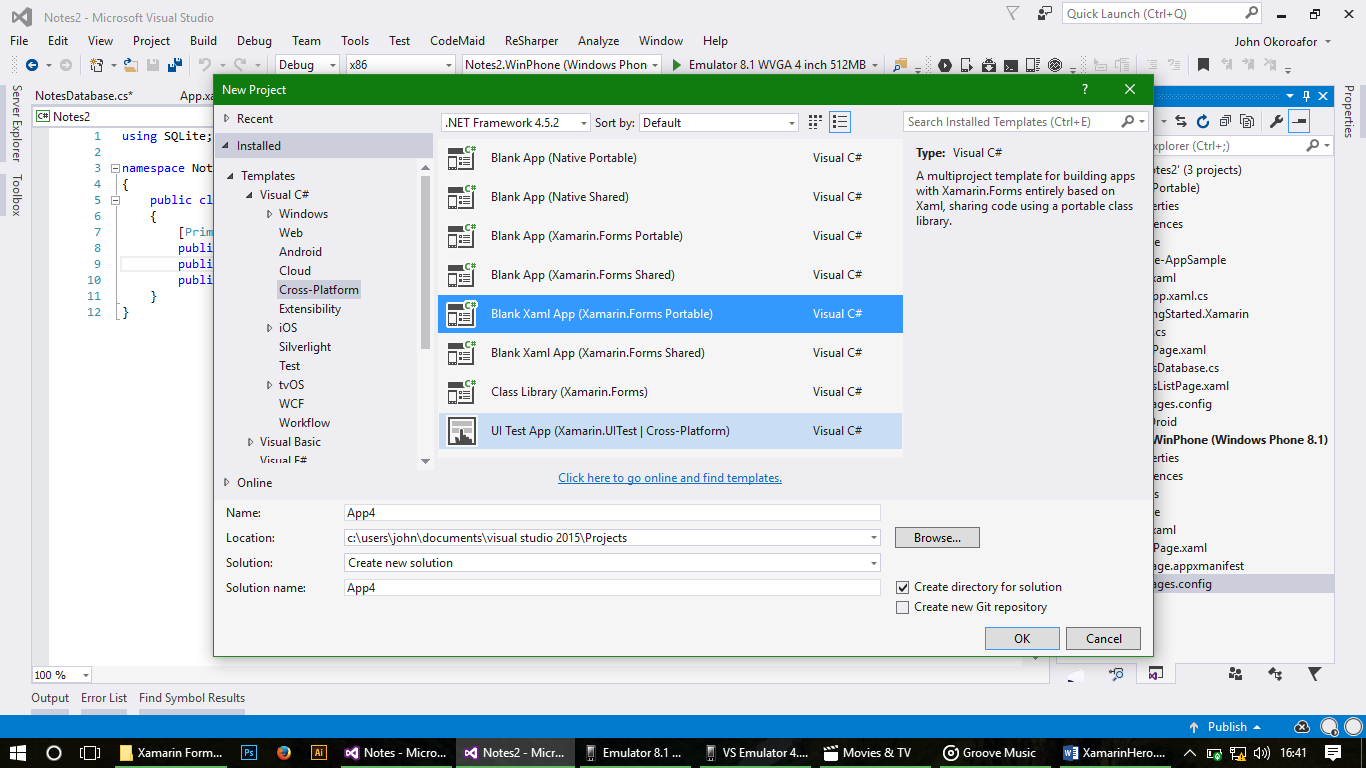
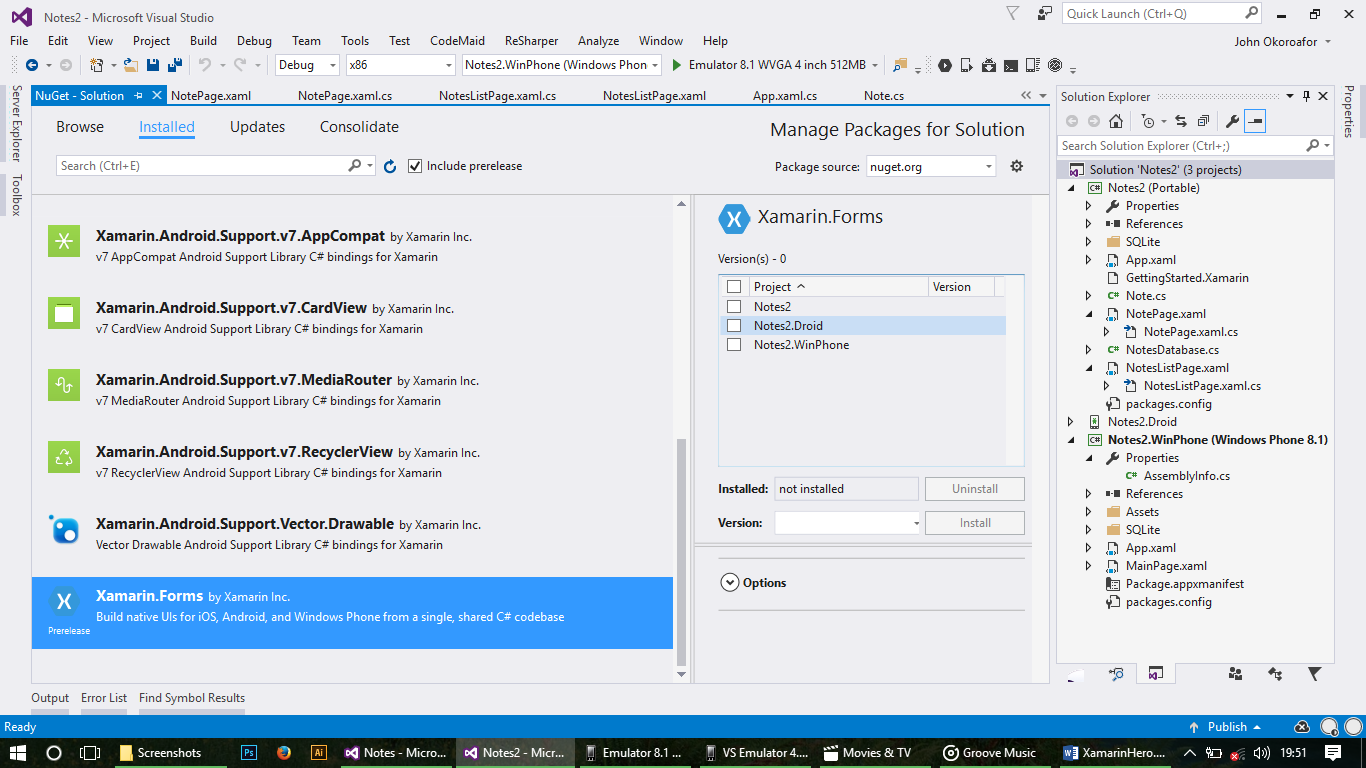
In this sample we Create a simple Note taking app on Android and windows phone 8 which would utilize and introduce us to plugins in Xamarin.Forms: the Speak plugin(for text to speech), and SQLite plugin (for saving data to a local database on the phone. Let’s get Started.

