Mobile Dev + Test
A TECHWELL EVENT
APRIL 24–28, 2017 | SAN DIEGO, CA

Building Cross-Platform Mobile Applications with C# and Xamarin

MIKE BENKOVICH

mike@benko.com | www.Benkotips.com

Intro to Xamarin

Silo Approach

No shared code • Many languages & development environments • Multiple teams

Write Once, Run Anywhere

Limited native API access • Slow performance • Poor user experience

Xamarin's Unique Approach

Shared C# codebase • 100% native API access • High performance

Windows APIs

Microsoft.Phone	Microsoft.Networking	Windows.Storage	Windows.Foundation	Microsoft.Devices
System.Net	System	System.IO	System.Linq	System.Xml
System.Data	System.Windows	System.Numerics	System.Core	System.ServiceModel

C#

iOS – 100% API Coverage

MapKit	UIKit	iBeacon	CoreGraphics	CoreMotion
System.Net	System	System.IO	System.Linq	System.Xml
System.Data	System.Windows	System.Numerics	System.Core	System.ServiceModel

C#

Android – 100% API Coverage

Text-to-speech	ActionBar	Printing Framework	Renderscript	NFC
System.Net	System	System.IO	System.Linq	System.Xml
System.Data	System.Windows	System.Numerics	System.Core	System.ServiceModel

C#

Building User Interfaces

Traditional UI approach

3 Native User Interfaces

Shared App Logic

Xamarin.Forms approach

Shared UI Code

Shared C# Logic

Shared User Interface

Shared App Logic

UrhoSharp

CocosSharp

Code sharing stats

	iCircuit	Touch Draw
Android		
iOS		
Mac		
Windows		✕

Native Performance

Xamarin.iOS does full Ahead Of Time (AOT) compilation to produce an ARM binary for Apple's App Store.

Xamarin.Android takes advantage of Just In Time (JIT) compilation on the Android device.

✓Always Up-to-Date

Full support for:

- Apple Watch
- Android Wear
- Amazon Fire TV
- Google Glass
- and much more

Same-day support:

- iOS 5
- iOS 6
- iOS 7
- iOS 8
- iOS 9
- iOS 10

Anything you can do in Objective-C, Swift, or Java can be done **in C# with Xamarin.**

Let’s Build an App

Development Experience

Xamarin is included in Visual Studio

Including Community Edition!

Open Source – open.xamarin.com

Visual Studio Integration

Xamarin.com/Download

Xamarin for Visual Studio

Visual Studio – Mac (was Xamarin Studio)

Visual Studio iOS Simulator Remoting

- Multi-Touch Enabled
- Pressure Sensitive
- Super Fast
- Rotate, screenshots, location changes
- Never leave VS

*Requires Visual Studio Enterprise

Streamlined Development

Shared C# Backend

Platform Specific Code

What if we didn’t have to write this code?

What if we could access it from shared code?

Speak(“Hello World”);
AVSpeechSynthesizer
TextToSpeech
SpeechSynthesizer

Plugins for Xamarin

github.com/xamarin/plugins

Common API

Let’s use a Plugin for Xamarin

15 Minute Break

Today

Intro to Xamarin

Xamarin Forms

DevOps

Advanced Binding

Mobile Apps

http://github.com/Xamarin/dev-days-labs

Xamarin Forms

Meet Xamarin.Forms

Build native UIs for iOS, Android, and Windows

from a single, shared C# codebase.

Xamarin + Xamarin.Forms

Traditional Xamarin Approach

With Xamarin.Forms:
More code-sharing, all native

What’s included

- ✓ 40+ Pages, layouts, and controls
  (Build from code behind or XAML)
- ✓ Two-way data binding
- ✓ Navigation
- ✓ Animation API
- ✓ Dependency Service
- ✓ Messaging Center

Pages

Content MasterDetail Navigation Tabbed Carousel

Layouts

Stack Absolute Relative Grid ContentView ScrollView Frame

Controls

ActivityIndicator	BoxView	Button	DatePicker	Editor
Entry	Image	Label	ListView	Map
OpenGLView	Picker	ProgressBar	SearchBar	Slider
Stepper	TableView	TimePicker	WebView	EntryCell
ImageCell	SwitchCell	TextCell	ViewCell	

