Xamarin Workshop

Gentle Introduction to XAML and Xamarin

1. Create a new Xamarin app using the Blank template.
2. Comment out all XAML except the first label.
3. Run the Windows app.
4. Modify various elements of the Label
5. Uncomment the other Labels
6. Uncomment the Frame
7. Name the first Label
8. Add a Button under the first label

```
<Button x:Name="Button1"
        Text="Click Me!"
        Clicked="Button1_Clicked" />
```
9. Add code in Clicked to change color to Label1

```
Label1.TextColor = Color.Red;
```

10. Run app.
11. Add width request to button. Nothing happens.

```
WidthRequest="100"
```

12. Add HorizontalOptions to change size of button.

```
HorizontalOptions="Start"
```

13. Run Android version
14. Explore solution
    a. Projects for Android, iOS, UWP
    b. Shared project for code and UI
    c. Xamarin.Forms is NuGet package
    d. All 3 platform projects call Xamarin.Forms

HoneyDo App

Requirements

- Keep track of todo list
- Items have properties:
  - Description
  - Person assigned to (you, partner, child, friend)
  - Priority (high, medium, low)
  - Due date
  - Category (home indoors, home outdoors, errand)
  - Recurring (none, daily, weekly, bi-monthly, monthly, quarterly, annual)
  - Completed
- Main page should display list of outstanding items, sorted by due date and priority, and have option to add an item
- Tap on an item to edit it. Swipe to complete or delete it.
- Main data store is Azure
- Provide local data store for offline usage and syncing when online

Basic Skeleton

Model

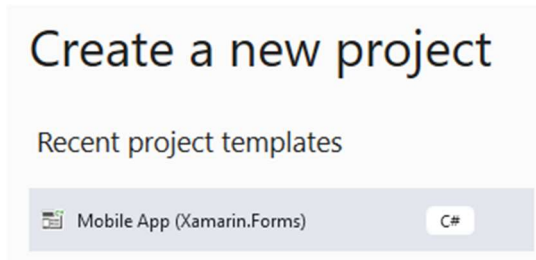- We need a data model class.
- We need a mock data service.

View

- We need an Items page to display items.
- We need an Item detail page to edit items.
- We need a New Item page to add items.

ViewModel

- We need an Items view model.
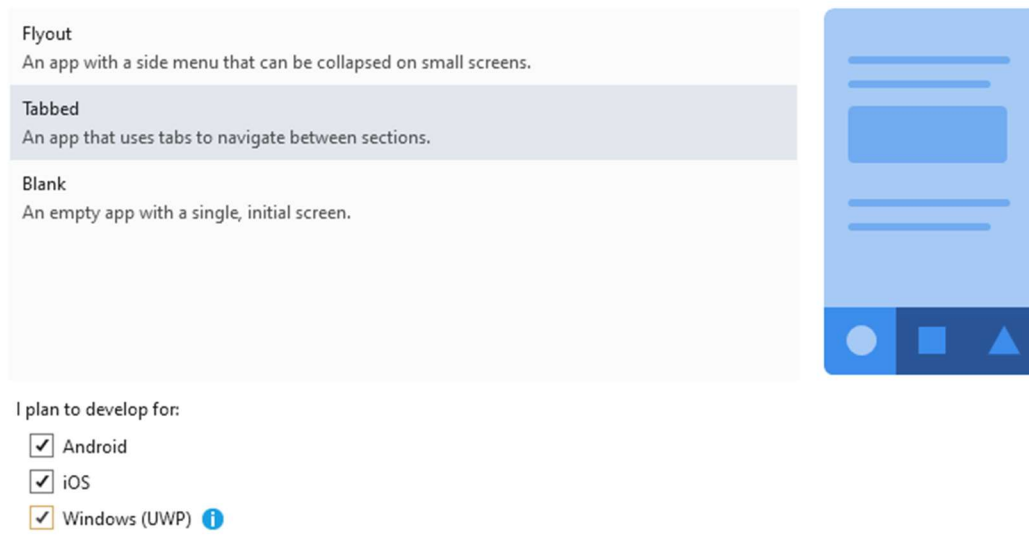- We need an Item view model.