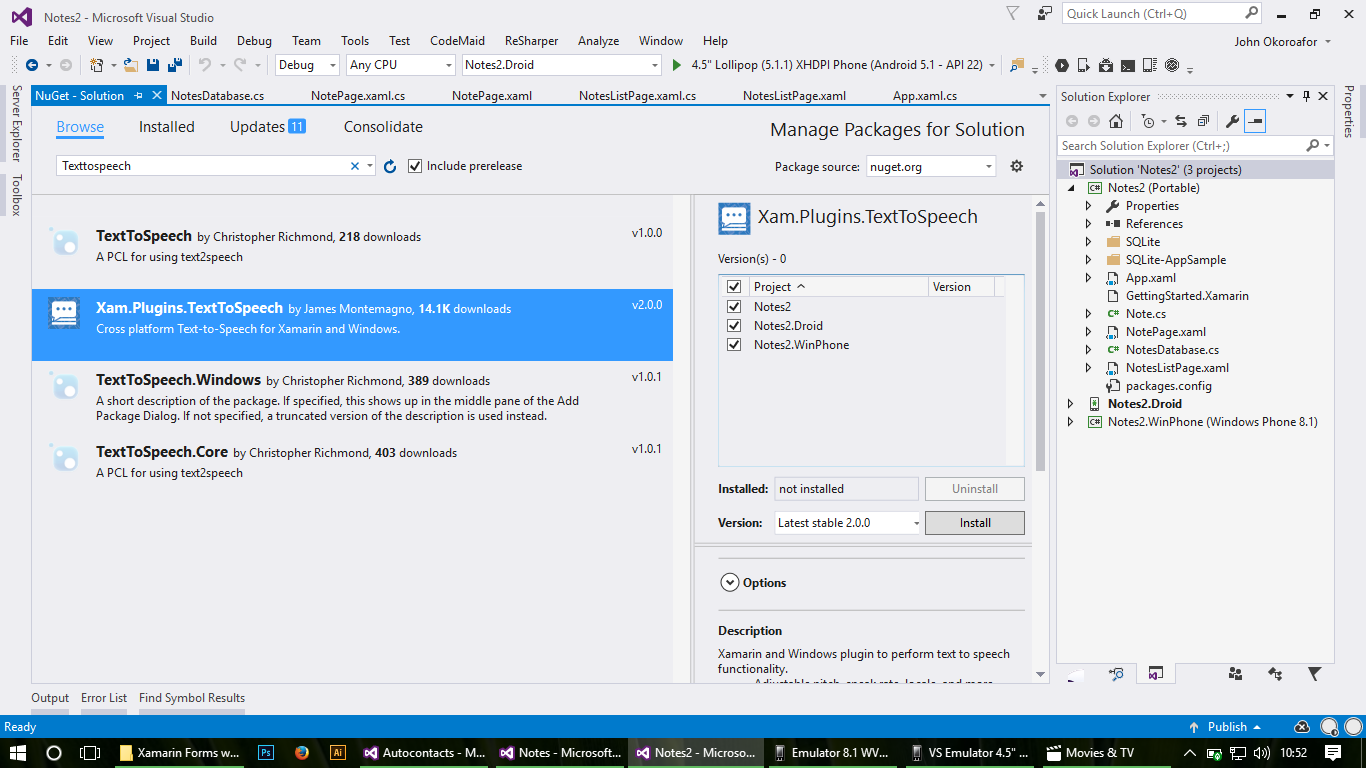
Create a new Xamarin.Forms PCL Project



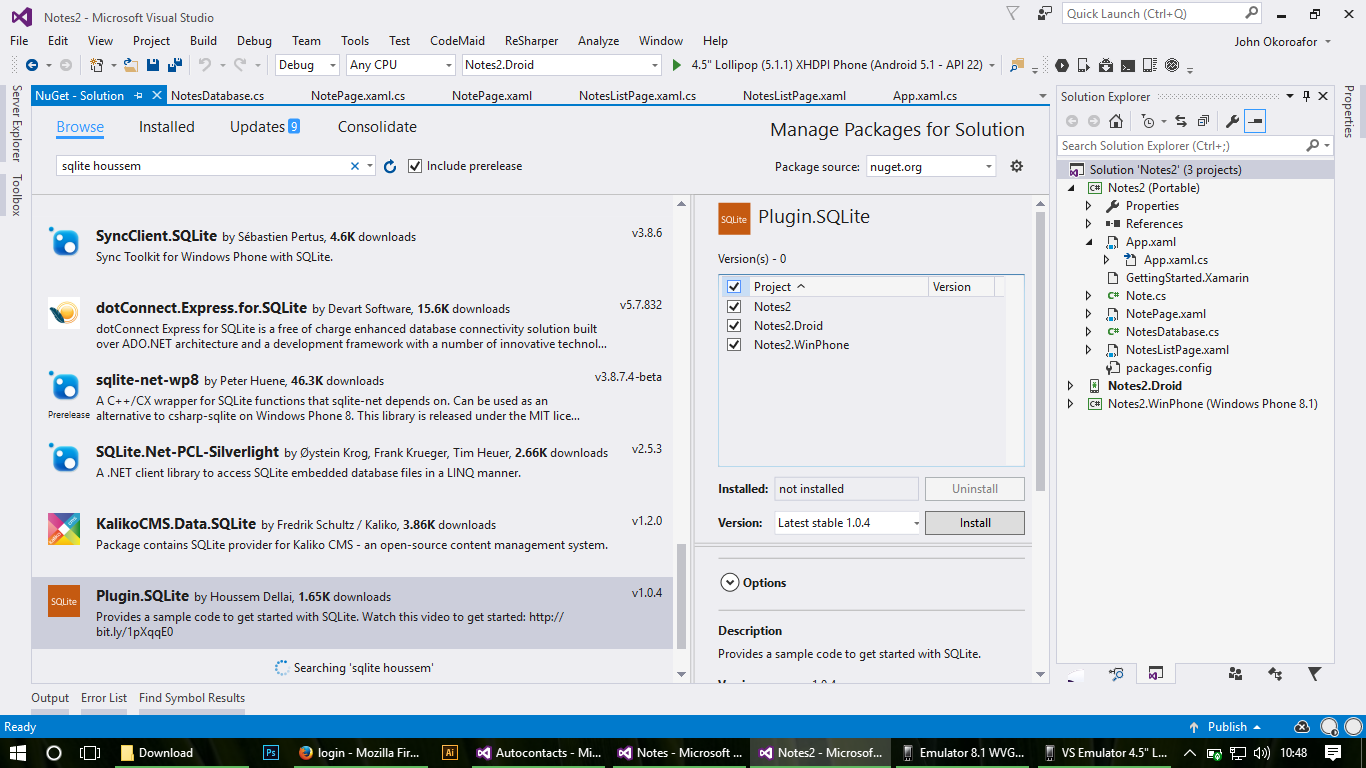
You may delete other projects we won’t be using. First things first, setup our projects and get the necessary plugins. Right click on the Solution and Manage Nuget Packages. In the update tab, update the Xamarin.Forms Nuget package. **ONLY THE XAMARIN.FORMS NUGET PACKAGE.** I can’t be any clear than this.



After that browse and search for the text to speech plugin by James Montemagno, install in all projects. (See Screenshot for details).



when it’s completed, search for Plugin.SQLite by Houssem Dellai and do likewise.



Now that’s done, you may notice extra folders in your projects, no worries, they were trying to Help. In your PCL, delete the folder titled “SQLite-Appsample” and its contents. Setup windows phone for SQLite by opening the windows phone project folder, right click on reference and add reference, under the windows phone 8.1 tab select extensions and tick “Microsoft visual c++ 2013” and OK it.

When that’s done roll up your sleeves and let’s get dirty, ahem, I meant to work. Let’s get to work.

Everything we are going to be doing would be in our PCL folder so no need to tamper with the others. Create a class and call it Note.cs, this would be our Note model class and replace the code.

Import using statements and fix namespace errors.

using SQLite;

namespace Notes

{

public class Note

{

[PrimaryKey, AutoIncrement]

public int Id { get; set; }

public string Name { get; set; }

public string Details { get; set; }

}

}

Now that’s done we need to setup our database. Create a new Class and call it NotesDatabase.cs and replace the code

Import using statements and fix namespace errors.

using SQLite;

using System.Collections.Generic;

using System.Linq;

using Xamarin.Forms;

using XamarinForms.SQLite.SQLite;

namespace Notes

{

public class NotesDatabase

{

private readonly SQLiteConnection database;

public NotesDatabase()

{

database = DependencyService.Get<ISQLite>().GetConnection();

database.CreateTable<Note>();

}

public List<Note> GetItems()

{

return database.Table<Note>().ToList();

}

public Note GetItem(int id)

{

return database.Table<Note>().FirstOrDefault(i => i.Id == id);

}

public int SaveItemAsync(Note item)

{

if (item.Id != 0)

{

return database.Update(item);

}

else

{

return database.Insert(item);

}

}

public int DeleteItemAsync(Note item)

{

return database.Delete(item);

}

}

}

This class will setup and handle our CRUD operations in our database.

