Xamarin Hands On

CHI SONO?

Chi sono?

- Alessandro Del Sole, romano da Cremona ☺
- Microsoft Certified Professional e Programming Specialist: C#
- Microsoft MVP – Visual Studio & Development Technologies
- Sviluppatore senior .NET e consulente in iTuna Srl
- Autore di articoli e libri tecnici (MSDN Magazine, VB 2015 Unleashed, Roslyn Succintly, VS 2015 Succintly e altri 11 ☺)
- Windows Presentation Foundation, Xamarin, Windows (Phone) 8 / 8.1 / 10, servizi web (WCF, Web Api)
- Poco, pochissimo web ☺

- @progalex

Di cosa parleremo?

- Overview dello sviluppo mobile Cross-Platform
- Xamarin, la soluzione cross-platform moderna
- Intro a Xamarin.Android
- Intro a Xamarin.iOS
- Intro a Xamarin.Forms: cross-platform in C# e Visual Studio
- Stategie di condivisione del codice
- Consumare servizi

XAMARIN, LA SOLUZIONE CROSS-PLATFORM MODERNA

Xamarin, la soluzione cross-platform moderna

- Problemi:
  - iOS: Xcode, Objective C/Swift
  - Android: Android Studio/Java
  - Windows: Visual Studio, C#
- Xamarin vuole risolvere i problemi precedenti
- Offre un prodotto che consente di sviluppare app **native** per più piattaforme mediante:
  - Unico linguaggio: C#
  - Unico ambiente di sviluppo: Xamarin Studio (Win/Mac), Microsoft Visual Studio, Visual Studio for Mac
  - Set di librerie e API che «wrappano» il nativo
  - Favorendo il riutilizzo del codice

Xamarin, la soluzione cross-platform moderna

- Xamarin Platform:
  - Tool di sviluppo
  - IDE (Xamarin Studio)
  - Xamarin University
  - Xamarin Insight (presto inglobata in VS Mobile Center)
  - Xamarin TestCloud

Xamarin, la soluzione cross-platform moderna

- Perché C#?
- Linguaggio potente, fortemente tipizzato, object oriented, avanzato, open source, diffuso ma soprattutto basato su .NET Framework
- Microsoft .NET: tecnologia estremamente potente per lo sviluppo di ogni tipo di app (desktop, Web, Cloud, mobile)
  - File I/O, collections, network, async, threading, GC, security, data, XML, Json, ecc.
  - LINQ, lambda, eventi, ecc.
- Talmente potente che Xamarin ha inventato un porting .NET chiamato Mono per Mac (MonoMac) e Android (MonoDroid) (C# su Apple e Android)

Xamarin, la soluzione cross-platform moderna

- Xamarin sviluppa librerie .NET (Xamarin Platform):
  - Xamarin.Mac
  - Xamarin.iOS
  - Xamarin.Android
- Sono versioni .NET delle API native di Mac, iOS, Android
- Posso usarle scrivendo C# per accedere alle API native dei vari device più a .NET Framework
- Con Xamarin Studio posso sviluppare app usando queste librerie sia per iOS che per Android usando C# e .NET
- Visual Studio include anche Windows 10 + Phone 8.1
- Microsoft Visual Studio non ha paragoni -> integrazione Xamarin e scelta preferenziale.

INTRO A XAMARIN.ANDROID

Xamarin.Android

- E’ la parte di Xamarin che consente di sviluppare app native per Android usando C#
- Concetti per lo sviluppo Android sempre validi e fondamentali
- Librerie, oggetti, nomenclatura uguale allo sviluppo nativo
- Solo che:
  - Uso C#
  - Namespace, classi, tipi, OOP ecc.
  - Async/Await
  - Garbage collection
- Visual Studio ha strumenti integrati, incluso designer