Create App

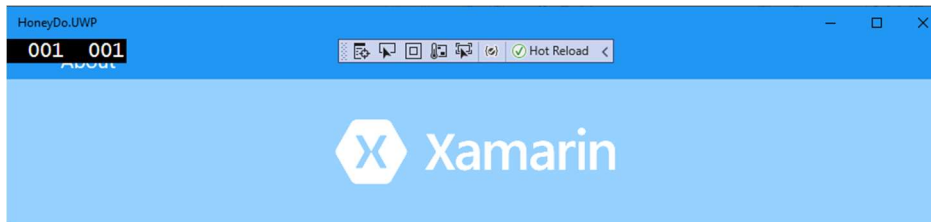1. Create a new project in Visual Studio. Select the Mobile App (Xamarin Forms) project template.

2. Name the project HoneyDo.
3. In the New Mobile App dialog, select the Tabbed template. Check Windows (UWP).

4. Once the project is created, build it.
5. Run the Windows version of the app to see what it looks like.

HoneyDo.UWP

001   001

About

Xamarin

Start developing now

Make changes to your XAML file and save to see your UI update in the running app with XAML Hot Reload. Give it a try!

Learn more at **https://aka.ms/xamarin-quickstart**

Learn more

About   Browse

HoneyDo.UWP

001   000

Browse

Add

First item
This is an item description.

Second item
This is an item description.

Third item
This is an item description.

Fourth item
This is an item description.

Fifth item
This is an item description.

Sixth item
This is an item description.

About   Browse

HoneyDo.UWP

001   000

Text

Description

Cancel   Save
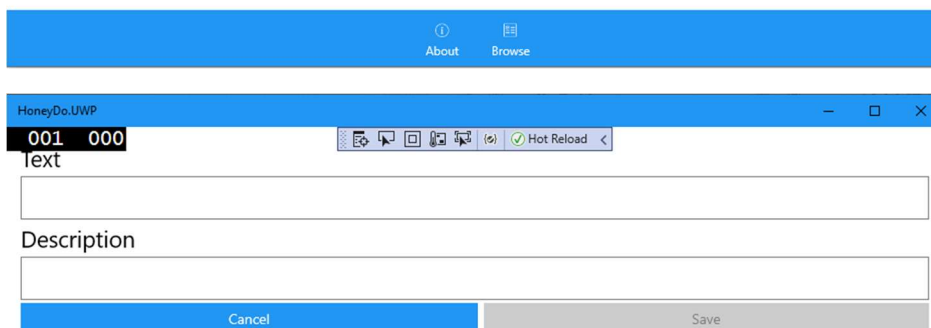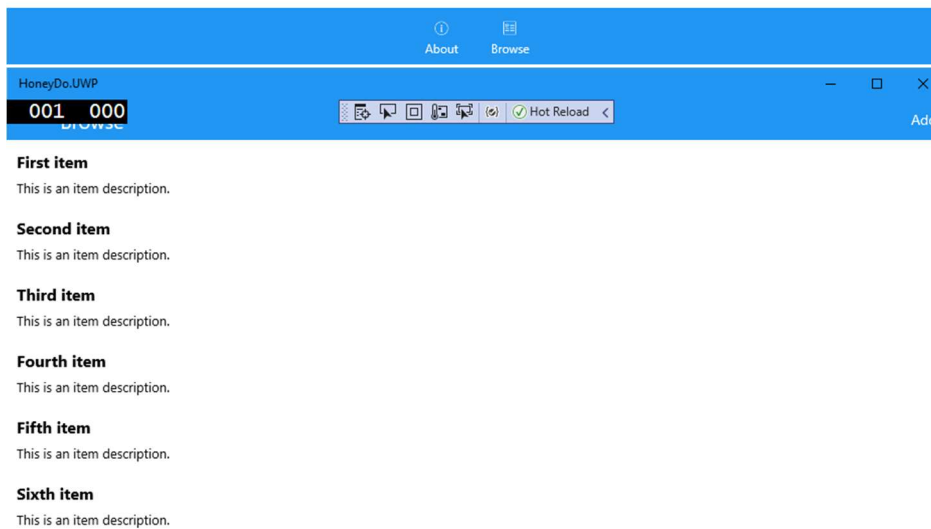
6. Right-click on the Solution and select Manage NuGet Packages for Solution. Notice what version of Xamarin.Forms you are using.

