionLock)
    {
        return _connection.QueryAsync<ABook>(
            "SELECT * FROM ABook WHERE title = 'self'");
    }
}

public async Task<ABook> GetBook(string id)
{
    return await _connection.Table<ABook>().Where(book => book.Id == id).
        FirstOrDefaultAsync();
}

public async Task<string> SaveBook(ABook bookInstance)
{
    var b =await GetBook(bookInstance.Id);
    if (b!= null)
    {
        _connection.UpdateAsync(bookInstance);
        return bookInstance.Id;
    }
    else
    {
        _connection.InsertAsync(bookInstance);
        return bookInstance.Id;
    }
}

public void SaveAllBooks()
{
    lock (collisionLock)
    {
        foreach (var bookInstance in this.Books)
        {
            if (bookInstance.Id != "")
            {
                _connection.UpdateAsync(bookInstance);
            }
            else
            {
                _connection.InsertAsync(bookInstance);
            }
        }
    }
}

public string DeleteBook(ABook bookInstance)
{
    var id = bookInstance.Id;
    if (id != "")
    {
        lock (collisionLock)
        {
            _connection.DeleteAsync<ABook>(id);
        }
    }
    this.Books.Remove(bookInstance);
}
```

```

public string DeleteBook(ABook bookInstance)
{
    var id = bookInstance.Id;
    if (id != "")
    {
        lock (collisionLock)
        {
            _connection.DeleteAsync<ABook>(id);
        }
    }
    this.Books.Remove(bookInstance);
    return id;
}

public void DeleteAllBooks()
{
    lock (collisionLock)
    {
        _connection.DropTableAsync<ABook>();
        _connection.CreateTableAsync<ABook>();
    }
    this.Books = null;
    var bks = _connection.Table<ABook>().ToListAsync();
    this.Books = new ObservableCollection<ABook>(bks.Result);
}

```

Στον κατασκευαστή της BooksDataAccess περνάμε το όνομα της Βάσης Δεδομένων. Με τη βοήθεια της DependencyService και της συνάρτησης GetLocalPathToFile λαμβάνουμε τη διαδρομή αποθήκευσης της Β.Δ. στο λειτουργικό σύστημα.

Επίσης, πάλι με τη βοήθεια της DependencyService και της συνάρτησης GetPlatform λαμβάνουμε την πλατφόρμα της SQLite ειδικά για το τρέχων λειτουργικό σύστημα.

Έπειτα δημιουργούμε ένα νέο στιγμιότυπο της SQLiteAsyncConnection, στον κατασκευαστή της οποίας περνάμε τη διαδρομή αποθήκευσης και την πλατφόρμα. Η εν λόγω κλάση μας επιστρέφει το connection, με το οποίο πλέον θα διαχειριζόμαστε την Βάση Δεδομένων μας.

Η εντολή `_connection.CreateTableAsync<ABook>()`; δημιουργεί έναν πίνακα με τη δομή της κλάσης ABook σε περίπτωση που δεν υπάρχει. Παρακάτω διαλαμβάνεται η δομή της κλάσης ABook που βρίσκεται μέσα στο αρχείο GoogleBooksService.cs:

```

public class ABook
{
    [PrimaryKey]
    public string Id
    {
        get;
        set;
    }
    public string saleability
    {
        get;
        set;
    }
    public double? listPrice
    {
        get;
        set;
    }
}

```

```

public string title
{
    get;
    set;
}
public string subtitle
{
    get;
    set;
}
public string authors
{
    get;
    set;
}
public string publisher
{
    get;
    set;
}
public string publishedDate
{
    get;
    set;
}
public string description
{
    get;
    set;
}
public int? pageCount
{
    get;
    set;
}
public double? averageRating
{
    get;
    set;
}
public int? ratingsCount
{
    get;
    set;
}
public string smallThumbnail
{
    get;
    set;
}
public string thumbnail
{
    get;
    set;
}
}
}

```

Αφού έχουμε κατασκευάσει τους πίνακες της Βάσης στον κατασκευαστή, μένει να υλοποιήσουμε τις βασικές συναρτήσεις, όπως προσθήκη βιβλίου, ερώτημα λήψης λίστας βιβλίων κτλ.

7. Database

Με τη συνάρτηση GetAllBooks λαμβάνουμε τη λίστα βιβλίων
_connection.Table<ABook>().ToListAsync(); Λαμβάνοντας τον πίνακα ABook από το connection ως λίστα.
Με τη συνάρτηση GetFilteredBooks λαμβάνουμε μια λίστα βιβλίων των οποίων το όνομα είναι bookName.
Για να γίνει αυτό εκτελείται ο κώδικας linq (η σύνταξη μοιάζει με την sql):

```
var query = from cust in _connection.Table<ABook>()
            where cust.title == bookName
            select cust;
```

Δηλαδή, από τη μεταβλητή cust που θα πάρεις από τον πίνακα ABook, όπου ο τίτλος είναι bookName φέρει το ίδιο το cust και βάλτο σε μια μεταβλητή query.

Αν θελήσουμε να γράψουμε κατευθείαν sql μπορούμε να δούμε το παράδειγμα τη GetAllBooks (χωρίς παραμέτρους) η οποία εκτελεί την εντολή sql:

```
_connection.QueryAsync<ABook>(
    "SELECT * FROM ABook WHERE title = 'self'");
```

Δηλαδή, φέρει τα ABook με τίτλο='self'.

Κάθε βιβλίο φέρει μια ταυτότητα που αποθηκεύεται στη μεταβλητή id. Για να φέρουμε ένα βιβλίο με βάση το id εκτελούμε τη συνάρτηση GetBook που περιέχει τον κώδικα:

```
_connection.Table<ABook>().Where(book => book.Id == id).FirstOrDefaultAsync();
```

Δηλαδή από το connection φέρει τους πίνακες ABook όπου το book.Id==id και από όλη τη λίστα φέρει μόνο το πρώτο ή κανένα (FirstOrDefaultAsync).

Για να αποθηκεύσουμε ένα βιβλίο χρησιμοποιούμε τη συνάρτηση SaveBook με παράμετρο το βιβλίο που θέλουμε να αποθηκεύσουμε. Με την GetBook αρχικά αναζητούμε το βιβλίο που έχει ίδιο id στη Βάση Δεδομένων. Αν υπάρχει τότε εκτελείται η εντολή _connection.UpdateAsync(bookInstance), η οποία ενημερώνει την εγγραφή του βιβλίου στη Βάση Δεδομένων με τα νέα στοιχεία. Αν δεν υπάρχει το βιβλίο τότε εκτελείται ο κώδικας _connection.InsertAsync(bookInstance) με τον οποίο εισάγεται μια νέα εγγραφή στη Βάση Δεδομένων με τα στοιχεία του βιβλίου.

Με την εντολή SaveAllBooks μπορούμε με όμοιο τρόπο να αποθηκεύσουμε/ενημερώσουμε τη λίστα βιβλίων Books στη ΒΔ.

Αν θέλουμε να διαγράψουμε ένα βιβλίο τότε καλούμε τη συνάρτηση DeleteBook, η οποία δέχεται ως όρισμα το βιβλίο που θέλουμε να διαγράψουμε και έπειτα εκτελεί τον κώδικα

```
_connection.DeleteAsync<ABook>(id) με τον οποίο διαγράφεται από τη ΒΔ το βιβλίο.
```

Σε περίπτωση που θέλουμε να διαγράψουμε όλα τα βιβλία καλούμε τη συνάρτηση DeleteAllBooks. Ο πιο σύντομος τρόπος είναι να διαγράψουμε όλο τον πίνακα και έπειτα να τον ξαναδημιουργήσουμε, χρησιμοποιώντας τις εντολές:

```
_connection.DropTableAsync<ABook>();
_connection.CreateTableAsync<ABook>();
```

Αφού πλέον έχουμε τελειώσει με την κλάση διαχείρισης της Βάσης Δεδομένων ακολουθεί το πέμπτο βήμα που αφορά τη φόρτωσή της κατά την εκκίνηση της εφαρμογής, στην κλάση App του κοινού κώδικα:

```
private static BooksDataAccess _database;
public static BooksDataAccess Database => GetDatabase();
```