INTRO A XAMARIN.IOS

Xamarin.iOS

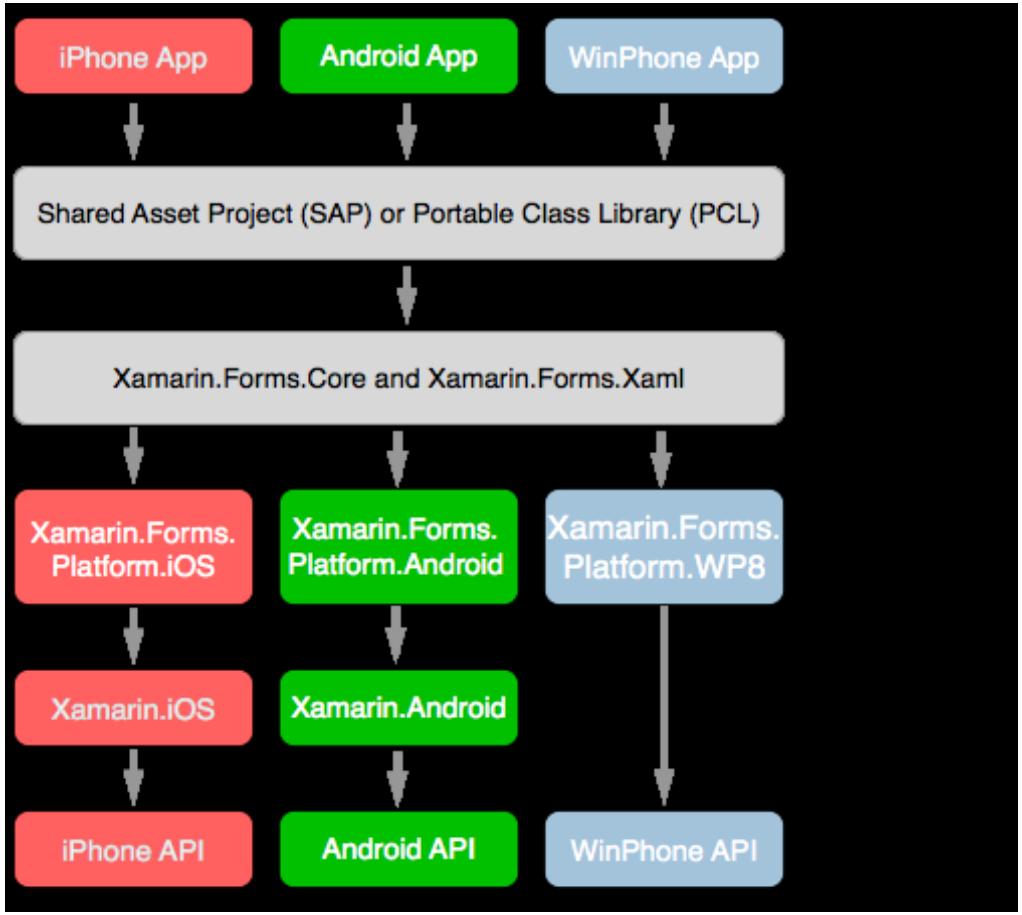
- E’ la parte di Xamarin che consente di sviluppare app native per iOS usando C#
- Concetti per lo sviluppo iOS sempre validi e fondamentali
- Librerie, framework, nomenclatura uguale allo sviluppo nativo (es. Cocoa)
- Solo che:
  - Uso C#
  - Namespace, classi, tipi, OOP ecc.
  - Async/Await
  - Garbage collection
- Visual Studio ha strumenti integrati, incluso designer
- Mi serve un Mac! (o in rete o in Cloud)

INTRO A XAMARIN.FORMS

Xamarin.Forms

- Non sempre si hanno app così platform-specific
- Nel 2014, Xamarin introduce Xamarin.Forms
- Consente di creare, con una sola code-base, interfacce utente che girino su più piattaforme
- Interfacce utente va inteso in senso ampio
- Riprende l' *eXtensible Application Markup Language* (XAML)
- Approccio unificato e semplificato
- Consente l'accesso a funzionalità native tramite dependency injection

Struttura delle solution Xamarin.Forms



Solution C# con vari progetti (immagine non mia ☺)

Struttura delle solution Xamarin.Forms

- Xamarin 4 introduce supporto per UWP (Win 10)
  - Progetti iOS, Android, UWP, Win Phone 8.1, Win 8.1
- Il concetto di base è la condivisione del codice tra diversi tipi di progetti
- La condivisione avviene mediante Portable Class Library (PCL) oppure Shared Project
- Differenze:
  - PCL: genera .dll riutilizzabile, no codice platform-specific, sì unit test, supporto XAML
  - Shared Project: visibile solo nella solution, no .dll, si codice platform-specific tramite conditional compilation (#if, #elif, #else), no supporto XAML
- Startup: classe App e istanza pagina principale più eventi per ciclo di vita

Portable Class Library

- E’ necessario selezionare un set di tecnologie da supportare
- Il risultato della compilazione di una PCL è un assembly
- Ciò che viene condiviso tra i progetti è un contenuto binario
- Non è possibile inserire codice platform-specific

Shared Projects

- Sono dei semplici contenitori per qualsiasi codice .NET
- Non vengono compilati & non producono alcun assembly
- Con VS2015 sono supportati nativamente (basta un semplice Add Reference), altrimenti occorre utilizzare l'extension «Shared Project Reference Manager»

Shared Projects

- Shared Projects let you write common code that is referenced by a number of different application projects

- **The code is compiled as part of each referencing project and can include compiler directives** to help incorporate platform-specific functionality into the shared code base.