7. Open the Android app's Properties. On the Application tab you can choose what version of the Android SDK you will use to compile the app. Of course, you have to have that SDK installed!
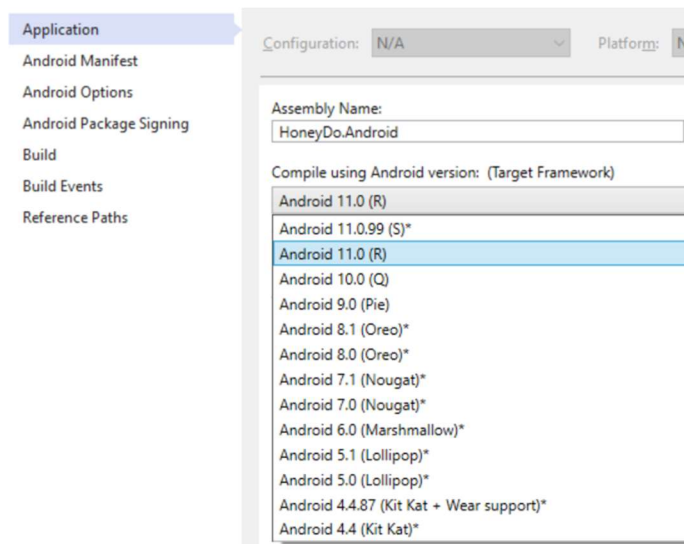
8. In the Android Manifest tab you can choose the minimum Android version your app will support and the version you will target.

Application	Configuration:	N/A	Platform:	N/A
Android Manifest				
Android Options				
Android Package Signing	Version number:			
Build	1			
Build Events	Version name:			
Reference Paths	1.0			

Install location:

Prefer Internal

Minimum Android version:

Android 5.0 (API Level 21 - Lollipop)

Target Android version:

Android 11.0 (API Level 30 - R)

9. The default Tabbed app uses a standard MVVM (Model-View-ViewModel) structure. This is very common for XAML apps. Our HoneyDo app will use the same structure.
10. Delete the contents of the Model, Services, ViewModels and Views folders. Keep the folders.

Add mock data

1. In the Models folder, add a HoneyDoItem class.

```
public class HoneyDoItem
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public string AssignedTo { get; set; }
    public string Priority { get; set; }
    public string Category { get; set; }
    public DateTime DueDate { get; set; }
    public string Recurrence { get; set; }
    public bool Completed { get; set; }
}
```

2. In the Services folder, add an IDataStore interface.

```
public interface IDataStore<T>
{
    void Initialize();
    Task<List<T>> GetItemsAsync();
    Task<T> GetItemAsync(int id);
    Task<bool> AddItemAsync(T item);
    Task<bool> UpdateItemAsync(T item);
    Task<bool> DeleteItemAsync(int id);
}
```

3. Replace the using statements with the following:

```
using System.Collections.Generic;
using System.Threading.Tasks;
```

4. Add a MockDataStore class

```
public class MockDataStore : IDataStore<HoneyDoItem>
{
```

```csharp
List<HoneyDoItem> honeyDoItems;

public void Initialize()
{
    honeyDoItems = new List<HoneyDoItem>()
    {
        new HoneyDoItem
        {
            Id = 1,
            Description = "Clean litter box",
            AssignedTo="Me",
            Priority="High",
            Recurrence="Monthly",
            Category="Home indoors",
            DueDate=DateTime.Today.AddDays(-2)
        },
        new HoneyDoItem
        {
            Id = 2,
            Description = "Grocery shop",
            AssignedTo="You",
            Priority="High",
            Recurrence="Weekly",
            Category="Errand",
            DueDate=DateTime.Today.AddDays(2)
        },
        new HoneyDoItem
        {
            Id = 3,
            Description = "Book vacation travel",
            AssignedTo="You",
            Priority="Low",
            Recurrence="None",
            Category="Home indoors",
            DueDate=DateTime.Today.AddDays(2)
        },
        new HoneyDoItem
        {
            Id = 4,
            Description = "Mow the lawn",
            AssignedTo="Me",
            Priority="Medium",
            Recurrence="Weekly",
            Category="Home outdoors",
            DueDate=DateTime.Today.AddDays(5)
        },
        new HoneyDoItem
        {
            Id = 5,
            Description = "Dust and vacuum",
            AssignedTo="Us",
            Priority="Medium",
            Recurrence="Monthly",
            Category="Home indoors",
            DueDate=DateTime.Today.AddDays(7)
        }
    };
}

public async Task<List<HoneyDoItem>> GetItemsAsync()
{
    return await Task.FromResult(honeyDoItems);
}

public async Task<HoneyDoItem> GetItemAsync(int id)
{
    return await Task.FromResult(honeyDoItems.FirstOrDefault(s => s.Id == id));
}
```

```csharp
    public async Task<Boolean> AddItemAsync(HoneyDoItem honeyDoItem)
    {
        honeyDoItems.Add(honeyDoItem);

        return await Task.FromResult(true);
    }

    public async Task<Boolean> UpdateItemAsync(HoneyDoItem honeyDoItem)
    {
        var oldHoneyDoItem = honeyDoItems.Where(
            (HoneyDoItem arg) => arg.Id == honeyDoItem.Id).FirstOrDefault();
        honeyDoItems.Remove(oldHoneyDoItem);
        honeyDoItems.Add(honeyDoItem);

        return await Task.FromResult(true);
    }

    public async Task<Boolean> DeleteItemAsync(int id)
    {
        var oldHoneyDoItem = honeyDoItems.Where(
            (HoneyDoItem arg) => arg.Id == id).FirstOrDefault();
        honeyDoItems.Remove(oldHoneyDoItem);

        return await Task.FromResult(true);
    }
}
```