When that’s done Open the App.xaml.cs file and make it look like this:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Xamarin.Forms;

using XamarinForms.SQLite;

namespace Notes2

{

public partial class App : Application

{

public static NotesDatabase database;

public App()

{

InitializeComponent();

MainPage = new NavigationPage(new NotesListPage());

}

public int ResumeAtNoteId { get; set; }

public static NotesDatabase Database

{

get

{

if (database == null)

{

database = new NotesDatabase();

}

return database;

}

}

protected override void OnStart()

{

// Handle when your app starts

}

protected override void OnSleep()

{

// Handle when your app sleeps

}

protected override void OnResume()

{

// Handle when your app resumes

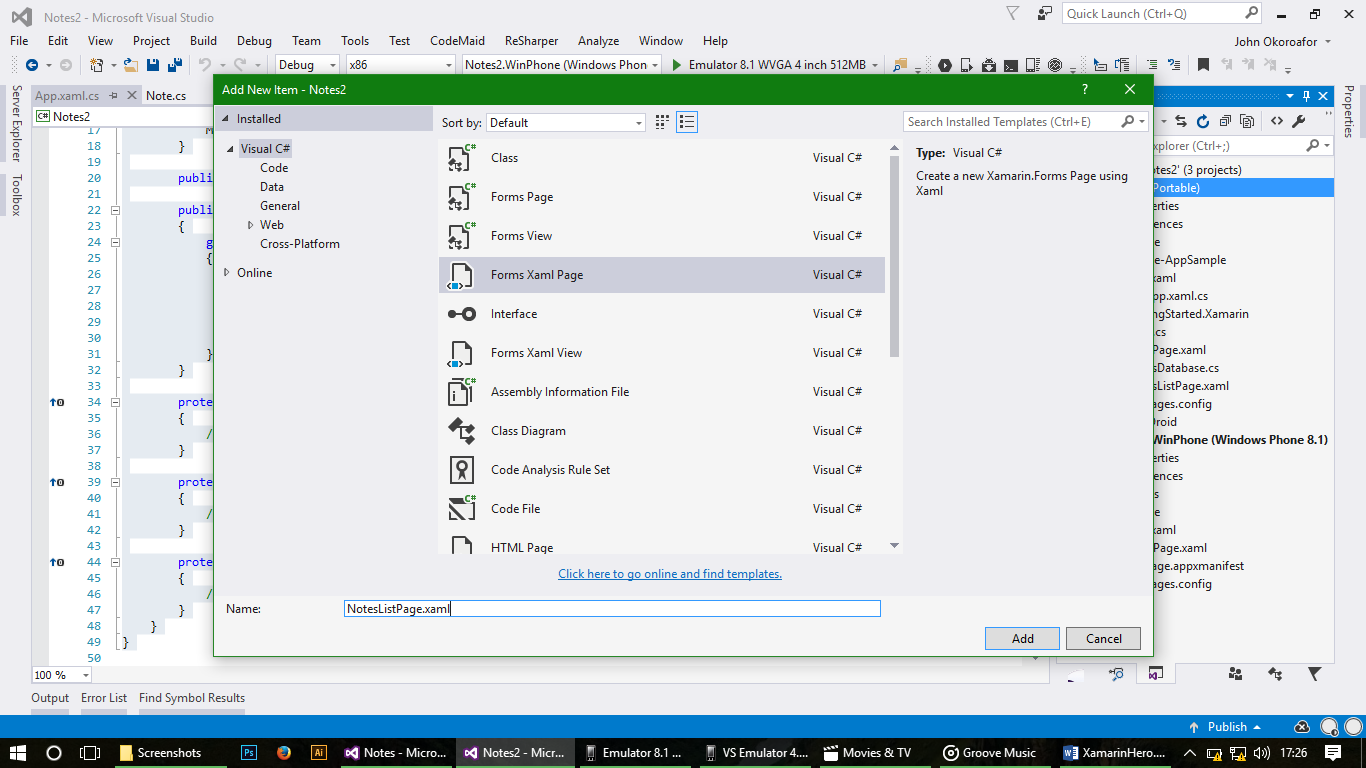
}

}

}

This would create a static instance of our database to utilize throughout our application. You may get an error in the MainPage area. No worries, let’s go create the pages.