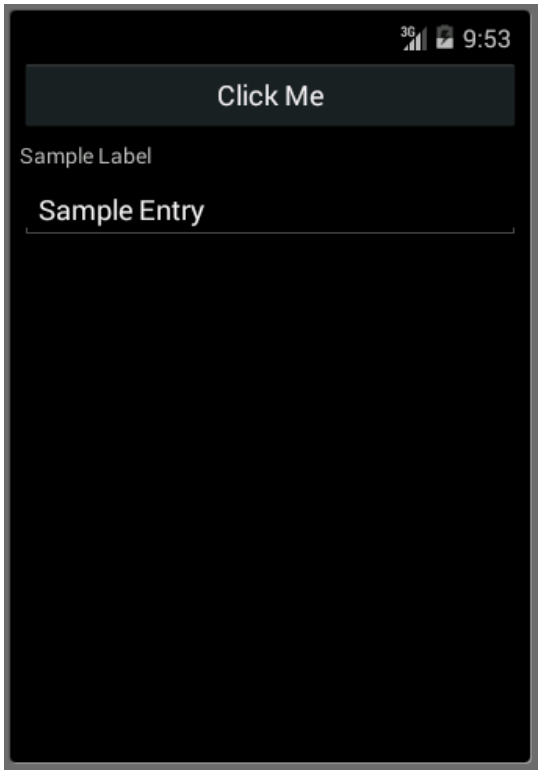
L’interfaccia grafica

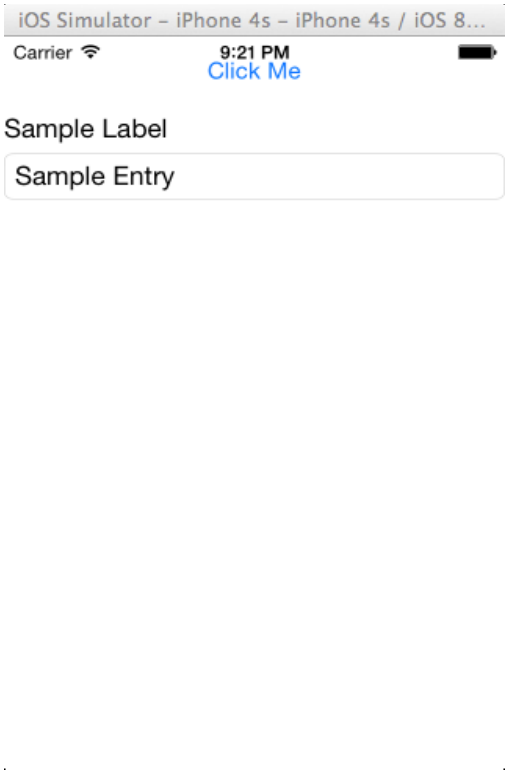
- Xamarin basa la UI sul concetto di Page, come in altri sistemi
- Xamarin.Forms consente definizione UI in modalità imperativa (in codice C#) e in modalità dichiarativa (XAML)
- XAML è un linguaggio di markup di derivazione XML per definire interfacce grafiche
- Creato da Microsoft nei primi 2000 per WPF, ripreso per Silverlight, Windows Phone, UWP
- In Xamarin.Forms leggermente meno versatile ma ugualmente efficace
- Ricorda sempre: ciò che faccio in XAML si può fare anche in C#

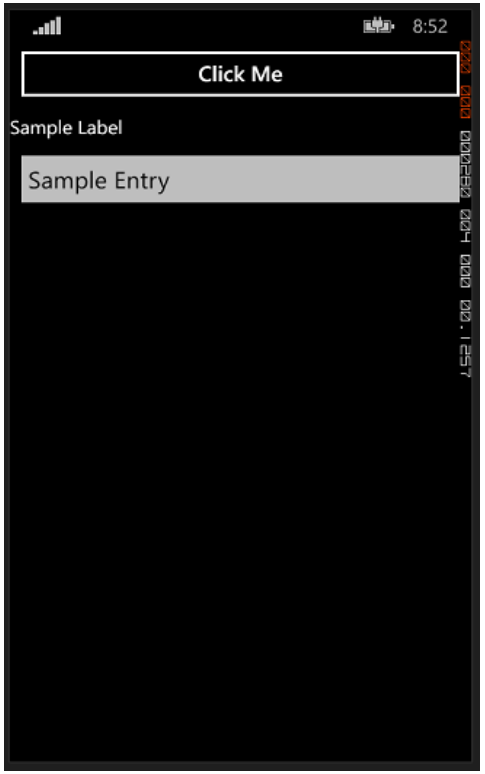
L’interfaccia grafica

```
<?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="XaSamples.SimplePage">
  <StackLayout Orientation="Vertical">
    <ActivityIndicator IsRunning="True"/>
    <Button Text="Click Me" HorizontalOptions="Center" VerticalOptions="Center"
Clicked="Button_Clicked" />
    <Entry Keyboard="Email" Placeholder="Enter your email" x:Name="Entry1"
TextChanged="Entry1_TextChanged"
           HorizontalOptions="Center" VerticalOptions="Center" />
  </StackLayout>
</ContentPage>
```