5. Replace the using statements with the following:

```csharp
using HoneyDo.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

Add ability to view a list of honey do items

1. In the ViewModels folder, add a BaseViewModel class

```csharp
public class BaseViewModel : INotifyPropertyChanged
{
    public IDataStore<HoneyDoItem> DataStore =>
        DependencyService.Get<IDataStore<HoneyDoItem>>();

    bool isBusy = false;
    public bool IsBusy
    {
        get { return isBusy; }
        set { SetProperty(ref isBusy, value); }
    }

    string title = string.Empty;
    public string Title
    {
        get { return title; }
        set { SetProperty(ref title, value); }
    }

    protected bool SetProperty<T>(ref T backingStore, T value,
        [CallerMemberName] string propertyName = "",
        Action onChanged = null)
    {
        if (EqualityComparer<T>.Default.Equals(backingStore, value))
            return false;

        backingStore = value;
```

```csharp
            onChanged?.Invoke();
            OnPropertyChanged(propertyName);
            return true;
        }

        #region INotifyPropertyChanged
        public event PropertyChangedEventHandler PropertyChanged;
        protected void OnPropertyChanged([CallerMemberName] string propertyName = "")
        {
            var changed = PropertyChanged;
            if (changed == null)
                return;

            changed.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
        #endregion
    }
```

2. Replace the using statements with the following:

```csharp
using HoneyDo.Models;
using HoneyDo.Services;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using Xamarin.Forms;
```

3. Add a HoneyDoItemsViewModel class.

```csharp
public class HoneyDoItemsViewModel : BaseViewModel
{
    public ObservableCollection<HoneyDoItem> HoneyDoItems { get; set; }

    public Command LoadItemsCommand { get; set; }

    public HoneyDoItemsViewModel()
    {
        HoneyDoItems = new ObservableCollection<HoneyDoItem>();

        DataStore.Initialize();

        LoadItemsCommand = new Command(async () => await ExecuteLoadItemsCommand());
    }

    async Task ExecuteLoadItemsCommand()
    {
        IsBusy = true;

        try
        {
            HoneyDoItems.Clear();
            var honeyDoItems = await DataStore.GetItemsAsync();
            foreach (var item in honeyDoItems)
            {
                HoneyDoItems.Add(item);
            }
        }
        catch (Exception ex)
        {
            Debug.WriteLine(ex);
        }
        finally
        {
            IsBusy = false;
        }
    }
```

```
        public void OnAppearing()
        {
            IsBusy = true;
        }
    }
```
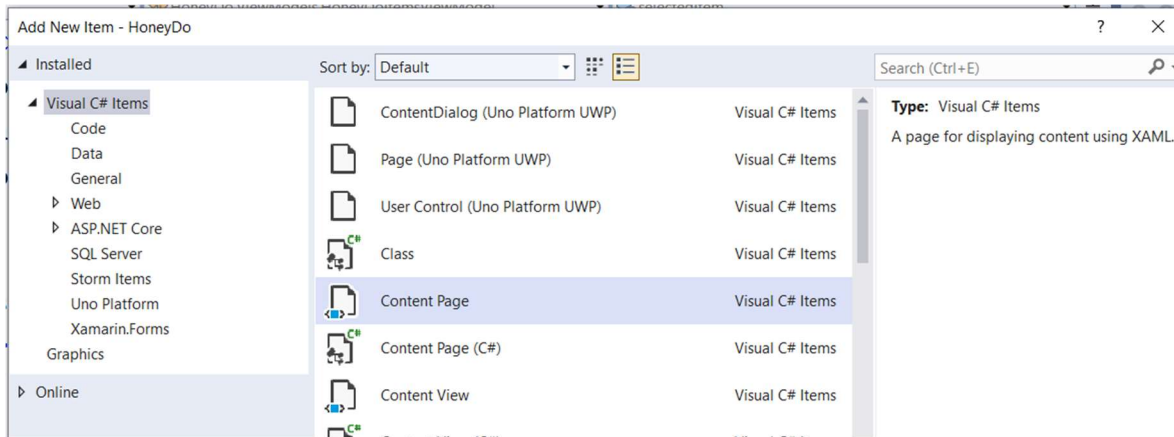
4. Replace the using statements with the following:

```
using HoneyDo.Models;
using System;
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.Threading.Tasks;
using Xamarin.Forms;
```

5. In the Views folder, add a HoneyDoItemsPage.xaml using the Content Page item template.

NOTE: This is a good time to review XAML and how it works.

6. Replace the page's XAML with the following.

```
<?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="HoneyDo.Views.HoneyDoItemsPage"
             Title="Honey Do Items"
             xmlns:local="clr-namespace:HoneyDo.ViewModels"
             xmlns:model="clr-namespace:HoneyDo.Models">

    <RefreshView x:DataType="local:HoneyDoItemsViewModel"
                 Command="{Binding LoadItemsCommand}"
                 IsRefreshing="{Binding IsBusy, Mode=TwoWay}">
        <CollectionView x:Name="HoneyDoItemsCollectionView"
                        ItemsSource="{Binding HoneyDoItems}"
                        SelectionMode="None">
            <CollectionView.ItemTemplate>
                <DataTemplate>
                    <StackLayout Padding="10"
                                 x:DataType="model:HoneyDoItem">
                        <Label Text="{Binding Description}"
                               LineBreakMode="NoWrap"
                               Style="{DynamicResource ListItemTextStyle}"
                               FontSize="16" />
                        <Label Text="{Binding AssignedTo}"
                               LineBreakMode="NoWrap"
                               Style="{DynamicResource ListItemDetailTextStyle}"
                               FontSize="14" />
                        <StackLayout Orientation="Horizontal">
                            <Label Text="{Binding Priority}"
                                   LineBreakMode="NoWrap"
                                   Style="{DynamicResource ListItemDetailTextStyle}"
                                   FontSize="14" />
                            <Label Text="priority"
                                   LineBreakMode="NoWrap"
```

```
                                        Style="{DynamicResource ListItemDetailTextStyle}"
                                        FontSize="14" />
                            </StackLayout>
                            <StackLayout Orientation="Horizontal">
                                <Label Text="Due "
                                        LineBreakMode="NoWrap"
                                        Style="{DynamicResource ListItemDetailTextStyle}"
                                        FontSize="14" />
                                <Label Text="{Binding DueDate, StringFormat='{0:MM/dd/yy}'}"
                                        LineBreakMode="NoWrap"
                                        Style="{DynamicResource ListItemDetailTextStyle}"
                                        FontSize="14" />
                            </StackLayout>
                        </StackLayout>
                    </DataTemplate>
                </CollectionView.ItemTemplate>
            </CollectionView>
        </RefreshView>

    </ContentPage>
```

7. Add the following code in bold to HoneyDoItemsPage.xaml.cs

```
HoneyDoItemsViewModel viewModel;

public HoneyDoItemsPage()
{
    InitializeComponent();

    BindingContext = viewModel = new HoneyDoItemsViewModel();
}

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

8. Replace the using statements with the following.

```
using HoneyDo.ViewModels;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
```

9. Notice the following code in App.xaml.cs

```
DependencyService.Register<MockDataStore>();
```

10. In AppShell.xaml, comment out the first ShellContent element in the first TabBar element at the bottom of the page.

11. Comment out the second TabBar element.

12. Make the following change in bold to the second ShellContent element.

```
<ShellContent Title="Browse"
              Icon="icon_feed.png"
              ContentTemplate="{DataTemplate local:HoneyDoItemsPage}" />
```