```
private static BooksDataAccess GetDatabase()
{
    if (_database == null)
    {
        _database = new BooksDataAccess("Database.db3");
    }

    return _database;
}
```

7. Database

Αρχικά, δηλώνουμε ένα ιδιωτικό και ένα δημόσιο μέλος Database τύπου BooksDataAccess, στο οποίο αναθέτουμε την τιμή που θα λάβουμε από τη στατική συνάρτηση GetDatabase.
Η τελευταία αρχικοποιεί την _database δίνοντας ως παράμετρο το όνομα Database.db3.
Έτσι, κάθε φορά που χρειαζόμαστε πρόσβαση στη Βάση Δεδομένων απλά θα καλούμε την Database μέσα από την App με μια κλήση όπως App.Database(...).
Έτσι, όταν για παράδειγμα χρειαστούμε να πάρουμε τη λίστα των βιβλίων, μπορούμε μέσα από τη σελίδα της εφαρμογής να καλέσουμε τον παρακάτω κώδικα:
`var res =await App.Database.GetAllBooks();`
Έπειτα χειριζόμαστε και μορφοποιούμε κατάλληλα τα εισερχόμενα δεδομένα για να τα προβάλλουμε στο χρήστη.

API

Εκτός από τα τοπικά δεδομένα, η εφαρμογή μας μπορεί να αποκτήσει δεδομένα και από διαδικτυακές υπηρεσίες (πχ restful services). Στο project UniBooks θα δούμε πως μπορούμε να αποκτήσουμε δεδομένων βιβλίων από την υπηρεσία της Google Books. Μπορείτε να προβάλλετε τεχνικές λεπτομέρειες του API στη σελίδα:

<https://developers.google.com/books/>

Αρχικά θα σχεδιάσουμε το Interface της υπηρεσίας με όνομα IRestService με τον παρακάτω κώδικα:

```
public interface IRestService
{
    Task<IRestResponse> GetBooksAsync(string q);
}
```

Εδώ δηλώνουμε μια συνάρτηση GetBooksAsync που δέχεται ως όρισμα ένα μέρος του τίτλου του βιβλίου που αναζητούμε (q) και επιστρέφει μια τιμή τύπου IRestResponse.

Για να υλοποιήσουμε το Interface δημιουργούμε μια κλάση με όνομα GoogleBooksService. Στην κλάση μας αρχικά δηλώνουμε τις κλάσεις που περιγράφουν τα δεδομένα που λαμβάνουμε από την υπηρεσία της Google Books.

Έτσι η κλάση Books περιγράφει και περιέχει τη συλλογή των βιβλίων (List<BookModel>). Κάθε βιβλίο περιέχεται σε ένα BookModel. Ένα BookModel περιέχει ένα volumeInfo που περιγράφει το βιβλίο και ένα saleInfo που περιγράφει τις λεπτομέρειες πώλησης του βιβλίου (τιμή κτλ).

Ένα volumeInfo μεταξύ άλλων περιγράφει τα imageLinks που είναι οι εικόνες του βιβλίου.

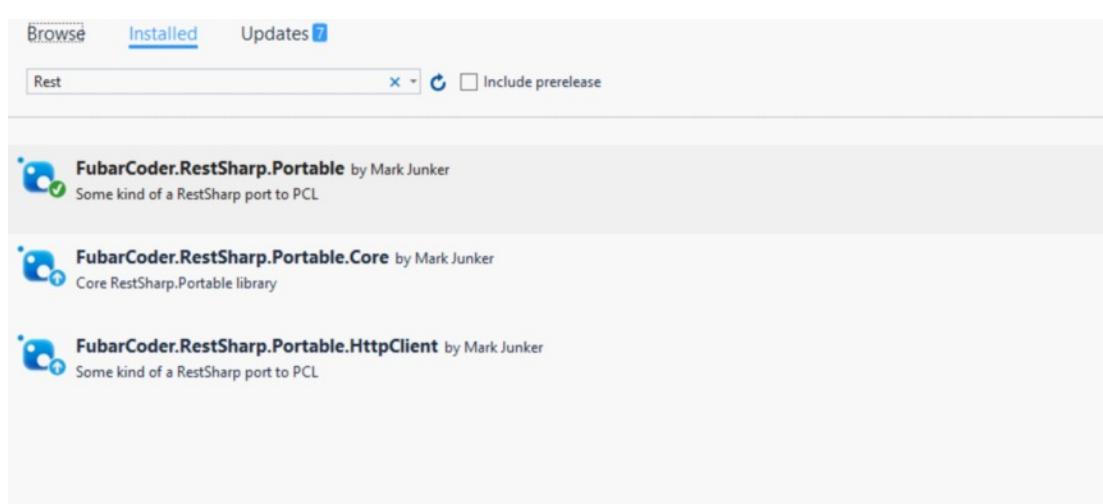
Ένα saleInfo μεταξύ άλλων περιγράφει μια listPrice που περιέχει την τιμή πώλησης του βιβλίου.

Επίσης, στην GoogleBooksService περιέχεται και η κλάση ABook που χρησιμοποιείται για να αποθηκεύσει τα δεδομένα στη Βάση Δεδομένων. Ουσιαστικά μετατρέπουμε την BookModel σε ABook κατά τη λήψη των δεδομένων από το δίκτυο.

Η μέθοδος GetBooksAsync, περιέχει έναν RestClient που μας βοηθά να λάβουμε δεδομένα από μια υπηρεσία.

```
public async Task<IRestResponse> GetBooksAsync(string q)
{
    using (var client = new RestClient(new
Uri("https://www.googleapis.com/books/v1/volumes?q=" + q + "&key=" + key + "&maxResults=40")))
    {
        var request = new RestRequest(Method.GET);
        return await client.Execute(request);
    }
}
```

Για να μπορέσουμε να χρησιμοποιήσουμε τον RestClient πρέπει να τον λάβουμε από την επιλογή “Manage NuGet packages” (δεξί κλικ στο project).



8. Διασύνδεση με API

Ομοίως, τον εγκαθιστούμε και στα υπόλοιπα projects.

Κατά τη δήλωσή του με τη χρήση της using περνάμε ως παράμετρο ένα Uri με τιμή “<https://www.googleapis.com/books/v1/volumes?q=+q+&key=>” + key + “&maxResults=40”. Στη διεύθυνση της Google η παράμετρος url q ορίζει τον τίτλο του βιβλίου ενώ το key ορίζει το κλειδί του developer που πρέπει να λάβετε από τη Google. Η χρήση κλειδιού πρόσβασης είναι απαραίτητη για να μην γίνεται κακόβουλη χρήση των διαδικτυακών υπηρεσιών.

Έπειτα, χρησιμοποιώντας ένα Request τύπου Get κάνουμε Execute και επιστρέφουμε την απάντηση.

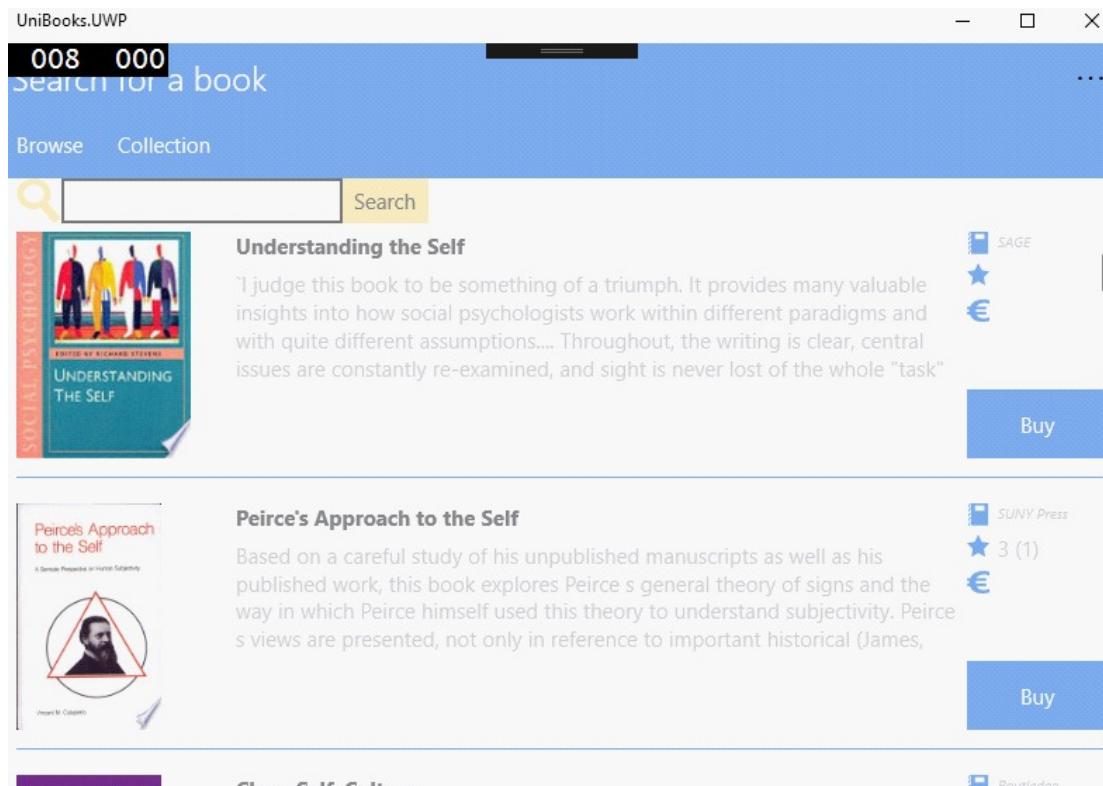
Τελειώσαμε με την κατασκευή της κλάσης πρόσβασης στην υπηρεσία της Google Books. Μένει να βρούμε έναν τρόπο για να την καλέσουμε από παντού. Όμοια με τη διαχείριση της Βάσης Δεδομένων γράφουμε στο αρχείο App.cs:

```
public static GoogleBooksService BooksServiceManager { get; set; }
public App()
{
    InitializeComponent();
    BooksServiceManager = new GoogleBooksService();
    ...
}
```

Δηλαδή, δηλώνουμε τον BooksServiceManager που θα διαχειρίζεται τις συνδέσεις με τη διαδικτυακή υπηρεσία Google Books και απλά τον αρχικοποιούμε στον κατασκευαστή της App.

Όμοια με τη Βάση Δεδομένων, για να καλέσουμε τον BooksServiceManager καλούμε τον κώδικα:

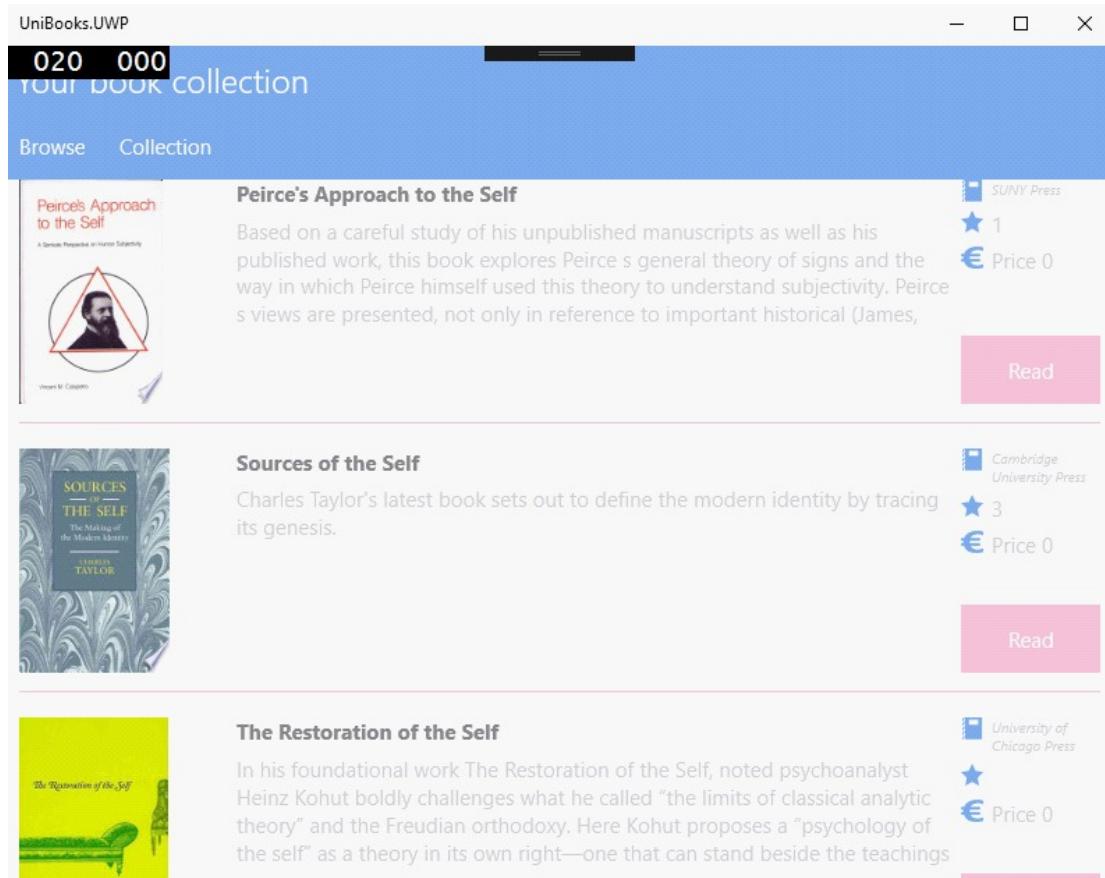
```
var res = await App.BooksServiceManager.GetBooksAsync(q);
```



9. Η εφαρμογή ηλεκτρονικών Βιβλίων UniBooks

UniBooks

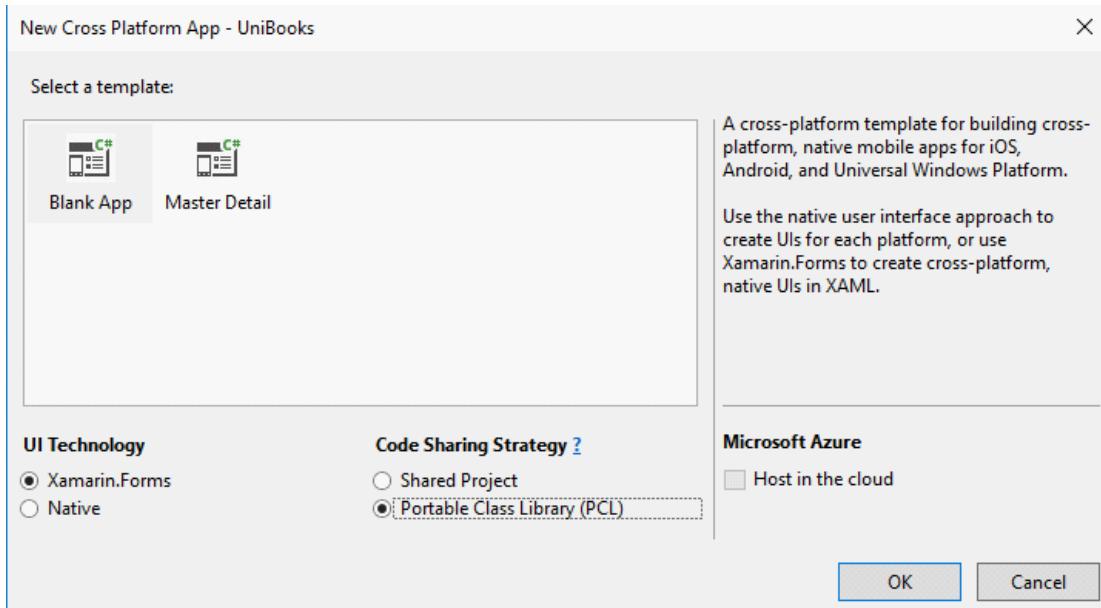
Το UniBooks είναι μία εφαρμογή cross-platform που αναπτύχθηκε με τη βοήθεια του Xamarin.Forms προκειμένου να δείξει τον τρόπο με τον οποίο φτιάχνουμε ένα εικονικό ηλεκτρονικό κατάστημα βιβλίων. Με το που ανοίγει η εφαρμογή, ο χρήστης βλέπει δύο καρτέλες, την καρτέλα Browse και την καρτέλα Collection. Στην καρτέλα Browse προβάλλονται τα βιβλία που λαμβάνουμε από τη διαδικτυακή υπηρεσία της Google Books. Στην καρτέλα Collection προβάλλονται τα βιβλία που έχει αγοράσει (εικονικά) ο χρήστης πατώντας το κουμπί Buy.



Ας ξεκινήσουμε από την αρχή το νέο project για να κατανοήσουμε καλύτερα την εμπειρία δημιουργίας μιας εμπορικής εφαρμογής.

Στο Visual Studio 2017 ξεκινάμε ένα νέο project τύπου Cross Platform App με όνομα UniBooks, επιλέγοντας ως Template “Blank App”, ως UI Technology το “Xamarin.Forms” και ως Code Sharing Strategy το Portable Class Library (PCL).

9. Η εφαρμογή ηλεκτρονικών Βιβλίων UniBooks



Έπειτα, περιμένουμε μέχρι να ολοκληρωθεί η φόρτωση όλων των project (κοινός κώδικας (portable), Android, iOS και UWP (Win10)).

Στα προηγούμενα κεφάλαια είδαμε πως υλοποιείται η πρόσβαση σε τοπικά δεδομένα της Βάσης Δεδομένων SQLite και η πρόσβαση σε εξωτερικά δεδομένα Restful διαδικτυακών υπηρεσιών, όπως αυτό της Google Books.

Αφού ολοκληρώσουμε την υλοποίηση των ανωτέρω δυνατοτήτων πρόσβασης σε εξωτερικά και τοπικά δεδομένα (SQLite και Google Books) συνεχίζουμε με την υλοποίηση του γραφικού περιβάλλοντος και της πλοιήγησης σε αυτό ξεκινώντας από τον κατασκευαστή της App.cs, δηλαδή τον κώδικα που εκτελείται πρώτος κάθε φορά που ξεκινάμε το πρόγραμμα.