L’interfaccia grafica







L’interfaccia grafica

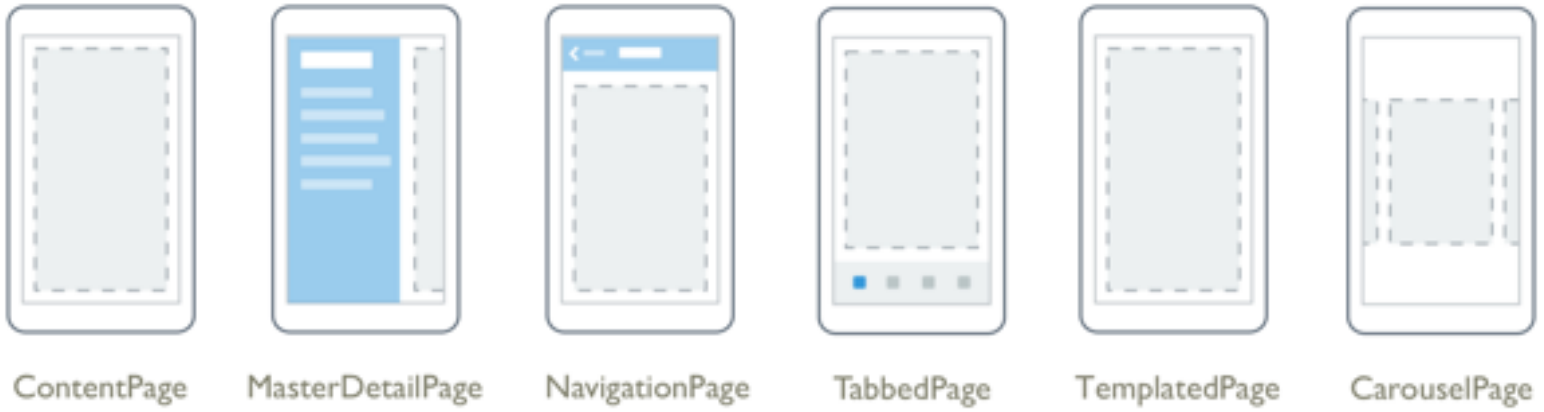
```
MainPage = new ContentPage
            {
                Content = new StackLayout
                {
                    VerticalOptions = LayoutOptions.Center,
                    Children = {
                        new Label {
                            XAlign = TextAlignment.Center,
                            Text = "Welcome to Xamarin Forms!"
                        }
                    }
                }
            };
```

L’interfaccia grafica

- L’interfaccia si basa sul concetto di Page, contenitore root
- Diversi tipi di page:
  - ContentPage (default)
  - MasterDetailPage
  - NavigationPage
  - TabbedPage
  - CarouselPage

Pagine e navigazione

- Xamarin Forms supporta nativamente differenti tipi di pagine, adatti per tutti gli scenari



Pagine e navigazione

- **ContentPage**
  Una ContentPage mostra il contenuto di una singola View, organizzata con controlli di tipo Layout o View

```
<ContentPage
  xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  x:Class="App1.NormalPage">
  <StackLayout>
    <Label Text="Label" />
    <Button Text="Confirm!" />
    <DatePicker />
    <ListView>
      <ListView.ItemTemplate>
        <DataTemplate />
      </ListView.ItemTemplate>
    </ListView>
  </StackLayout>
</ContentPage>
```

Pagine e navigazione

- **MasterDetailPage**
  Una MasterDetailPage mostra al suo interno due sotto-pagine (Master e Detail), e permette la navigazione tramite la proprietà booleana IsPresented

```
<MasterDetailPage
  xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  x:Class="App1.MainMasterPage" Title="Sample App">

  <MasterDetailPage.Master>
    <ContentPage>

    </ContentPage>
  </MasterDetailPage.Master>
  <MasterDetailPage.Detail>
    <ContentPage>

    </ContentPage>
  </MasterDetailPage.Detail>

</MasterDetailPage>
```

Pagine e navigazione

- **NavigationPage**
  Una NavigationPage è adatta per gestire uno stack di pagine e fornisce elementi di UI per la navigazione

- Come si utilizza?
  Si utilizza una normale ContentPage e la si «wrappa» attorno ad una NavigationPage tramite il suo costruttore

- this.MainPage = new NavigationPage
  (new NormalPage());

Pagine e navigazione

- **TabbedPage**
  Una TabbedPage mostra il contenuto di più ContentPage contemporaneamente. L'utente utilizza le «tab» (schede) per spostarsi da una all'altra

```
<?xml version="1.0" encoding="utf-8" ?>
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="App1.MainTabbedPage">
    <ContentPage Title="Sezione 1">
      <Button Text="Click 1" TextColor="Black" />
    </ContentPage>

    <ContentPage Title="Sezione 2">
      <Button Text="Click 2" TextColor="Black" />
    </ContentPage>

    <ContentPage Title="Sezione 3">
      <Button Text="Click 3" TextColor="Black" />
    </ContentPage>
</TabbedPage>
```

Pagine e navigazione

- **CarouselPage**
  Una CarouselPage mostra il contenuto di più ContentPage contemporaneamente. L'utente si sposta da una pagina all'altra tramite le gesture

```
<CarouselPage
  xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  x:Class="App1.MainCarouselPage">
  <ContentPage>
    <StackLayout />
  </ContentPage>
  <ContentPage>
    <StackLayout />
  </ContentPage>
  <ContentPage>
    <StackLayout />
  </ContentPage>
  <ContentPage>
    <StackLayout />
  </ContentPage>

</CarouselPage>
```

Pagine e navigazione

- I diversi tipi di Page ereditano dalla classe Xamarin.Forms.Page ed espongono proprietà interessanti

- public string BackgroundImage { get; set; }
- public FileImageSource Icon { get; set; }
- public bool IsBusy { get; set; }
- public Thickness Padding { get; set; }
- public string Title { get; set; }

- public IList<ToolbarItem> ToolbarItems { get; }

Pagine e navigazione

- Si scatena la navigazione dal code-behind di una qualsiasi tipologia di pagina

- await this.Navigation.PushAsync
  (new DestinationPage());

- await this.Navigation.PushModalAsync
  (new DestinationPage());

Pagine e navigazione

- I metodi PushAsync() e PushModalAsync() non permettono il passaggio di un parametro alla pagina di destinazione

- Soluzione?
  Sfruttare un costruttore custom delle nostre pagine!