Your XAML should look like this

```
        </ResourceDictionary>
    </Shell.Resources>

    <TabBar>
        <!--<ShellContent Title="About"
                      Icon="icon_about.png"
                      Route="AboutPage"
                      ContentTemplate="{DataTemplate local:AboutPage}" />-->
        <ShellContent Title="Browse"
                      Icon="icon_feed.png"
                      ContentTemplate="{DataTemplate local:HoneyDoItemsPage}" />
    </TabBar>

    <!--
        If you would like to navigate to this content you can do so by calling
        await Shell.Current.GoToAsync("//LoginPage");
    -->
    <!--<TabBar>
        <ShellContent Route="LoginPage"
                      ContentTemplate="{DataTemplate local:LoginPage}" />
    </TabBar>-->

</Shell>
```
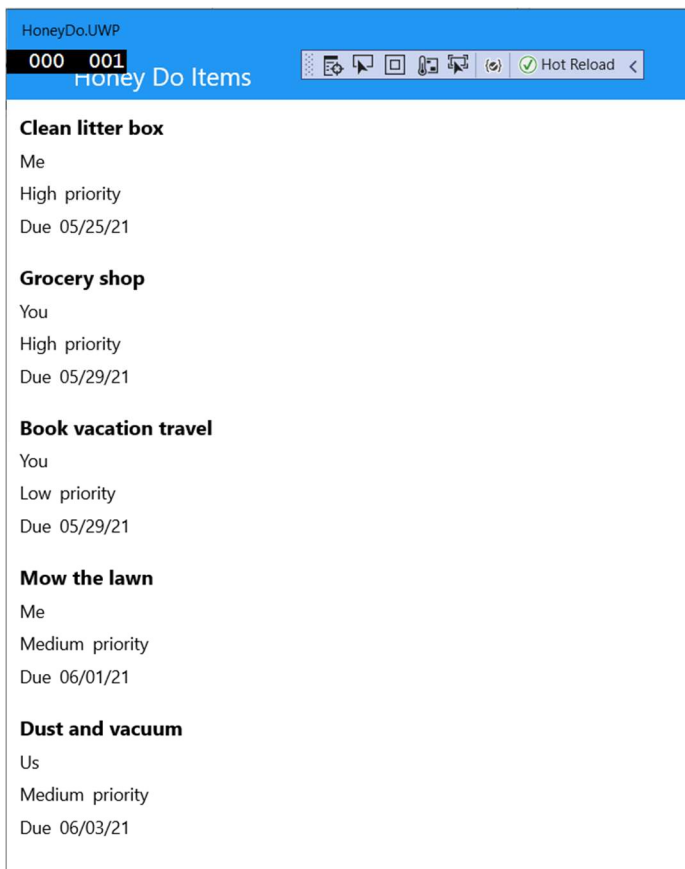
13. In AppShell.xaml.cs, comment out the two Routing.Register lines of code.

```
    3 references
    public partial class AppShell : Xamarin.Forms.Shell
    {
        1 reference
        public AppShell()
        {
            InitializeComponent();
            //Routing.RegisterRoute(nameof(ItemDetailPage), typeof(ItemDetailPage));
            //Routing.RegisterRoute(nameof(NewItemPage), typeof(NewItemPage));
        }
    }
```

14. Run the Windows app. You should see five honey do items.

HoneyDo.UWP

000 001
Honey Do Items

Clean litter box
Me
High priority
Due 05/25/21

Grocery shop
You
High priority
Due 05/29/21

Book vacation travel
You
Low priority
Due 05/29/21

Mow the lawn
Me
Medium priority
Due 06/01/21

Dust and vacuum
Us
Medium priority
Due 06/03/21

NOTE: If you want to hide the frame rate counters in the black box, comment out line 45 in App.xaml.cs in the Windows project.