```
public App()
{
    InitializeComponent();
    BooksServiceManager = new GoogleBooksService();

    MainPage = new MainPage();
}
```

Βλέπουμε, ότι έπειτα από την αρχικοποίηση της υπηρεσίας Google Books, ορίζεται ως αρχική σελίδα του προγράμματος η MainPage. Ας δούμε τον c# κώδικα του κατασκευαστή της MainPage:

9. Η εφαρμογή ηλεκτρονικών Βιβλίων UniBooks

```
public partial class MainPage : TabbedPage
{
    public MainPage()
    {
        InitializeComponent();
        this.Children.Add(
            new NavigationPage(new BrowsePage())
            {
                Title = "Browse",
                BarBackgroundColor = Color.FromHex("7BABED"),
                BarTextColor = Color.FromHex("FCFCFE")
            });
        this.Children.Add(
            new NavigationPage(new CollectionPage())
            {
                Title = "Collection",
                BarBackgroundColor = Color.FromHex("7BABED"),
                BarTextColor = Color.FromHex("FCFCFE")
            });
        this.BarTextColor = Color.FromHex("FCFCFE");
        Title = "Search for a book";
        this.CurrentPageChanged += CurrentPageHasChanged;
    }
}
```

Η MainPage ορίζεται ως TabbedPage, δηλαδή ως μια σελίδα που περιέχει πολλές σελίδες στις οποίες πλοηγούμαστε με τη χρήση Tabs (καρτέλες). Το ίδιο βλέπουμε και στον xaml κώδικα της MainPage:

```
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:UniBooks"
             x:Class="UniBooks.MainPage"
             Title="Browse books">
</TabbedPage>
```

Αντί για το γνωστό ContentPage πλέον χρησιμοποιούμε την ετικέτα TabbedPage.

Στον κώδικα c# που παραθέσαμε προηγουμένως βλέπουμε ότι στη φόρτωση της σελίδας MainPage προστίθενται ως καρτέλες (Children.Add) δύο NavigationPage, ένα για τη σελίδα BrowsePage και ένα για τη σελίδα CollectionPage. Σε αυτό το σημείο βλέπουμε ότι χρησιμοποιούμε για πρώτη φορά τη δυνατότητα NavigationPage που είδαμε στα προηγούμενα κεφάλαια. Μέσα σε κάθε NavigationPage δηλώνουμε τον τίτλο, το χρώμα του παρασκηνίου της μπάρας που περιέχει τον τίτλο και το χρώμα του κειμένου της μπάρας.

Έπειτα, αρχικοποιούμε το χρώμα κειμένου της μπάρας του MainPage και δηλώνουμε τον αρχικό τίτλο.

Browse

Η σελίδα Browse είναι η σελίδα στην οποία μπορεί ο χρήστης να αναζητήσει βιβλία στο διαδίκτυο και να προβάλλει σε λίστα διάφορα στοιχεία όπως:

- Τιμή
- Εικόνα
- Τίτλος
- Περιγραφή
- Αξιολόγηση/Πλήθος αξιολογήσεων
- Έκδότης

Όταν φορτωθεί το αρχικό γραφικό περιβάλλον της εφαρμογής, εκτελείται μια αρχική αναζήτηση βιβλίων από τη διαδικτυακή υπηρεσία της Google Books, τα αποτελέσματα της οποίας προβάλλονται στη λίστα. Ο χρήστης έχει τη δυνατότητα να γράψει ένα νέο κείμενο τίτλου και να το αναζητήσει στην υπηρεσία. Σε κάθε εγγραφή υπάρχουν τα παραπάνω στοιχεία ενός βιβλίου καθώς και ένα κουμπί “Buy” που πατώντας το ο χρήστης μπορεί να αγοράσει (εικονικά το βιβλίο).

```
public partial class BrowsePage : ContentPage, INotifyPropertyChanged
{
    private ObservableCollection<BookModel> bookList;
    public ObservableCollection<BookModel> BookList
    {
        get { return bookList; }
        set
        {
            if (bookList != value)
            {
                bookList = value;

                OnPropertyChanged("Books");

                PropertyChanged?.Invoke(this,
new PropertyChangedEventArgs("Books"));
            }
        }
    }
    private string title;
    public string ATitle
    {
        get { return title; }
        set
        {
            if (title != value)
            {
                title = value;

                OnPropertyChanged("Title");
            }
        }
    }
}
```

9. Η εφαρμογή ηλεκτρονικών Βιβλίων UniBooks

```
public BrowsePage()
{
    InitializeComponent();
    Title = "Search for a book";
    ATitle = "Search for a book";
    BookList = new ObservableCollection<BookModel>();
    BindingContext = this;
    test.ItemsSource = BookList;
    SearchForBooks("Self");
}

public async void SearchForBooks(string q)
{
    var res = await App.BooksServiceManager.GetBooksAsync(q);
    var d=JsonConvert.DeserializeObject<Books>(res.Content);
    var books = d.items;
    BookList.Clear();
    foreach (var item in d.items)
    {
        BookList.Add(item);
    }
}

public event PropertyChangedEventHandler PropertyChanged;
void OnPropertyChanged([CallerMemberName]string propertyName = "") =>
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

private void SearchBtn_Clicked(object sender, EventArgs e)
{
    SearchForBooks(SearchTxt.Text);
}

private void BuyBtn_Clicked(object sender, EventArgs e)
{
    Button btn = sender as Button;
    string id= btn.CommandParameter.ToString();
    var book = BookList.FirstOrDefault(i => i.id == id);
    if (book != null)
        Navigation.PushModalAsync(new NavigationPage(new BuyPage(book)) {
            BarBackgroundColor = Color.FromHex("7BABED"),
            BarTextColor= Color.FromHex("FCFCFE")
        });
}
private void ToolbarItem_Clicked(object sender, EventArgs e)
{
    DisplayAlert("UniBooks", "Search and Buy your favorite books!", "OK");
}
```

To Browse είναι μια κλάση ContentPage που έχει ως μέλη τα BookList και το ATitle. To BookList είναι μια λίστα ObservableCollection με αντικείμενα BookModel. Οι λίστες ObservableCollection είναι ειδικές για το Binding με τις λίστες. Όταν τροποποιείται ένα ObservableCollection ενημερώνεται αυτόματα η λίστα και αντιστρέφεται. Το ATitle είναι ένα string που συνδέεται (Binding) με τον τίτλο της σελίδας.

1. Παρουσίαση του Xamarin.Forms: Εγκατάσταση, XAML, Pages, Layouts

Βλέπουμε ότι η κλάση υλοποιεί την INotifyPropertyChanged. Με αυτήν την υλοποίηση, όταν αλλάζει μια ιδιότητα μέλους της κλάσης εκτελείται το συμβάν PropertyChanged το οποίο ενημερώνει όλα Bindings που αναφέρονται σε αυτό το μέλος.

Στον κατασκευαστή του Browse αρχικοποιούμε τη λίστα των βιβλίων και θέτουμε ως BindingContext της σελίδας Browse την τρέχουσα κλάση. Έπειτα, ενημερώνουμε με τον κώδικα test.ItemsSource=BookList ότι τα δεδομένα της λίστας θα ληφθούν από την BookList. Μετά εκτελούμε τη συνάρτηση SearchForBooks("Self") που εκτελεί αναζήτηση βιβλίων στην Google Books που περιέχουν τη λέξη Self.

Εξετάζοντας την SearchForBooks βλέπουμε ότι αρχικά εκτελείται η App.BooksServiceManager.GetBooksAsync(q), η οποία όπως είδαμε στο κεφάλαιο των διαδικτυακών υπηρεσιών λαμβάνει τα βιβλία από την Google Books. Τα δεδομένα λαμβάνονται σε μορφή json. Η json είναι ένα πρότυπο που χρησιμοποιείται ευρέως σήμερα στη μεταφορά δεδομένων. Τα δεδομένα json περιέχονται μέσα σε ένα string το οποίο πρέπει να κάνουμε deserialize για να το φορτώσουμε σε μια κλάση. Αφού λάβουμε τη λίστα σε μορφή string / json εκτελούμε τον κώδικα