Xamarin.Forms Ecosystem

Control Comparison

Windows	Xamarin.Forms
StackPanel	StackLayout
TextBox	Entry
ListBox	ListView
CheckBox	Switch
ProgressBar	ActivityIndicator
Grid	Grid
Label	Label
Button	Button
Image	Image
Date/TimePicker	Date/TimePicker

Binding Comparison

Windows	Xamarin.Forms
DataContext	BindingContext
{Binding Property}	{Binding Property}
ItemsSource	ItemsSource
ItemTemplate	ItemTemplate
DataTemplate	DataTemplate

```
<Label Text="{Binding Color.R,
                 Converter={StaticResource intConverter},
                 ConverterParameter=255,
                 StringFormat='R={0:X2}'}" />
```

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

Platform Customization

Native UI from shared code

```xml
<?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="MyApp.MainPage">
<TabbedPage.Children>
<ContentPage Title="Profile" Icon="Profile.png">
    <StackLayout Spacing="20" Padding="20"
                 VerticalOptions="Center">
        <Entry Placeholder="Username"
               Text="{Binding Username}"/>
        <Entry Placeholder="Password"
               Text="{Binding Password}"
               IsPassword="true"/>
        <Button Text="Login" TextColor="White"
                BackgroundColor="#77D065"
                Command="{Binding LoginCommand}"/>
```

Xamarin.Forms

Demo

Latest and Greatest in Xamarin.Forms

Xamarin.Forms is Open Source!

open.xamarin.com

XAML Previewer

Debug | iPhoneSimulator › iPhone 6s iOS 9.3 | Xamarin Studio Business | Press '⌘.' to search

App.cs | ImageCircleRenderer.cs | ImageCircleRenderer.cs | CoffeesPage.xaml

Hide Preview

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ContentPage
3      xmlns="http://xamarin.com/schemas/2014/forms"
4      xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
5      xmlns:local="clr-namespace:CoffeeCups;assembly=CoffeeCups"
6      xmlns:controls="clr-namespace:ImageCircle.Forms.Plugin.Abstractions;assembl
7      x:Class="CoffeeCups.CoffeesPage"
8      Title="Cups Of Coffee">
9       <AbsoluteLayout HorizontalOptions="FillAndExpand" VerticalOptions="FillAnd
10         <Grid AbsoluteLayout.LayoutFlags="All"
11               AbsoluteLayout.LayoutBounds="0,0,1,1"
12               RowSpacing="0">
13             <Grid.RowDefinitions>
14                 <RowDefinition Height="Auto"/>
15                 <RowDefinition Height="*"/>
16                 <RowDefinition Height="Auto"/>
17             </Grid.RowDefinitions>
18             <StackLayout Orientation="Horizontal" Padding="16" Spacing="16" Bacl
19                 <Label Text="Made at Home?"
20                        TextColor="White"
21                        VerticalOptions="Center"
22                        HorizontalOptions="Start"
23                        FontSize="16">
24                 <Label FontSize=
```

Device: Phone Tablet Platform: Android iOS

Forms Embedding

```
// iOS
UIViewController viewController = new
MyFormsPage().AsViewController();
```

```
// Android
Fragment fragment = new MyFormsPage().AsFragment();

// UWP
Page page = new MyFormsPage().AsPage();

// How about….
var page = new ContentPage {
        Content = someView
}.AsViewController();
```

Fast Renderers

FORMS STRUCTURE

Fast Renderers

WHAT IS PRODUCED NOW

Fast Renderers

AFTER FAST RENDERERS + COMPRESSION

Layout Compression

Native View Declarations

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage Title="Xaml Native Views" xmlns="http://xamarin.com/schemas/2014/forms" xmlns:ios="clr-namespace:UIKit;assembly=Xamarin.iOS;target
    <ContentPage.Resources>
        <ResourceDictionary>
            <iosControls:UIColorConverter x:Key="colorConverterIOS" />
        </ResourceDictionary>
    </ContentPage.Resources>
    <ContentPage.Content>
        <StackLayout BackgroundColor="#eeeeee">
            <Label Text="This text color is Bound to the picker" FontSize="22" FontAttributes="Bold" HorizontalTextAlignment="Center"
                   TextColor="{Binding FormsSelectedColor}"></Label>
            <Button Text="Change Text From Forms" Command="{Binding ChangeTextCommand}" />
            <ios:UILabel BackgroundColor="{x:Static ios:UIColor.LightGray}" Text="{Binding LabelName,}"
            <ios:UITextView Text="{Binding LabelName, Mode=TwoWay, UpdateSourceEventName=Ended}" />
            <iosColorPicker:ColorPickerView
                SelectedColor="{Binding Path=FormsSelectedColor,
                                        Mode=TwoWay,
                                        Converter={StaticResource colorConverterIOS},
                                        UpdateSourceEventName=ColorPicked}" >
                <x:Arguments>
                    <iosCoreGraphics:CGRect>
                        <x:Arguments>
                            <x:Double>0</x:Double>
                            <x:Double>0</x:Double>
```