Add a new xaml page and name it NotesListPage.xaml. See screenshot

Add the Listview and define its template in your xaml:

<ContentPage.ToolbarItems>

<ToolbarItem Text="+"

Clicked="AddItem">

</ToolbarItem>

</ContentPage.ToolbarItems>

<ListView x:Name="ListView" Margin="10" ItemSelected="OnListItemSelected">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<StackLayout Margin="10,0,0,0"

Orientation="Horizontal"

HorizontalOptions="FillAndExpand">

<Label Text="{Binding Name}"

FontAttributes="Bold"

VerticalTextAlignment="Center"

HorizontalOptions="StartAndExpand" />

</StackLayout>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

Check to fix any impending errors that may arise. We have defined our Listview and given it its data template which comprise a stacklayout and a label to display text.

In the code behind, NotelistPage.xaml.cs adjust the code to look like this:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Xamarin.Forms;

namespace Notes

{

public partial class NotesListPage : ContentPage

{

public NotesListPage()

{

InitializeComponent();

}

protected override void OnAppearing()

{

base.OnAppearing();

// Reset the 'resume' id, since we just want to re-start here

((App)Application.Current).ResumeAtNoteId = -1;

ListView.ItemsSource = App.Database.GetItems();

}

async void AddItem(object sender, EventArgs e)

{

// Navigate to the next page passing the binding context

await Navigation.PushAsync(new NotePage()

{

BindingContext = new Note()

});

}

async void OnListItemSelected(object sender, SelectedItemChangedEventArgs e)

{

var note = (Note) e.SelectedItem;

if (note != null)

((App) Application.Current).ResumeAtNoteId = note.Id;

await Navigation.PushAsync(new NotePage

{

BindingContext = (Note) e.SelectedItem

});

}

}

}

Again remember to fix namespace errors if they come up.

Now we are done with the landing page which shows our lists of notes with a ‘+’ button on the toolbar to add a new note, let’s go ahead and create a page which would serve as a note editor. Create a new xaml page name NotePage.xaml and adjust the code:

<?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="Notes2.NotesListPage"

Title="Notes">

<ContentPage.ToolbarItems>

<ToolbarItem Text="Speak"

Priority="0"

Order="Secondary"

Clicked="OnSpeakClicked" />

<ToolbarItem Text="Delete"

Priority="1"

Order="Secondary"

Clicked="OnDeleteClicked" />

</ContentPage.ToolbarItems>

<StackLayout VerticalOptions="FillAndExpand" HorizontalOptions="FillAndExpand" Padding="10">

<Entry Placeholder="Title"

Text="{Binding Name}"

FontAttributes="Bold"

TextChanged="TextEntered"

VerticalOptions="Start" />

<Editor TextChanged="TextEntered"

Text="{Binding Details}"

VerticalOptions="FillAndExpand" />

</StackLayout>

</ContentPage>

This creates a new page with and entry and an editor control to input text: Title of note and Note Details with a toolbar options menu containing the speak and delete items. Let’s go over to the code behind to add functionality to the page.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Plugin.TextToSpeech;

using Xamarin.Forms;

namespace Notes2

{

public partial class NotePage : ContentPage

{

public NotePage()

{

InitializeComponent();

}

async void OnDeleteClicked(object sender, EventArgs e)

{

var note = (Note)BindingContext;

App.Database.DeleteItemAsync(note);

await Navigation.PopAsync();

}

void OnSpeakClicked(object sender, EventArgs e)

{

var note = (Note)BindingContext;

CrossTextToSpeech.Current.Speak($"{note.Name} {note.Details}");

}

void TextEntered(object sender, EventArgs e)

{

var note = (Note)BindingContext;