```
var d=JsonConvert.DeserializeObject<Books>(res.Content);
```

Εδώ δημιουργούμε μια μεταβλητή d τύπου Books κάνοντας deserialize το κείμενο json που λαμβάνουμε. Έπειτα πάρινοντας τα Items (βιβλία) της d και τα αναθέτουμε στη μεταβλητή books. Μετά καθαρίζουμε τη BookList και την ξαναφορτώνουμε με τα νέα δεδομένα.

Όταν ο χρήστης πατάει το κουμπί SearchBtn εκτελείται η SearchForBooks με παράμετρο το κείμενο του Entry SearchTxt.

Σε κάθε εγγραφή υπάρχει ένα κουμπί Buy το οποίο ανοίγει τη σελίδα Buy για να αγοράσει ο χρήστης εικονικά το βιβλίο. Κάθε φορά που πατάμε το BuyBtn ενεργοποιείται το event BuyBtn. Εκεί θέτουμε τον sender ως btn (Button). Ας δούμε τον κώδικα xaml του btn:

```
<Button x:Name="BuyBtn" Grid.Column="2" Grid.Row="2" Text="Buy" BackgroundColor="#7BABED"
TextColor="#FCFCFE" Clicked="BuyBtn_Clicked"
CommandParameter ="{Binding id}"></Button>
```

Από τον κώδικα xaml βλέπουμε ότι συνδέουμε την ιδιότητα CommandParameter με την τιμή id της Book. Αυτό γίνεται για να μπορέσουμε να ζεχωρίσουμε το βιβλίο που πρόκειται να αγοράσουμε κάθε φορά που πατάμε το κουμπί Buy.

Με τον κώδικα string id= btn.CommandParameter.ToString() θέτουμε στο string id την τιμή της CommandParameter του κουμπιού btn, δηλαδή το id του βιβλίου.

Έπειτα, αναζητούμε το βιβλίο στη BookList και το φορτώνουμε στη μεταβλητή book. Μετά μεταβαίνουμε σε μια νέα NavigationPage με παράμετρο BuyPage, η οποία με τη σειρά της λαμβάνει την παράμετρο book με τη χρήση της εντολής Navigation.PushModalAsync. Με αυτή βάζουμε στη στοίβα Navigation μια νέα σελίδα προβάλλοντάς την όμως ως Modal. Αυτό σημαίνει ότι θα καλύπτει όλο το παράθυρο (αποκρύπτοντας Title, Bar κτλ όπως θα γινόταν με την εντολή Navigation.PushAsync). Μέσα στη NavigationPage θέτουμε το χρώμα του φόντου της Bar και του κειμένου.

9. Η εφαρμογή ηλεκτρονικών Βιβλίων UniBooks

Στο πάνω μέρος του παραθύρου βλέπουμε ότι μπορούμε να πατήσουμε τις τρεις τελίτσες που μας δείχνουν ένα νέο κουμπί με όνομα About. Το κουμπί ενεργοποιεί το συμβάν Clicked που το χειρίζεται η συνάρτηση ToolbarItem_Clicked(object sender, EventArgs e).

Όταν εκτελεστεί θα εκτελεστεί η DisplayAlert("UniBooks", "Search and Buy your favorite books!", "OK") με την οποία θα προβληθεί το σχετικό pop up μήνυμα.

Παρακάτω προβάλλεται ο κώδικας xaml της σελίδας Browse:

```
<ContentPage.ToolbarItems>
<ToolbarItem Text="About" Order="Primary" Priority="0" Clicked="ToolbarItem_Clicked" />
</ContentPage.ToolbarItems>
<ContentPage.Content>
<StackLayout>
<StackLayout Orientation="Horizontal" Spacing="0">
<Image Source="Assets\Search.png" />
<Entry x:Name="SearchTxt" Margin="0,0,0,0" WidthRequest="200"></Entry>
<Button x:Name="SearchBtn" Margin="0,0,0,0" Clicked="SearchBtn_Clicked" Text="Search"
BackgroundColor="#FCEBBF" TextColor="#86878B" ></Button>
</StackLayout>
<ListView x:Name="test" HasUnevenRows="True" SeparatorVisibility="Default" SeparatorColor="Blue" >
<ListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<Grid HeightRequest="175" WidthRequest="150" Margin="0,0,0,18" >
<Grid.ColumnDefinitions>
<ColumnDefinition Width="150"></ColumnDefinition>
<ColumnDefinition Width="*></ColumnDefinition>
<ColumnDefinition Width="100"></ColumnDefinition>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="50"></RowDefinition>
<RowDefinition Height="50"></RowDefinition>
<RowDefinition Height="49"></RowDefinition>
<RowDefinition Height="1"></RowDefinition>
</Grid.RowDefinitions>
<Image Grid.Column="0" Grid.Row="0" Grid.RowSpan="3" Source="{Binding
volumeInfo.imageLinks.smallThumbnail}" Aspect="AspectFit"></Image>
<StackLayout Grid.Column="1" Grid.Row="0" Grid.RowSpan="3" >
<Label Text="{Binding volumeInfo.title}" TextColor="#86878B" FontAttributes="Bold" />
<Label Text="{Binding volumeInfo.description}" TextColor="#CFD0D5" HeightRequest="80" />
</StackLayout>
<StackLayout Grid.Column="2" Grid.Row="0" Grid.RowSpan="2" >
<StackLayout Orientation="Horizontal">
<Image>
<Image.Source>
<OnPlatform x:TypeArguments="ImageSource">
<OnPlatform.iOS>publ.png</OnPlatform.iOS>
<OnPlatform.Android>publ.png</OnPlatform.Android>
<OnPlatform.WinPhone>Assets/publ.png</OnPlatform.WinPhone>
</OnPlatform>
</Image.Source>
</Image>
<Label Text="{Binding volumeInfo.publisher}" TextColor="#CFD0D5" FontSize="10"
FontAttributes="Italic" />
</StackLayout>
<StackLayout Orientation="Horizontal">
<Image>
<Image.Source>
<OnPlatform x:TypeArguments="ImageSource">
<OnPlatform.iOS>rating.png</OnPlatform.iOS>
<OnPlatform.Android>rating.png</OnPlatform.Android>
<OnPlatform.WinPhone>Assets/rating.png</OnPlatform.WinPhone>
</OnPlatform>
</Image.Source>
```

9. Η εφαρμογή ηλεκτρονικών Βιβλίων UniBooks

```
</Image>
<Label Text="{Binding volumeInfo.Ratings}" TextColor="#CFD0D5" />
</StackLayout>
<StackLayout Orientation="Horizontal">
<Image>
<Image.Source>
<OnPlatform x:TypeArguments="ImageSource">
<OnPlatform.iOS>euro.png</OnPlatform.iOS>
<OnPlatform.Android>euro.png</OnPlatform.Android>
<OnPlatform.WinPhone>Assets/euro.png</OnPlatform.WinPhone>
</OnPlatform>
</Image.Source>
</Image>
<Label Text="{Binding saleInfo.listPrice.amount, StringFormat='Price {0}'}" TextColor="#CFD0D5" />
</StackLayout>
</StackLayout>
<Button x:Name="BuyBtn" Grid.Column="2" Grid.Row="2" Text="Buy" BackgroundColor="#7BABED" TextColor="#FCFCFE" Clicked="BuyBtn_Clicked" CommandParameter ="{Binding id}"></Button>
<BoxView Grid.Row="4" Grid.Column="0" Grid.ColumnSpan="3" HeightRequest="1" Color="#7BABED" />
</Grid>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</StackLayout>
</ContentPage.Content>
```

Buy

Όταν ο χρήστης πατάει το κουμπί Buy σε ένα συγκεκριμένο βιβλίο ανοίγει η σελίδα BuyPage.



9. Η εφαρμογή ηλεκτρονικών Βιβλίων UniBooks

Στη σελίδα Buy βλέπουμε τον τίτλο της μπάρας, τις πληροφορίες του επιλεγμένου βιβλίου καθώς και δύο κουμπιά: το κουμπί αγοράς Buy και το κουμπί επιστροφής Back.

Ο κώδικας xaml της σελίδας είναι:

```
<ContentPage.Content>
    <StackLayout>
        <Label Text="Book Information" />
        <Grid HeightRequest="175" WidthRequest="150" Margin="0,0,0,18" >
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="150"/>
                <ColumnDefinition Width="*"/>
                <ColumnDefinition Width="100"/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="50"/>
                <RowDefinition Height="50"/>
                <RowDefinition Height="49"/>
                <RowDefinition Height="1"/>
            </Grid.RowDefinitions>
            <Image Grid.Column="0" Grid.Row="0" Grid.RowSpan="3" Source="{Binding Book.volumeInfo.imageLinks.smallThumbnail}" Aspect="AspectFit"/>
            <StackLayout Grid.Column="1" Grid.Row="0" Grid.RowSpan="3" >
                <Label Text="{Binding Book.volumeInfo.title}" TextColor="#86878B" FontAttributes="Bold" />
                <Label Text="{Binding Book.volumeInfo.description}" TextColor="#CFD0D5" HeightRequest="80" />
            </StackLayout>
            <StackLayout Grid.Column="2" Grid.Row="0" Grid.RowSpan="2" >
                <StackLayout Orientation="Horizontal">
                    <Image>
                        <Image.Source>
                            <OnPlatform x:TypeArguments="ImageSource">
                                <OnPlatform.iOS>publ.png</OnPlatform.iOS>
                                <OnPlatform.Android>publ.png</OnPlatform.Android>
                                <OnPlatform.WinPhone>Assets/publ.png</OnPlatform.WinPhone>
                            </OnPlatform>
                        </Image.Source>
                    </Image>
                    <Label Text="{Binding Book.volumeInfo.publisher}" TextColor="#CFD0D5" FontSize="10" FontAttributes="Italic" />
                </StackLayout>
                <StackLayout Orientation="Horizontal">
                    <Image>
                        <Image.Source>
                            <OnPlatform x:TypeArguments="ImageSource">
                                <OnPlatform.iOS>rating.png</OnPlatform.iOS>
                                <OnPlatform.Android>rating.png</OnPlatform.Android>
                                <OnPlatform.WinPhone>Assets/rating.png</OnPlatform.WinPhone>
                            </OnPlatform>
                        </Image.Source>
                    </Image>
                    <Label Text="{Binding Book.volumeInfo.Ratings}" TextColor="#CFD0D5" />
                </StackLayout>
            <StackLayout Orientation="Horizontal">
                <Image>
                    <Image.Source>
                        <OnPlatform x:TypeArguments="ImageSource">
                            <OnPlatform.iOS>euro.png</OnPlatform.iOS>
                            <OnPlatform.Android>euro.png</OnPlatform.Android>
                            <OnPlatform.WinPhone>Assets/euro.png</OnPlatform.WinPhone>
                        </OnPlatform>
                    </Image.Source>
                </Image>
                <Label Text="{Binding Book.saleInfo.listPrice.amount, StringFormat='Price {0}'}" TextColor="#CFD0D5" />
            </StackLayout>
        </StackLayout>
        <BoxView Grid.Row="4" Grid.Column="0" Grid.ColumnSpan="3" HeightRequest="1" Color="#7BABED" />
```

9. Η εφαρμογή ηλεκτρονικών Βιβλίων UniBooks

```
<OnPlatform.Android>euro.png</OnPlatform.Android>
    <OnPlatform.WinPhone>Assets/euro.png</OnPlatform.WinPhone>
        </Image>
        <Label Text="{Binding Book.saleInfo.listPrice.amount, StringFormat='Price
{0}'}" TextColor="#CFD0D5" />
    </StackLayout>
</StackLayout>
<BoxView Grid.Row="4" Grid.Column="0" Grid.ColumnSpan="3" HeightRequest="1"
Color="#7BABED" />
</Grid>
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*"/></ColumnDefinition>
        <ColumnDefinition Width="*"/></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <Button x:Name="BuyBtn" Grid.Column="0" Text="Buy" BackgroundColor="#7BABED"
Clicked="BuyBtn_Clicked"></Button>
    <Button x:Name="Back" Grid.Column="1" Text="Back" BackgroundColor="#F6C2D8"
Clicked="Back_Clicked"></Button>
</Grid>
</StackLayout>
</ContentPage.Content>
```

Μέσα σε ένα Grid, όμοιο με το Grid που παρουσιάσαμε στη σελίδα Browse, παρουσιάζονται τα στοιχεία του βιβλίου. Βασική διαφορά είναι το γεγονός ότι θέτουμε μπροστά από τα δεδομένα που γίνονται Binding στη λίστα το "Book." εννοώντας το επιλεγμένο στιγμιότυπο.

Κατά τη φόρτωση της σελίδας Buy λαμβάνεται μέσα από τον κατασκευαστή το επιλεγμένο βιβλίο και ανατίθεται στη μεταβλητή Book.

```
public BuyPage(BookModel book)
{
    InitializeComponent();
    Title = "Book payment";
    Book = book;

    this.BindingContext = this;
}
```

Το κουμπί BuyBtn κατά το συμβάν Clicked τρέχει τον παρακάτω c# κώδικα:

```
private void BuyBtn_Clicked(object sender, EventArgs e)
{
    ABook a = new ABook();
    foreach (var i in Book.volumeInfo.authors)
        a.authors += i + " - ";
    a.averageRating = Book.volumeInfo.averageRating;
    a.description = Book.volumeInfo.description;
    a.Id = Book.id;
    a.listPrice = Book.saleInfo.listPrice!=null?Book.saleInfo.listPrice.amount:0;
    a.pageCount = Book.volumeInfo.pageCount;
    a.publishedDate = Book.volumeInfo.publishedDate;
    a.publisher = Book.volumeInfo.publisher;
    a.ratingsCount = Book.volumeInfo.ratingsCount;
    a.saleability = Book.saleInfo.saleability;
    a.smallThumbnail = Book.volumeInfo.imageLinks.smallThumbnail;
    a.thumbnail = Book.volumeInfo.imageLinks.thumbnail;
    a.subtitle = Book.volumeInfo.subtitle;
    a.title = Book.volumeInfo.title;

    App.Database.SaveBook(a);
    DisplayAlert("Congratulations", "You have successfully bought the book "+ a.title, "OK");

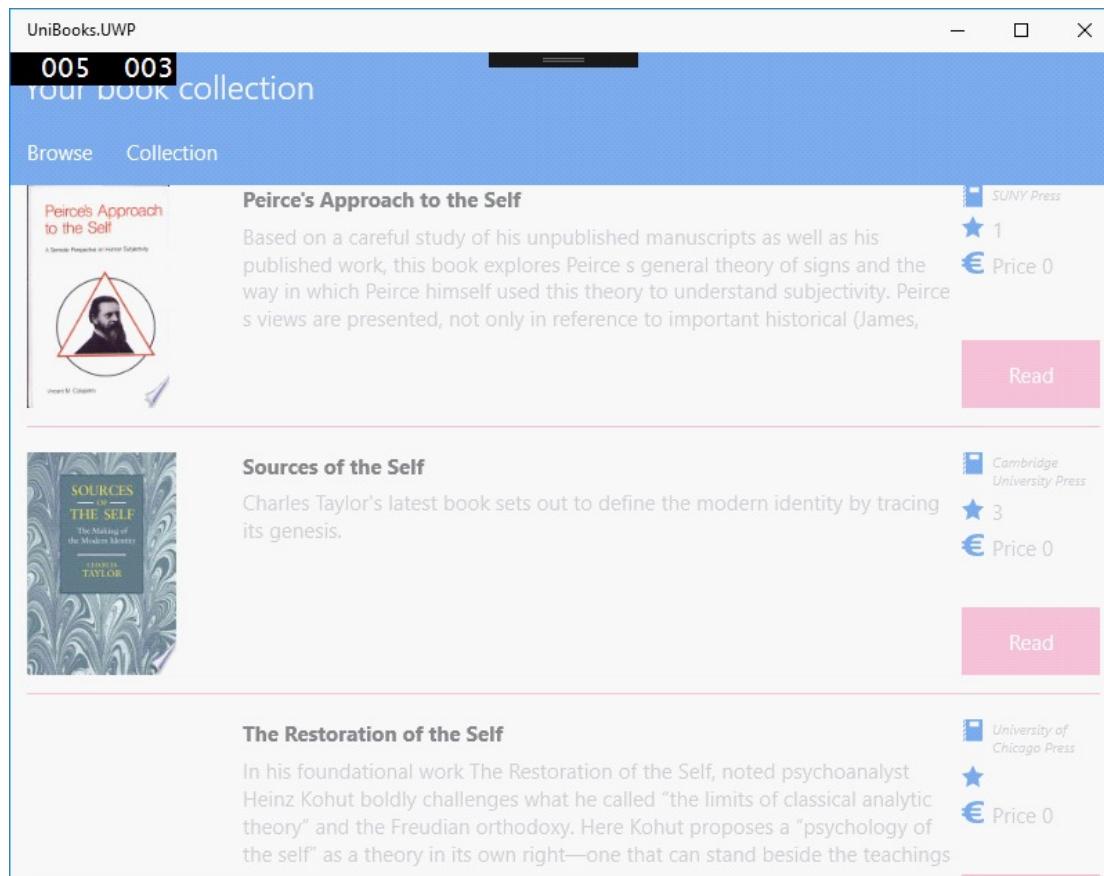
    Navigation.PopModalAsync();
}
```

9. Η εφαρμογή ηλεκτρονικών Βιβλίων UniBooks

Ο ανωτέρω κώδικας δημιουργεί ένα νέο βιβλίο a, τύπου ABook, δηλαδή ο κατάλληλος τύπος για την φόρτωση βιβλίων στη Βάση Δεδομένων. Αφού αντιγραφούν οι τιμές στο βιβλίο a γίνεται αποθήκευση του βιβλίου στη B.D. με την εντολή App.Database.SaveBook(a). Τέλος, προβάλλεται ένα μήνυμα επιτυχούς αποθήκευσης στο χρήστη και με την εντολή Navigation.PopModalAsync αφαιρείται από τη στοίβα η τελευταία Modal σελίδα, δηλαδή η σελίδα Buy. Το ίδιο συμβαίνει και με το χειρισμό του συμβάντος κλικ από το κουμπί Back.

Collection

Η σελίδα Collection προβάλλει στο χρήστη τα αγορασμένα και αποθηκευμένα στην εφαρμογή βιβλία. Ομοίως, όπως η Browse, η σελίδα περιέχει μία λίστα με τα αποθηκευμένα βιβλία με τη διαφορά ότι αντί για κουμπί Buy υπάρχει σε κάθε εγγραφή ένα κουμπί Read για το άνοιγμα του επιλεγμένου βιβλίου προς ανάγνωση.



1. Παρουσίαση του Xamarin.Forms: Εγκατάσταση, XAML, Pages, Layouts

Παρακάτω παρουσιάζεται ο κώδικας xaml της σελίδας:

```
<ContentPage.Content>
    <StackLayout>
        <ListView x:Name="test" HasUnevenRows="True" SeparatorVisibility="Default"
SeparatorColor="Blue">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <Grid HeightRequest="175" WidthRequest="150" Margin="0,0,0,18" >
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition Width="150"/></ColumnDefinition>
                                <ColumnDefinition Width="*"/></ColumnDefinition>
                                <ColumnDefinition Width="100"/></ColumnDefinition>
                            </Grid.ColumnDefinitions>
                            <Grid.RowDefinitions>
                                <RowDefinition Height="50"/></RowDefinition>
                                <RowDefinition Height="50"/></RowDefinition>
                                <RowDefinition Height="49"/></RowDefinition>
                                <RowDefinition Height="1"/></RowDefinition>
                            </Grid.RowDefinitions>
                            <Image Grid.Column="0" Grid.Row="0" Grid.RowSpan="3"
Source="{Binding smallThumbnail}" Aspect="AspectFit"/>
                            <StackLayout Grid.Column="1" Grid.Row="0" Grid.RowSpan="3" >
                                <Label Text="{Binding title}" TextColor="#86878B"
FontAttributes="Bold" />
                                <Label Text="{Binding description}" TextColor="#CFD0D5"
HeightRequest="80" />
                            </StackLayout>
                            <StackLayout Grid.Column="2" Grid.Row="0" Grid.RowSpan="2" >
                                <StackLayout Orientation="Horizontal">
                                    <Image>
                                        <Image.Source>
                                            <OnPlatform x:TypeArguments="ImageSource">
                                                <OnPlatform.iOS>publ.png</OnPlatform.iOS>
                                                <OnPlatform.Android>publ.png</OnPlatform.Android>
                                                <OnPlatform.WinPhone>Assets/publ.png</OnPlatform.WinPhone>
                                                </OnPlatform>
                                            </Image.Source>
                                        </Image>
                                        <Label Text="{Binding publisher}" TextColor="#CFD0D5"
FontSize="10" FontAttributes="Italic" />
                                    </StackLayout>
                                    <StackLayout Orientation="Horizontal">
                                        <Image>
                                            <Image.Source>
                                                <OnPlatform x:TypeArguments="ImageSource">
                                                    <OnPlatform.iOS>rating.png</OnPlatform.iOS>
                                                    <OnPlatform.Android>rating.png</OnPlatform.Android>
                                                    <OnPlatform.WinPhone>Assets/rating.png</OnPlatform.WinPhone>
                                                    </OnPlatform>
                                                </Image.Source>
                                            </Image>
                                            <Label Text="{Binding ratingsCount}" TextColor="#CFD0D5"
/>
                                        </StackLayout>
                                        <StackLayout Orientation="Horizontal">
                                            <Image>
                                                <Image.Source>
                                                    <OnPlatform
```

9. Η εφαρμογή ηλεκτρονικών Βιβλίων UniBooks

```
x:TypeArguments="ImageSource">
    <OnPlatform.iOS>euro.png</OnPlatform.iOS>
<OnPlatform.Android>euro.png</OnPlatform.Android>
<OnPlatform.WinPhone>Assets/euro.png</OnPlatform.WinPhone>
    </OnPlatform>
    </Image.Source>
    </Image>
    <Label Text="{Binding listPrice, StringFormat='Price
{0}'}" TextColor="#CFD0D5" />
        </StackLayout>
    </StackLayout>
    <Button x:Name="ReadBtn" Grid.Column="2" Grid.Row="2" Text="Read"
BackgroundColor="#F6C2D8" TextColor="#FCFCFE" Clicked="ReadBtn_Clicked"
CommandParameter="{Binding Id}"></Button>
    <BoxView Grid.Row="4" Grid.Column="0" Grid.ColumnSpan="3"
HeightRequest="1" Color="#F6C2D8" />
        </Grid>
    </ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</StackLayout>
</ContentPage.Content>
```

Στον κώδικα παρατηρούμε ότι υπάρχει μια λίστα που περιέχει τα αποθηκευμένα βιβλία στη Βάση Δεδομένων SQLite και κάθε βιβλίο μορφοποιείται με τη χρήση ενός Grid. Στο κάτω δεξιό μέρος του κάθε βιβλίου υπάρχει ένα κουμπί ReadBtn το οποίο για το χειρισμό του συμβάντος Clicked καλεί τη c# συνάρτηση ReadBtn_Clicked περνώντας ως όρισμα στην CommandParameter την τιμή Id του βιβλίου. Ας δούμε αρχικά τον κατασκευαστή της Collection

```
public CollectionPage()
{
    InitializeComponent();
    Title = "Your book collection";
    BookList = new ObservableCollection<ABook>();
    BindingContext = this;
    test.ItemsSource = BookList;
}
```

Η λίστα τύπου ObservableCollection<ABook> BookList αρχικοποιείται και συνδέεται με το ItemsSource της λίστας test.

Παρατηρούμε μέσα στον κώδικα το παρακάτω:

```
protected override void OnAppearing()
{
    base.OnAppearing();
    SearchForBooks();
}
```

9. Η εφαρμογή ηλεκτρονικών Βιβλίων UniBooks

To OnAppearing εκτελείται κάθε φορά που προβάλλεται η σελίδα. Εκεί καλούμε και τη συνάρτηση SearchForBooks η οποία φορτώνει τα αποθηκευμένα βιβλία της Βάσης Δεδομένων. Έτσι έχουμε τον κώδικα:

```
public async void SearchForBooks()
{
    var res =await App.Database.GetAllBooks();
    BookList.Clear();
    foreach (var item in res)
    {
        BookList.Add(item);
    }
}
```

Η App.Database.GetAllBooks φορτώνει τα βιβλία της Βάσης Δεδομένων SQLite στην res. Έπειτα καθαρίζεται η BookList και φορτώνεται με τα λαμβανόμενα βιβλία.

Σε κάθε βιβλίο υπάρχει το κουμπί ReadBtn, το οποίο όταν πατηθεί ανοίγει η σελίδα ReadPage με παράμετρο το επιλεγμένο βιβλίο ABook. Ο κώδικας χειρισμού του συμβάντος είναι ο εξής:

```
private async void ReadBtn_Clicked(object sender, EventArgs e)
{
    Button btn = sender as Button;
    string id = btn.CommandParameter.ToString();
    ABook a=await App.Database.GetBook(id);

    if (a != null)
        await Navigation.PushModalAsync(new NavigationPage(new ReadPage(a))
    {
        BarBackgroundColor = Color.FromHex("7BABED"),
        BarTextColor= Color.FromHex("FCFCFE"),
        Title = "Book Reader"
    });
}
```

Αυτό σημαίνει ότι πάρε τον sender ως Button και βάλτον στη συνάρτηση btn. Μετά πάρε από το CommandParameter το id του βιβλίου και φέρτο από τη Βάση Δεδομένων με την εντολή App.Database.GetBook(id). Αφού φέρεις το βιβλίο άνοιξε τη σελίδα ReadPage με παράμετρο το βιβλίο a, μέσα από μια νέα NavigationPage που θα βάλεις στη λίστα με την εντολή Navigation.PushModalAsync. Τέλος, θέσε τον τίτλο, το χρήμα φόντου και το χρώμα κειμένου το νέου NavigationPage.

Read

Η υλοποίηση της σελίδας ανάγνωσης ενός βιβλίου στο παράδειγμά μας είναι πολύ απλή αφού το μόνο που έχουμε να κάνουμε είναι να προβάλλουμε τις πληροφορίες που έχουμε αποθηκεύσει στη Βάση Δεδομένων.

9. Η εφαρμογή ηλεκτρονικών Βιβλίων UniBooks

Ο κώδικας xaml της σελίδας ReadPage είναι ο εξής:

```
<ContentPage.Content>
    <StackLayout>
        <Label Text="Read your selected book" />
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="150"/>
                <ColumnDefinition Width="*"/>
                <ColumnDefinition Width="100"/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="50"/>
                <RowDefinition Height="50"/>
                <RowDefinition Height="49"/>
                <RowDefinition Height="1"/>
            </Grid.RowDefinitions>
            <Image Grid.Column="0" Grid.Row="0" Grid.RowSpan="3" Source="{Binding Book.smallThumbnail}" Aspect="AspectFit"></Image>
            <StackLayout Grid.Column="1" Grid.Row="0" Grid.RowSpan="3" >
                <Label Text="{Binding Book.title}" TextColor="#868788" FontAttributes="Bold" />
            </StackLayout>
            <StackLayout Grid.Column="2" Grid.Row="0" Grid.RowSpan="2" >
                <StackLayout Orientation="Horizontal">
                    <Image>
                        <Image.Source>
                            <OnPlatform x:TypeArguments="ImageSource">
                                <OnPlatform.iOS>publ.png</OnPlatform.iOS>
                                <OnPlatform.Android>publ.png</OnPlatform.Android>
                                <OnPlatform.WinPhone>Assets/publ.png</OnPlatform.WinPhone>
                            </OnPlatform>
                        </Image.Source>
                    </Image>
                    <Label Text="{Binding Book.publisher}" TextColor="#CFD0D5" FontSize="10" FontAttributes="Italic" />
                </StackLayout>
                <StackLayout Orientation="Horizontal">
                    <Image>
                        <Image.Source>
                            <OnPlatform x:TypeArguments="ImageSource">
                                <OnPlatform.iOS>rating.png</OnPlatform.iOS>
                                <OnPlatform.Android>rating.png</OnPlatform.Android>
                                <OnPlatform.WinPhone>Assets/rating.png</OnPlatform.WinPhone>
                            </OnPlatform>
                        </Image.Source>
                    </Image>
                    <Label Text="{Binding Book.Ratings}" TextColor="#CFD0D5" />
                </StackLayout>
                <StackLayout Orientation="Horizontal">
                    <Image>
                        <Image.Source>
                            <OnPlatform x:TypeArguments="ImageSource">
                                <OnPlatform.iOS>euro.png</OnPlatform.iOS>
                                <OnPlatform.Android>euro.png</OnPlatform.Android>
                                <OnPlatform.WinPhone>Assets/euro.png</OnPlatform.WinPhone>
                            </OnPlatform>
                        </Image.Source>
                    </Image>
                </StackLayout>
            </StackLayout>
        </Grid>
    </StackLayout>

```

9. Η εφαρμογή ηλεκτρονικών Βιβλίων UniBooks

```
</Image>
<Label Text="{Binding Book.listPrice, StringFormat='Price {0}'}"
TextColor="#CFD0D5" />
</StackLayout>
</StackLayout>
<BoxView Grid.Row="4" Grid.Column="0" Grid.ColumnSpan="3" HeightRequest="1"
Color="#7BABED" />
</Grid>
<StackLayout>
<ScrollView>
<Content View>
<Label Text="{Binding Book.description}"></Label>
</Content View>
</ScrollView>
<Button x:Name="Back" Text="Back" BackgroundColor="#F6C2D8"
Clicked="Back_Clicked"></Button>
</StackLayout>
</StackLayout>
</ContentPage.Content>
```

Προβάλλουμε τα στοιχεία του βιβλίου μέσα σε ένα Grid, βάζοντας αριστερά την εικόνα του βιβλίου, στο κέντρο τον τίτλο και δεξιά τον εκδοτικό οίκο, την τιμή και την αξιολόγηση. Κάτω ακριβώς τοποθετούμε το κείμενο της περιγραφής του βιβλίου, εξομοιώνοντας το πώς ήταν αν είχαμε το πλήρες περιεχόμενο του βιβλίου. Αν και πολύ πρόχειρο μας δίνει μια ιδέα με το σχετικό layout που θα χρησιμοποιούσαμε.

To Binding των δεδομένων γίνεται με τη δημόσια μεταβλητή Book τύπου ABook.

Ας δούμε πιο αναλυτικά τον κατασκευαστή της σελίδας:

```
public ReadPage(ABook book)
{
    InitializeComponent();
    Title = "Book Reader";
    Book = book;
    this.BindingContext = this;
}
```

Αφού ενημερώσουμε τον τίτλο, φορτώνουμε την book που λάβαμε από την παράμετρο στην Book και θέτουμε ως BindingContext όλης της σελίδας την τρέχουσα κλάση.

Στο κάτω μέρος της σελίδας υπάρχει ένα κουμπί Back με το οποίο επιστρέφουμε στην προηγούμενη σελίδα με τη χρήση του Navigation.PopModalAsync().

Animation	Κίνηση γραφικών στοιχείων
API	Διαδικτυακές (συνήθως) υπηρεσίες που καλούμε για να εκτελέσουν συγκεκριμένη εργασία
Business logic	Λογική υπολογισμών που συνήθως λαμβάνει δεδομένα από το γραφικό περιβάλλον και αποθηκεύει στο επίπεδο πρόσβασης δεδομένων data access
Code sharing strategy	Στρατηγική διαμοιρασμού κώδικα
Controls	Στοιχεία ελέγχου, όπως κουμπιά, ετικέτες κτλ
Cross-Platform	Γράφουμε έναν κώδικα και τρέχει σε πολλά συστήματα, όπως Android, iOS κτλ
Data access	Επίπεδο πρόσβασης δεδομένων. Μόνο αυτό επικοινωνεί απευθείας με τη Βάση Δεδομένων
IDE	Περιβάλλον ανάπτυξης λογισμικού
Layouts	Στοιχεία που επιτρέπουν την οργάνωση των αντικειμένων που περιέχουν με σκοπό την καταννόηση από το χρήστη
Namespace	Ονοματοχώρος στον οποίο μια κλάση έχει ορατότητα
Navigation	Πλοήγηση ανάμεσα στις σελίδες της εφαρμογής
Sensors	Αισθητήρες ενός κινητού όπως GPS κτλ
Spacing	Απόσταση μεταξύ στοιχείων
User interface	Γραφικό περιβάλλον

Εισαγωγή στη Xamarin.Forms	https://www.xamarin.com/forms
Οδηγός της Xamarin-Έναρξη	https://developer.xamarin.com/guides/cross-platform/getting_started/
Ο πρώτος κώδικας στη Xamarin.Forms	https://developer.xamarin.com/guides/xamarin-forms/getting-started/introduction-to-xamarin-forms/
Ο κώδικας XAML	https://developer.xamarin.com/guides/xamarin-forms/xaml/
Xamarin.Forms Pages	https://developer.xamarin.com/guides/xamarin-forms/user-interface/controls/pages/
Xamarin.Forms Layouts	https://developer.xamarin.com/guides/xamarin-forms/user-interface/layouts/

LAYOUTS

AbsoluteLayout

-AbsoluteLayout.LayoutBounds (in controls)
[X,Y,Width,Height]
-AbsoluteLayout.LayoutFlags (in controls) [All,
PositionProportional]

RelativeLayout

-RelativeLayout.Xconstraint /Yconstraint:
 "{ConstraintExpression Type=Constant,
 Constant=10}"
 "{ConstraintExpression Type=RelativeToParent,
 Property=Width,
 Factor=0.5}"
 "{ConstraintExpression Type=RelativeToView,
 Property=Y,
 ElementName=Red,
 Constant=-5}"

Frame

-HasShadow: [True/False]
-OutlineColor: [Silver]

Grid Layout

```
<Grid.RowDefinitions>
  <RowDefinition Height="*" />
  .....
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="*" />
  .....
</Grid.ColumnDefinitions>
<Label Text="Top Right" Grid.Row="0" Grid.Column="1" />
```

CONTROLS

Label

-LineBreakMode: Change line [WordWrap]
-Text

BoxView

-Color [Red,Olive..]
-BackgroundColor [Red,..]