Nightly Builds

https://github.com/xamarin/Xamarin.Forms/wiki/Nightly-Builds

More Performance Enhancements

- Android Startup
- Reduce Android GPU Overdraw
- Single DLL

15 Minute Break

Today

Intro to Xamarin

Xamarin Forms

DevOps

Advanced Binding

Mobile Apps

http://github.com/Xamarin/dev-days-labs

https://github.com/mbenko/MDT17AppForms

Microsoft’s complete mobile DevOps solution

Develop **Test** **Build** **Distribute** **Monitor**

Test
Analytics
Build
Distribute
Live Update
Crash Reporting
A/B Testing
Storage
Identity
Push Notifications
Remote Config.
Tables

mobile.azure.com

XAML Binding

Data Binding

Data Binding involves creating a loose relationship between a source property and a target property, source and target are unaware of each other

Binding acts as an *intermediary* – moving the data between the source and target

Creating Bindings

Bindings require **three pieces of information**

Creating bindings [Source]

```
Person person = new Person() { Name = "Homer Simpson", ... };

Entry nameEntry = new Entry();

Binding nameBinding = new Binding();
nameBinding.Source = person;

...
```

1

Binding identifies the source of the binding data – this is where the data comes from, in this case it's a single person defined in our application

Creating bindings [Path]

```
Person person = new Person() { Name = "Homer Simpson", ... };

Entry nameEntry = new Entry();

Binding nameBinding = new Binding();
nameBinding.Source = person;
nameBinding.Path = "Name";

...
```

2

Binding identifies the *property path* which identifies a property on the source to get the data from, in this case we want to get the value from the **Person.Name** property

Creating bindings [Path]

```
Person person = new Person() { Name = "Homer Simpson", ... };

Entry na

Binding
nameBind
nameBind

...
```

More Path Examples

```
new Binding("Property")
new Binding("Property.Child")
new Binding("Property[Key]")
new Binding("Property[1]")
new Binding("Item[Key]")
new Binding(".")
```

Creating bindings [Target]

```
Person person = new Person() { Name = "Homer Simpson", ... };

Entry nameEntry = new Entry();

Binding nameBinding = new Binding();
nameBinding.Source = person;
nameBinding.Path = "Name";

nameEntry.SetBinding(Entry.TextProperty, nameBinding);
```

3

Binding is associated to the target property using the **`BindableObject.SetBinding`** method

Creating bindings [Target]

```
Person person = new Person() { Name = "Homer Simpson", ... };

Entry nameEntry = new Entry();

Binding nameBinding = new Binding();
nameBinding.Source = person;
nameBinding.Path = "Name";

nameEntry.SetBinding(Entry.TextProperty, nameBinding);
```

3

This is passed the specific target property the binding will work with – this must be a **BindableProperty**

Creating bindings [Target]

```
Person person = new Person() { Name = "Homer Simpson", ... };

Entry nameEntry = new Entry();

Binding nameBinding = new Binding();
nameBinding.Source = person;
nameBinding.Path = "Name";

nameEntry.SetBinding(Entry.TextProperty, nameBinding);
```

3

… and the binding which identifies the source and the property on the source to apply

Creating bindings [XAML]

Create bindings in XAML with `{Binding}` markup extension

```
<StackLayout Padding="20" Spacing="20">
   <StackLayout.Resources>
      <ResourceDictionary>
         <Person x:Key="homer" Name="Homer Simpson" .../>
      </ResourceDictionary>
   </StackLayout.Resources>
   <Entry Text="{Binding Name,
                 Source={StaticResource homer}}" />
   ...
</StackLayout>
```

`{Binding}` takes the Path as the first unnamed argument

Source supplied through resource

Assigned to Target property

Sharing the Binding Source

- Many screens pull data from a singular object – mapping different properties to UI controls