Pagine e navigazione

- Sotto Android e Windows Phone la zona di display dedicata alla status bar in alto è fuori dal nostro controllo XAML
- Sotto iOS il nostro codice XAML può scrivere all'interno della status bar del sistema operativo
- Quindi?
  Prevedere un Padding="0,20,0,0" per la piattaforma iOS

DEFINIZIONE DELL' INTERFACCIA UTENTE

L’interfaccia grafica

- Gli elementi dell’interfaccia sono organizzati in contenitori detti Layout
- Diversi tipi di Layout:
  - StackLayout
  - Grid
  - AbsoluteLayout
  - RelativeLayout
  - Frame
  - ScrollView
  - ContentView

L’interfaccia grafica

- I diversi tipi di contenitori ereditano dalla classe Xamarin.Forms.Layout<View> ed espongono tutti la proprietà

- public IList<View> Children { get; set; }

L’interfaccia grafica

- Gli elementi dell’interfaccia si definiscono View
- UIView in iOS, Widget in Android, Controls in Windows
- Controlli comuni:

- Button
- Label, Entry, Editor
- DatePicker/TimePicker
- ActivityIndicator
- Image
- WebView
- Picker
- ScrollView
- SearchBar
- Slider
- Stepper
- Switch
- Box / Frame

L’interfaccia grafica

- Tutti gli elementi dell’interfaccia utente ereditano dalla classe Xamarin.Forms.View

- public LayoutOptions HorizontalOptions { get; set; }
- public LayoutOptions VerticalOptions { get; set; }
- public Color BackgroundColor { get; set; }
- public bool IsEnabled { get; set; }
- public bool IsFocused { get; }
- public bool IsVisible { get; set; }
- public double Opacity { get; set; }
- public double Rotation { get; set; }

L’interfaccia grafica

- Tutti gli elementi dell’interfaccia utente ereditano dalla classe Xamarin.Forms.View

- public bool InputTransparent { get; set; }
- public double Scale { get; set; }
- public Style Style { get; set; }
- public double TranslationX { get; set; }
- public double TranslationY { get; set; }

L’interfaccia grafica

- Per evitare di cablare i valori di FontSize è possibile utilizzare
- Device.GetNamedSize(NamedSize.Large, typeof(Label));

L’interfaccia grafica

- E’ possibile definire stili per uniformare l’aspetto della nostra interfaccia utente
- Gli stili possono essere definiti come locali ad una Page, oppure locali ad un certo contenitore della nostra UI
- In ogni caso, uno Style viene inserito all’interno delle Resource di Page o Layout

L’interfaccia grafica

- Stile definito all’interno di una Page

- ```
  <ContentPage.Resources>
      <ResourceDictionary>
          <Style x:Key=“nomeStile" TargetType="Button">
              <Setter Property="HorizontalOptions" Value="Center" />
              <Setter Property="BorderWidth" Value="3" />
          </Style>
      </ResourceDictionary>
  </ContentPage.Resources>
  ```
  ```

L’interfaccia grafica

- Uno stile può ereditare da un altro stile e modificarne la definizione

- ```
  <ContentPage.Resources>
      <ResourceDictionary>
          <Style x:Key=“s1" TargetType="Button">
          </Style>
          <Style x:Key=“s2" TargetType="Button“
                  BasedOn=“{StaticResource s1}”>
          </Style>
      </ResourceDictionary>
  </ContentPage.Resources>
  ```
  ```

L’interfaccia grafica

- Stile definito all’interno di un qualsiasi Layout<View>

- ```
  <StackLayout.Resources>
      <ResourceDictionary>
          <Style x:Key="nomeStile" TargetType="Label">
              <Setter Property="HorizontalOptions" Value="Center" />
              <Setter Property="BorderWidth" Value="3" />
          </Style>
      </ResourceDictionary>
  </StackLayout.Resources>
  ```
  ```

L’interfaccia grafica

- E’ possibile definire uno stile senza specificare x:Key
- In questo caso, lo stile diventa quello predefinito per quel tipo particolare di oggetto sulla UI
-
```
<StackLayout.Resources>
        <ResourceDictionary>
            <Style TargetType=“Entry">
                <Setter Property=“Fontsize" Value=“24" />
            </Style>
        </ResourceDictionary>
</StackLayout.Resources>
```

L’interfaccia grafica

- Per referenziare uno stile definito nelle Resource, si utilizzano
  StaticResource
  DynamicResource

- <Entry Style="{StaticResource nomeStile}" />
  <Label Style="{StaticResource header}" />

- StaticResource elabora lo stile all’apertura della pagina, una volta sola

- DynamicResource permette il cambio dello stile a run-time

Utilizzo del tag <OnPlatform />

- E’ possibile utilizzare il tag <OnPlatform /> per customizzare i valori delle proprietà XAML in base alla piattaforma