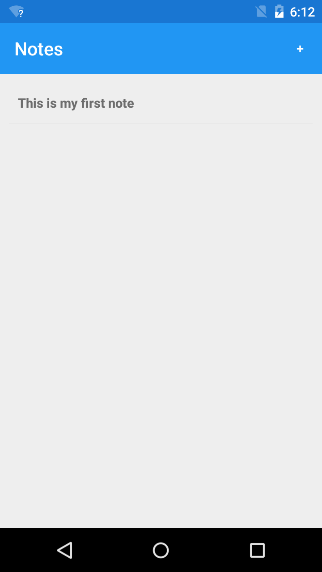
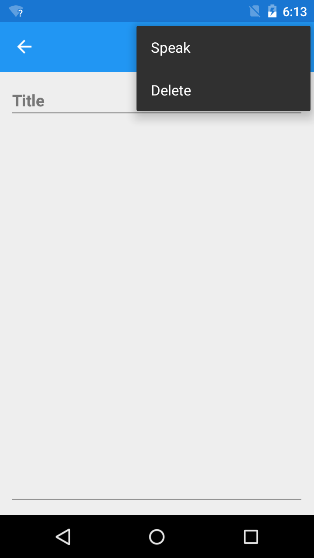
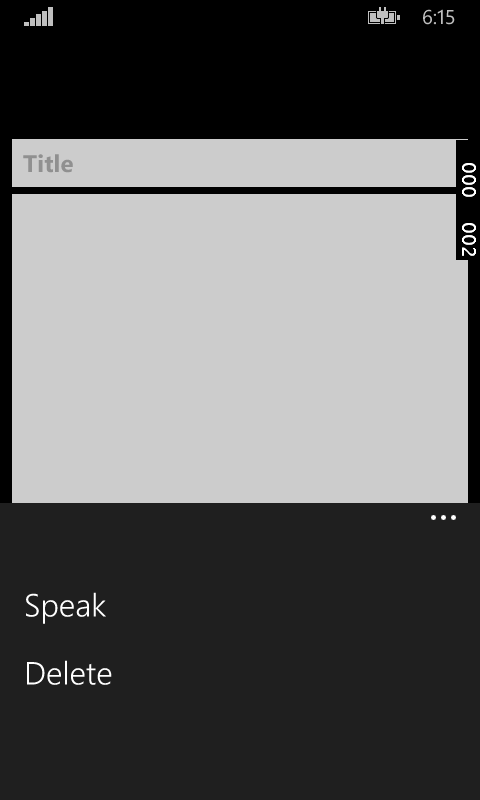
App.Database.SaveItemAsync(note);