15. Run the Android app using the emulator. A best practice would be to use an emulator that matches the compile target you are using. Download new emulators using the Android Device Manager.
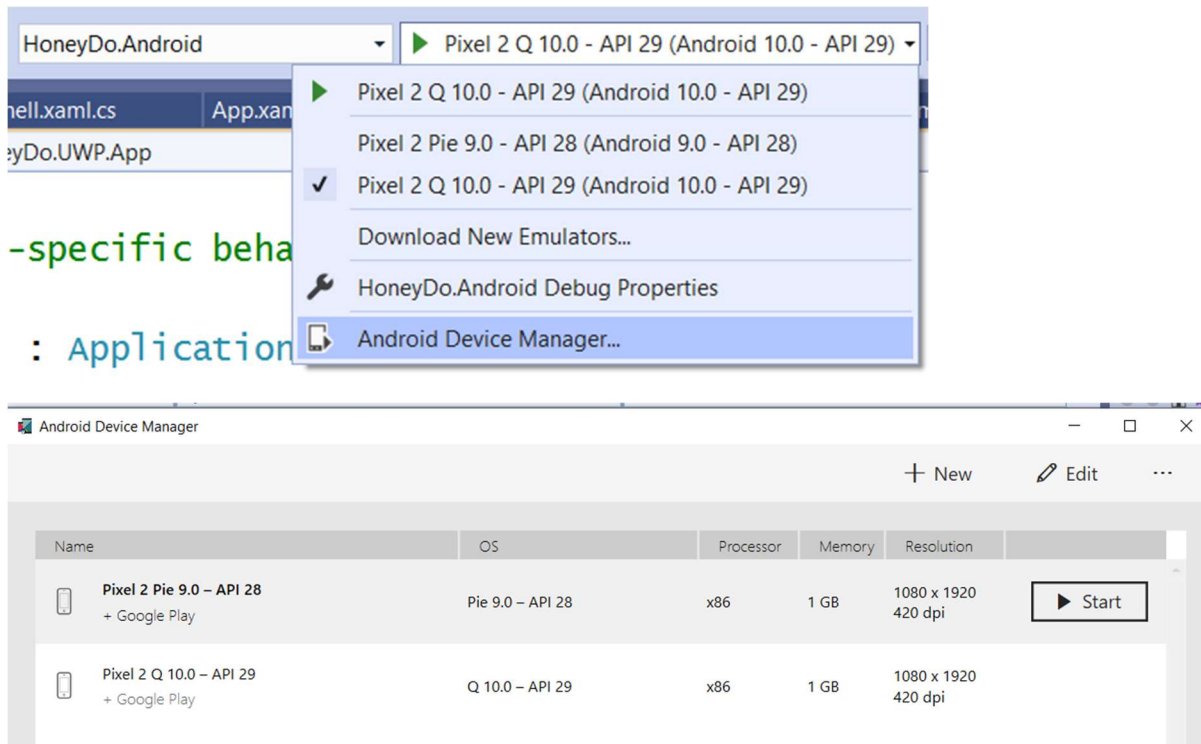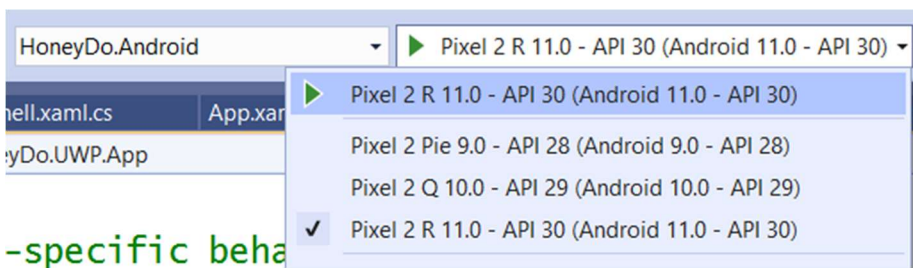