- ```
  <Setter Property="BackgroundColor">
      <Setter.Value>
          <OnPlatform x:TypeArguments="Color"
                      Android="#404040"
                      WinPhone="Red" />
      </Setter.Value>
  </Setter>
  ```
  ```

Utilizzo del tag <OnPlatform />

- E’ possibile utilizzare il tag <OnPlatform /> per customizzare i valori delle proprietà XAML in base alla piattaforma

- <ContentPage.Padding>
- <OnPlatform
- x:TypeArguments="Thickness"
- iOS="0,20,0,0"
- Android="0"
- WinPhone="0" />
- </ContentPage.Padding>

Utilizzo del tag <OnPlatform />

- E’ possibile utilizzare il tag <OnPlatform /> per customizzare i valori delle proprietà XAML in base alla piattaforma

- <Label.TextColor>
- <OnPlatform
- x:TypeArguments= "Color"
- iOS="White"
- Android="Red"
- WinPhone="Green" />
- </ContentPage.Padding>

Utilizzo del tag <OnPlatform />

- La classe OnPlatform espone (nella versione corrente di Xamarin Forms) tre proprietà per distinguere le piattaforme:
  Android
  iOS
  WinPhone
  per determinare il valore di una proprietà in base alla piattaforma

- E' necessario specificare x:TypeArguments con il tipo restituito

- Non è necessario specificare tutte le piattaforme

- La classe OnPlatform è disponibile anche in C#

Rispondere alle azioni con C#

- Gli elementi dell'interfaccia:
  - Ricevono input utente
  - Eseguono azioni per loro natura
- Scatenano **eventi**
  - Es. Button -> Click
  - Es. Entry -> TextChanged
- In C#, si eseguono azioni a seguito di eventi mediante «gestori»
- Non solo eventi, ma anche Command (vedi Model-View-ViewModel & Data-Binding)

Rispondere alle azioni con C#

- Xamarin Forms prevede la classe GestureRecognizer
- TapGestureRecognizer
- PinchGestureRecognizer
- PanGestureRecognizer

ACCESSO AI DATI

Quando **non** usare un database SQL

- **Impostazioni / Settaggi / Settings**
  Ogni piattaforma hanno un meccanismo built-in per salvare/caricare le impostazioni delle nostre app

- **File di testo**
  File scaricati dalla rete possono essere salvati direttamente su file system

- **Serializzazione / Deserializzazione**
  Usiamo XML / JSON per salvare oggetti .NET

Quando usare un database

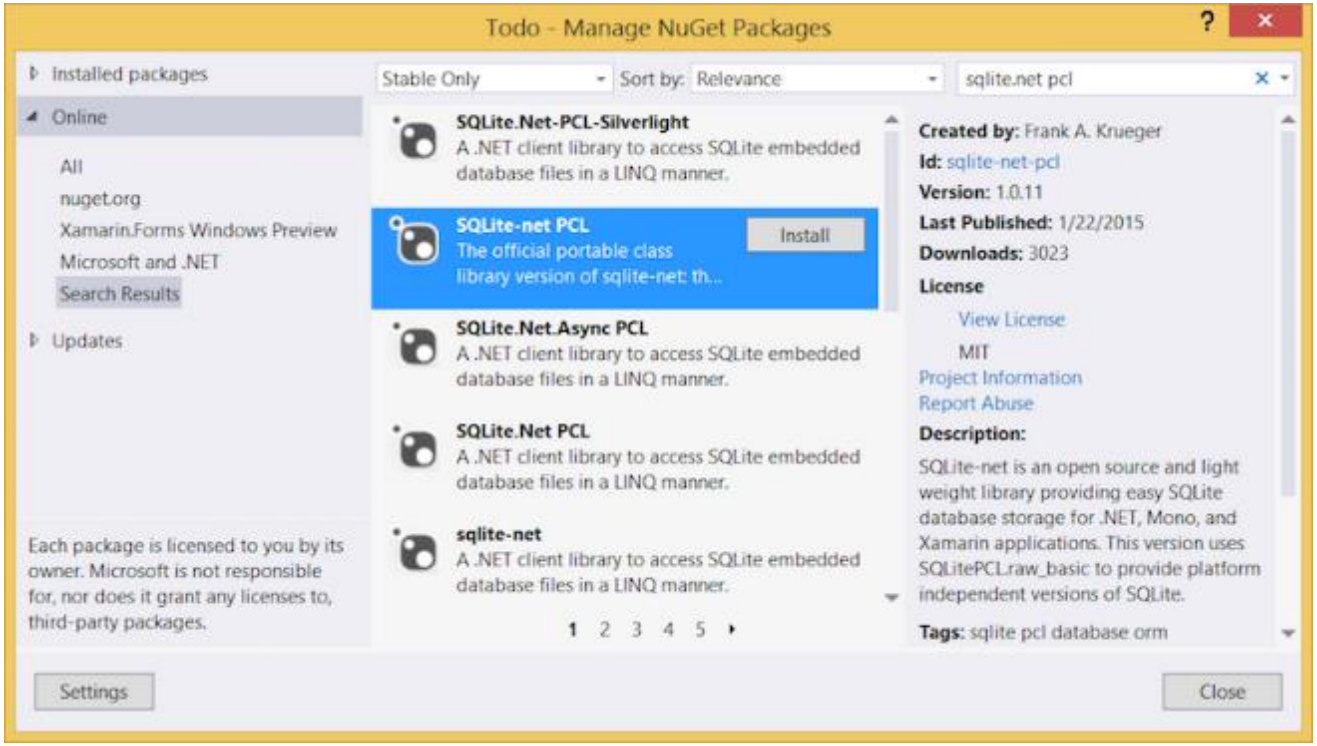
- SQL databases allow efficient storage of structured data
- Possibilità di query complessi per l'estrazione dei dati
- I risultati possono essere ordinati ed aggregate
- Grande mole di dati con il minimo sforzo