- Can set the binding source on each binding separately, or use the `BindingContext` as the *default* binding source

```
public class Person
{
    public string Name { get; set; }
    public string Email { get; set; }
    public Gender Gender { get; set; }
}
```

Multiple Bindings

`BindingContext` supplies the data source for all bindings on the given UI element *if the* `Binding.Source` *property is unassigned*

```
Person person = new Person() { Name = "Homer Simpson", ... };
...
nameEntry.BindingContext = person;
nameEntry.SetBinding(Entry.TextProperty, new Binding("Name"));
```

Binding only identifies the **`Path`** – no source is set; it will then look to the **`BindingContext`** on the parent control as the source for the binding data

BindingContext inheritence

`BindingContext` is automatically *inherited* from parent to child

```
public partial class PersonDetailsPage : ContentPage
{
    public PersonDetailsPage (Person person)
    {
      BindingContext = person;
      InitializeComponent ();
    }
}
```

Can set the **`BindingContext`** *once* on the root page and it is then assigned to every control (child) on that page

View-to-View Bindings

`{x:Reference}` allows XAML to locate named elements and use them as a binding source

```
<StackLayout Padding="20" Spacing="20">
    <Label Text="Hello, Bindings" TextColor="Blue" ...
           BindingContext="{x:Reference slider}"
           Rotation="{Binding Value}" />
     ...
    <Slider x:Name="slider" Minimum="0" Maximum="360" />
</StackLayout>
```

Creating two-way bindings

Typically want data to flow in both directions

- source > target (always happens)
- target > source (optional)

```
nameEntry.TextChanged += (sender, e)
      => person.Name = nameEntry.Text;
```

Binding Mode

Binding **Mode** controls the direction of the data transfer

Source Property must have
public setter

```
name.SetBinding(Entry.TextProperty,
  new Binding("Name") {
      Mode = BindingMode.TwoWay
});
```

```
<Entry
  Text="{Binding Name, Mode=TwoWay}" />
```

Manually controlled through the
`Binding.Mode` property

Available Binding Modes

`BindingMode.Default` is the default value and it decides the mode based on the target property preference – either `OneWay` or `TwoWay`

Default Binding Mode

Default binding mode is *property-specific*, most are one-way by default with a few exceptions that default to two-way

```
DatePicker.Date
Entry.Text
ListView.SelectedItem
MultiPage<T>.SelectedItem
Picker.SelectedIndex
SearchBar.Text
Stepper.Value
Switch.IsToggled
TimePicker.Time
```

XAML platforms handle **binding modes differently**, best practice to get in the habit of explicitly setting the mode if it's not one-way – even if it defaults to what you want

Pushing changes to the UI

One-Way and Two-Way bindings always update the UI when the source property is changed

```
person.Email = "homer@dunkin.com";
```

```
public class Person
{
    public string Name { get; set; }
    public string Email { get; set; }
    public Gender Gender { get; set; }
    public DateTime Dob { get; set; }
    public bool IsFavorite { get; set; }
}
```

Q: How does Xamarin.Forms know **Email** has changed?

INotifyPropertyChanged

`INotifyPropertyChanged` provides change notification contract, should be implemented by any modifiable model object you bind to

```
namespace System.ComponentModel
{
   public interface INotifyPropertyChanged
   {
       event PropertyChangedEventHandler PropertyChanged;
   }
}
```

Implementing INotifyPropertyChanged

```
public class Person : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged = delegate {};

    string email;
    public string Email {
        get { return email; }
        set {
            if (email != value) {
                email = value;
                PropertyChanged(this,
                  new PropertyChangedEventArgs("Email");
            }
        }
    }
}
```

Must raise the PropertyChanged event when any property is changed – otherwise the UI will not update

INPC + Bindings

Binding will subscribe to the `PropertyChanged` event and update the target property when it sees the source property notification

Summary

- Mapping Data to Visuals
- Creating Bindings in Code
- Creating Bindings in XAML
- Working with Binding Context
- Binding Modes
- Property Change Notifications

Tasks

- Working with Incompatible Types
- Value Converters
- Simple Textual Conversions

Incompatible Types

Bindings attempt to **automatically coerce data** when C# would allow it, but sometimes the data available isn't quite what the UI needs to display

Want the text color to change based on the password strength

```
<Label Text="{Binding PasswordStrength}"
       TextColor="{Binding PasswordStrength}"
       Font="24" />
```