}

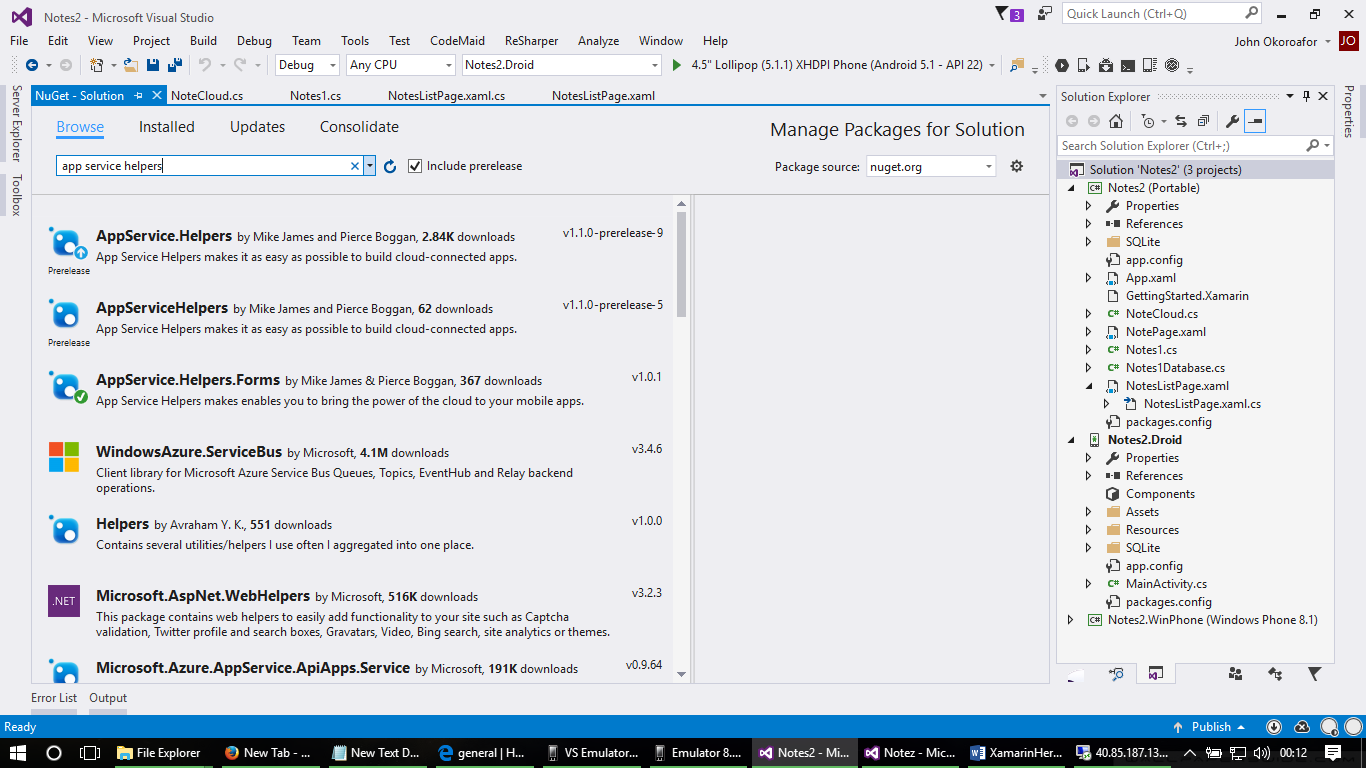
}

}

Now all things working fine and running select your start up project from the toolbar and run. It may take a while on first run, just on first run, so be patient and grab a snack. It should look like this all things going well. If not get help from an instructor.



Now we are done with saving locally, next step is to save to the cloud. Add the following Nuget Packages. Add the app service helper nuget packages. We are going to use the 1st and 3rd options that come up. See screenshot for details:



After That’s done, create a new class called NoteCloud.cs and edit the code:

using AppServiceHelpers.Models;

namespace Notes2

{

public class NoteCloud: EntityData

{

public string Name { get; set; }

public string Details { get; set; }

}

}

After that, we go back to edit our notelistpage.xaml.cs adding the following methods, just below the “ONLISTITEMSELECTED” method. NOTICE THE DIFFERNECE IN CODES.

async void SyncClicked(object sender, EventArgs e)

{

await Sync();

}

async Task Sync()

{

ITableDataStore<NoteCloud> table;

var client = EasyMobileServiceClient.Create();

// 2. Initialize the library with the URL of the Azure Mobile App you created in Step #1.

client.Initialize("http://herotest.azurewebsites.net");

// 3. Register a model with the EasyMobileServiceClient to create a table.

client.RegisterTable<NoteCloud>();

// 4. Finalize the schema for our database. All table registrations must be done before this step.

await client.FinalizeSchema();

table = client.Table<NoteCloud>();

foreach (var item in App.Database.GetItems())

{

await table.AddAsync(new NoteCloud()

{

Name = item.Name,

Details = item.Details

});

}

And in the notelispage edit the xaml to the following notice the differences:

<?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="Notes2.NotesListPage"

Title="NoteTaker">

<ContentPage.ToolbarItems>

<ToolbarItem Text="+"

Clicked="AddItem" />

</ContentPage.ToolbarItems>

<StackLayout>

<ListView x:Name="ListView" ItemSelected="OnListItemSelected">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<StackLayout Orientation="Horizontal"

HorizontalOptions="FillAndExpand">

<Label Text="{Binding Name}"

FontAttributes="Bold"

VerticalTextAlignment="Center"

HorizontalOptions="StartAndExpand" />

</StackLayout>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

<Button Text="Send" Clicked="SyncClicked" />

</StackLayout>

</ContentPage>

After that’s done, run the app, add items and sync, on your Azure Backend easy table, you should see the list being populated.