Accesso ai dati

- Cosa scelgo? Naturalmente SQLite
  - Già presente su iOS e Android
  - Facilmente installabile su Windows
  - Librerie client da NuGet
  - Serverless
- Consente le normali operazioni C.R.U.D. (Create/Read/Update/Delete) più query SQL/LINQ
- Eseguo le operazioni con normale codice C#
- La connessione al db è platform-specific
- Implemento connessione nei 5 progetti poi uso dep. Injection
- Le tabelle che ottengo sono rappresentazioni .NET -> Binding

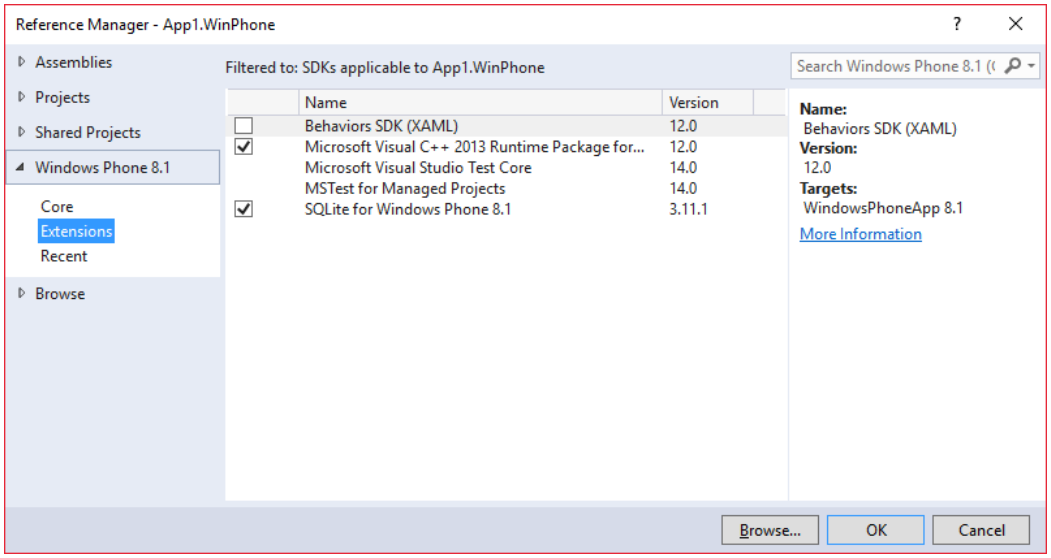
Accesso ai dati

- Via NuGet aggiungere il package su tutti i progetti
  `Install-Package sqlite-net-pcl`



Accesso ai dati

- Nel progetto dedicato a Windows Phone aggiungere una reference a «SQLite for Windows Phone 8.1»



- Non è possibile utilizzare «Any CPU»

Accesso ai dati

- La stringa di connessione deve essere platform-specific
- Nella Portable Class Library prevedere un'interfaccia:

```
public interface IDbConnection
{
    SQLiteConnection GetConnection();
}
```

- E' responsabilità di ogni piattaforma quella di generare la stringa di connessione verso il database

Accesso ai dati

- Windows Phone

```
[assembly: Dependency(typeof(SQLite_WinPhone))]
namespace App1.WinPhone.Db
{
    public class SQLite_WinPhone : IDbConnection
    {
        public SQLiteConnection GetConnection()
        {
            var sqliteFilename = "TodoSQLite.db3";
            string path = Path.Combine(ApplicationData.Current.LocalFolder.Path,
                sqliteFilename);
            // Create the connection
            var conn = new SQLite.SQLiteConnection(path);
            // Return the database connection
            return conn;
        }
    }
}
```

Accesso ai dati

- Android

```
[assembly: Dependency(typeof(SQLite_Android))]
namespace App1.Droid.Db
{
    public class SQLite_Android : IDbConnection
    {
        public SQLiteConnection GetConnection()
        {
            var sqliteFilename = "TodoSQLite.db3";
            string documentsPath = System.Environment.GetFolderPath
                (System.Environment.SpecialFolder.Personal);
            var path = Path.Combine(documentsPath, sqliteFilename);

            var conn = new SQLiteConnection(path);

            return conn;
        }
    }
}
```