Value Converters

- **Value Converters** enable type coercion and formatting
- Assigned to `Converter` property of Binding
- Supports optional parameter (`Binding.ConverterParameter`)

```
public interface IValueConverter
{
   object Convert(object value,
                  Type targetType,
                  object parameter,
                  CultureInfo culture);

   object ConvertBack(object value,
                      Type targetType,
                      object parameter,
                      CultureInfo culture);
}
```

`Convert` used for source ⇨ target
`ConvertBack` used for target ⇨ source

Creating a Value Converter

```
public class PWStrengthConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        PasswordStrength pwdstr = (PasswordStrength) value;
        ...
        return Color.Red;
    }

    public object ConvertBack(object value, Ty
    {
        throw new NotSupportedException();
    }
}
```

Converter performs whatever translation is necessary to provide target with data – this can be simple conversions or even completely different objects!

Creating a Value Converter

```
public class PWStrengthConverter :
{
    public object Convert(object v                                  reInfo culture)
    {
        PasswordStrength pwdstr =
        ...
        return Color.Red;
    }

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

Provides backwards conversion for two-way binding, or can throw exception if this is not supported – this will cause a runtime failure

Using a Value Converter

Value Converter is assigned to the binding `Converter` property

```
var binding = new Binding("PasswordStrength"){
   Converter = new PWStrengthConverter()
};
```

```
<ContentPage.Resources>
  <ResourceDictionary>
    <c:PWStrengthConverter x:Key="pwsCvt"/>
  </ResourceDictionary>
</ContentPage.Resources>

<Label TextColor="{Binding PasswordStrength,
    Converter={StaticResource pwsCvt}}" />
```

Binding passes values through converter

Simple Textual Conversions

Binding has a `StringFormat` property used for text conversions

```
<Label Text="{Binding ., StringFormat='Total Cost: {0:C}'}">
    <Label.BindingContext>
        <x:Double>23.54</x:Double>
    </Label.BindingContext>
</Label>
```

Total Cost: $23.54

Binding performs a **`String.Format`** on the value when going from Source > Target

Debugging Bindings

Can use dummy converter to debug data bindings – gets called during the data transfer and provides for a convenient breakpoint location

```
10  public class DummyConverter : IValueConverter
11  {
12      public object Convert(object value, Type targetType,
13              object parameter, CultureInfo culture) {
14          return value;
15      }
16
17      public object ConvertBack(object value, Type targetType,
18              object parameter, CultureInfo culture) {
19          return value;
20      }
21  }
```

Summary

- Working with Incompatible Types
- Value Converters
- Simple Textual Conversions

Cloud Connected

The best apps stand out

Bad app experiences

- Crashes
- Hard to navigate, poor UI, bad user experience
- Features not as advertised
- Slow or laggy experience
- Data not available when you need it

Always connected?

opensignal.com/coverage-maps

Weak Signal
Strong Signal
Morocco
Algeria
Libya
Egypt
Israel
Jordan
Iran
Pakistan
Nepal
Bhutan
Bangladesh
India
Western Sahara
Mauritania
Mali
Niger
Chad
Sudan
Eritrea
Yemen
Saudi Arabia
Riyadh
Persian Gulf
United Arab Emirates
Mecca
Red Sea
Oman
Mumbai
Hyderabad
Chennai
Bengaluru
Bay of Bengal
Arabian Sea
Gulf of Aden
Djibouti
Ethiopia
Sri Lanka
Laccadive Sea
Senegal
The Gambia
Guinea-Bissau
Guinea
Sierra Leone
Liberia
Côte d'Ivoire
Burkina Faso
Ghana
Togo
Benin
Nigeria
Lagos
Cameroon
Central African Republic
South Sudan
Gulf of Guinea
Equatorial Guinea
Gabon
Uganda
Kenya
Somalia
Rwanda
Burundi
Nairobi
DR Congo
Kinshasa
Tanzania
Dar es Salaam
Luanda
Angola
Zambia
Malawi
Mozambique
Zimbabwe
Madagascar
Namibia
Botswana
Pretoria
Swaziland
Lesotho
Durban
South Africa
Cape Town
South Atlantic Ocean
Indian Ocean
Reset
2G
3G
4G
Coverage
Towers
NetworkRank™
Advanced View
Claro
Vivo
TIM
Oi
All Networks
Map data ©2016 Google, INEGI, ORION-ME
500 km
Terms of Use

Weak Signal
Strong Signal
Morocco
Algeria
Libya
Egypt
Israel
Jordan
Iran
Pakistan
Nepal
Bhutan
Bangladesh
India
Western Sahara
Mauritania
Mali
Niger
Chad
Sudan
Eritrea
Yemen
Saudi Arabia
Riyadh
Mecca
Persian Gulf
United Arab Emirates
Oman
Red Sea
Mumbai
Hyderabad
Bengaluru
Chennai
Bay of Bengal
Arabian Sea
Gulf of Aden
Djibouti
Ethiopia
Sri Lanka
Laccadive Sea
Senegal
The Gambia
Guinea-Bissau
Guinea
Sierra Leone
Liberia
Côte d'Ivoire
Burkina Faso
Ghana
Togo
Benin
Nigeria
Lagos
Cameroon
Central African Republic
South Sudan
Gulf of Guinea
Equatorial Guinea
Gabon
Uganda
Kenya
Nairobi
Somalia
Rwanda
Burundi
DR Congo
Kinshasa
Tanzania
Dar es Salaam
Luanda
Angola
Zambia
Malawi
Mozambique
Zimbabwe
Madagascar
Namibia
Botswana
Pretoria
Swaziland
Lesotho
Durban
South Africa
Cape Town
South Atlantic Ocean
Indian Ocean
Reset
2G
3G
4G
Coverage
Towers
NetworkRank™
Advanced View
Claro
Oi
TIM
Vivo
All Networks
Map data ©2016 Google, INEGI, ORION-ME
500 km
Terms of Use

What about a backend?

Infrastructure designed for Scale

100+ datacenters

Top 3 networks in the world

2x AWS, 6x Google DC Regions

G Series – Largest VM in World
32 cores, 448GB Ram, SSD…

Operational

Announced

* Operated by 21Vianet

38

Azure compute regions open today

More than AWS and Google Cloud combined

Datacenters recently added in Canada, China, and India

Why Azure Mobile Apps?

Extremely powerful

Flexible

- Easy Tables (Node.js Backend)
- App Service (ASP.NET Backend)

Cross-platform client SDK

- iOS, Android, and Windows

Open source C#

- Backend and Client SDKS on GitHub

Azure Mobile Apps

Let’s build a backend

Create a Mobile Service

```
MobileService =  new MobileServiceClient(
    "https://myapp.azurewebsites.net");
```

Create tables

```
IMobileServiceSyncTable<TodoItem> syncTable;

public async Task Init()
{
    const string path = "syncstore.db";
    var db = new MobileServiceSQLiteStore(path);
    db.DefineTable<TodoItem>();
    await MobileService.SyncContext.InitializeAsync(db);
    syncTable = MobileService.GetSyncTable<TodoItem>();

}
```

Push and pull with sync table

```
private async Task SyncAsync()
{
    await MobileService.SyncContext.PushAsync();
    var query = syncTable.CreateQuery();

    await syncTable.PullAsync("todoItems", query);
}
```

```
private async Task InsertTodoItem(TodoItem todoItem)
{
    await syncTable.InsertAsync(todoItem);
    await MobileService.SyncContext.PushAsync();
}
```

Query local table

```
public async Task<IEnumerable<TodoItem>> GetOpenItemsAsync()
{
    return await todoTable
                 .Where(item => item.Complete == false)
                 .ToEnumerableAsync();
}
```

User authentication

Secure your app with prebuilt authentication providers

- Facebook
- Twitter
- Google
- Microsoft
- Azure AD
- Anything OAuth 2

Much More

Push notifications with Notification Hub

- Maps between tags and handles

File sync

Sync files to Azure Storage

Online/Offline, just like table storage

Shopping Demo App

Azure Services:

- Cognitive Services (Emotion API)
- Storage
- On/offline sync
- Push notifications
- Authentication

What's Available:

- Complete sample app with documentation
- Individual "quick starts"

github.com/Microsoft/XamarinAzure_ShoppingDemoApp

Mobile + Server

Shared C# codebase • 100% native API access • High performance

Summary

Plan for global reach

Always cache data locally

Assume success and push updates on re-connect

Select a backend that grows with you

Get started today
xamarin.com/download
xamarin.com/azure
xamarin.com/dev-ops

Thank you!