Accesso ai dati

- iOS

```
[assembly: Dependency(typeof(SQLite_iOS))]
namespace App1.iOS.Db
{
    public class SQLite_iOS : IDbConnection
    {
        public SQLiteConnection GetConnection()
        {
            var sqliteFilename = "TodoSQLite.db3";
            string documentsPath = Environment.GetFolderPath
                (Environment.SpecialFolder.Personal);
            string libraryPath = Path.Combine(documentsPath, "..",
                "Library");
            var path = Path.Combine(libraryPath, sqliteFilename);
            // Create the connection
            var conn = new SQLiteConnection(path);
            // Return the database connection
            return conn;
        }
    }
}
```

Accesso ai dati

- Una volta ottenuta la connessione al database, l'approccio sulle diverse piattaforme è semplice ed unificato

- Possiamo scrivere il nostro codice nella PCL cross-platform

- ```
  var db = DependencyService.Get<IDbConnection>();
  var conn = db.GetConnection();
  conn.CreateTable<Car>();
  conn.Insert(new Car());
  var table = conn.Table<Car>();
  ```
  ```

Accesso ai dati

- Possiamo usare LINQ per le nostre query

- ```
  var db = DependencyService.Get<IDbConnection>();
  var conn = db.GetConnection();
  var table = conn.Table<Car>()
        .Where(c => c.Brand == "Honda");
  ```

- https://developer.xamarin.com/guides/cross-platform/application_fundamentals/data/part_3_using_sqlite_orm/
  ```

ACCESSO ALLA RETE CON LA CLASSE HTTPCLIENT

Accesso alla rete

- Xamarin Forms non mette a disposizione la classe HttpClient
- Affidarsi al seguente package Nuget: https://www.nuget.org/packages/Microsoft.Net.Http

Accesso alla rete

- HttpClient c = new HttpClient();
  c.BaseAddress = "http://";
  c.Timeout = 90; // milliseconds
  c.GetByteArrayAsync(…);
  c.GetStreamAsync(…);
  c.GetStringAsync(…);
  c.PostAsync(…);
  c.SendAsync(…);

Accesso alla rete

- Aggiungere il package
  https://www.nuget.org/packages/Microsoft.AspNet.WebApi.Client/

  per avere il supporto a JSON, XML ed i
  «Form Url Encoded Data»

- Attenzione al multi-piattaforma!

Accesso alla rete

- Per Android

- HttpClient c = new HttpClient();
  var task = c.GetStringAsync("http://website");
  var html = task.Result;

Accesso alla rete

- Per le piattaforme Microsoft
  (Windows 8/8.1, UWP, WinPhone)

- HttpClient c = new HttpClient();
  var html = await c.GetStringAsync("http://website");

ACCESSO ALLE API NATIVE

Accesso alle API native

- Xamarin.Forms è prevalentemente un layer di unificazione per la creazione di interfacce e accesso ai dati
- Spesso servono funzioni specifiche di un device
  - SMS/Messaging
  - Sensori
  - Hardware in generale
- Xamarin.Forms consente *dependency injection*
- Classe DependencyService + interfacce di servizio
- Vediamolo in azione

Accesso alle API native - DependencyService

- Definire un'interfaccia nella PCL che definisca il servizio platform-specific da implementare

```
public interface ITextToSpeech
{
    void Speak(string text);
}

public interface ILocalNotification
{
    void Show(string text, object parameter);
}
```

Accesso alle API native - DependencyService

- Implementare una classe in ciascun progetto (Android, iOS, Windows Phone, etc.)
- La classe **deve** implementare l'interfaccia prevista
- La classe, trovandosi nel progetto platform-specific, può fare uso delle API native per quella piattaforma

Accesso alle API native - DependencyService

```
[assembly: Xamarin.Forms.Dependency(typeof(TextToSpeechImplementation))]
namespace App1.WinPhone
{
    public class TextToSpeechImplementation : ITextToSpeech
    {
        public async void Speak(string text)
        {
            // Codice
        }
    }
}
```

Accesso alle API native - DependencyService

```
[assembly: Xamarin.Forms.Dependency(typeof(TextToSpeechImplementation))]
namespace App1.Droid
{
    public class TextToSpeechImplementation : Java.Lang.Object,
        ITextToSpeech, TextToSpeech.IOnInitListener
    {
        TextToSpeech speaker;
        string toSpeak;

        public void Speak(string text)
        {
            // codice
        }

        IOnInitListener implementation
    }
}
```

Accesso alle API native - DependencyService

- Dall’applicazione Xamarin Forms è possibile accedere alle funzionalità delle API native tramite interfaccia

```
var engine = DependencyService.Get<ITextToSpeech>();
engine.Speak("Hello world");
```

- Chiedo al motore di dependency injection native di Xamarin Forms di fornirmi una classe che implementi l’interfaccia richiesta

Accesso alle API native - DependencyService

- Esistono metodi alternativi?
- Ricordiamoci gli Shared Projects e le Portable Class Library
- Sfruttiamo i plugin di Xamarin Forms
  (morale: non reinventiamo la ruota ☺)
- https://github.com/xamarin/plugins