16. Click New to create a new emulator.
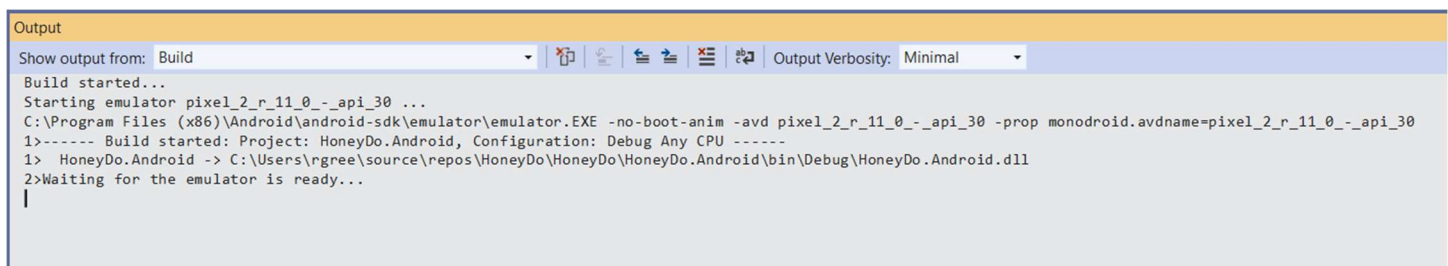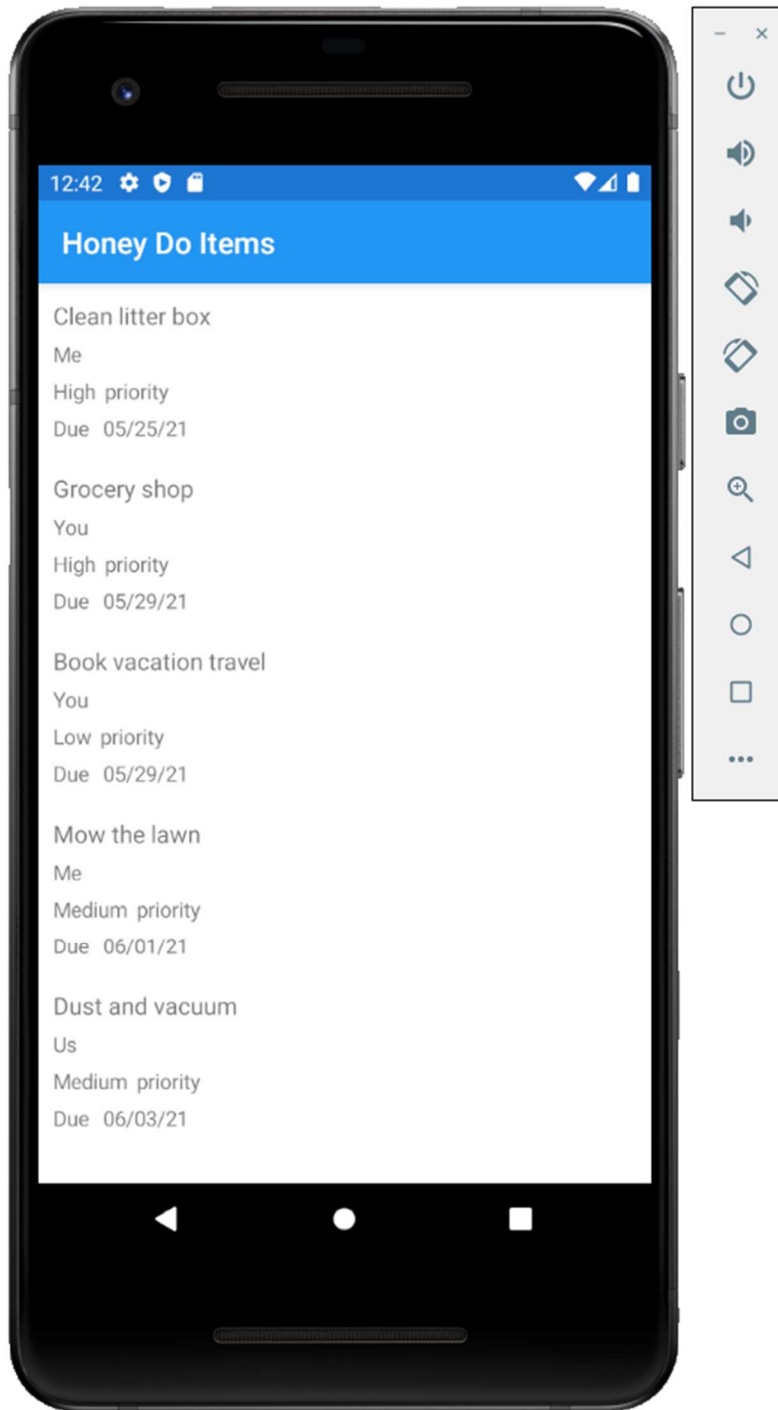17. Select the Android version from the OS drop-down.

18. Click Create.
19. In Visual Studio you can now select that emulator and run the app.

NOTE: It may take some time for the emulator to start the first time you use it.

Add ability to view and edit one honey do item

1. In the ViewModels folder, add a HoneyDoItemViewModel class.

```
public class HoneyDoItemViewModel : BaseViewModel
{
    public HoneyDoItem HoneyDoItem { get; set; }
    public Command SaveItemCommand { get; set; }
    public Command DeleteItemCommand { get; set; }

    public HoneyDoItemViewModel(HoneyDoItem honeyDoItem = null)
```

```
        {
            HoneyDoItem = honeyDoItem;

            SaveItemCommand = new Command(async () => await ExecuteSaveItemCommand());
            DeleteItemCommand = new Command(async () => await ExecuteDeleteItemCommand());
        }

        async Task ExecuteSaveItemCommand()
        {
            IsBusy = true;

            try
            {
                if (HoneyDoItem.Id == 0)
                {
                    await DataStore.AddItemAsync(HoneyDoItem);
                }
                else
                {
                    await DataStore.UpdateItemAsync(HoneyDoItem);
                }
            }
            catch (Exception ex)
            {
                Debug.WriteLine(ex);
            }
            finally
            {
                IsBusy = false;
            }

            await Shell.Current.GoToAsync("..");
        }

        public async Task ExecuteDeleteItemCommand()
        {
            IsBusy = true;

            try
            {
                await DataStore.DeleteItemAsync(HoneyDoItem.Id);
            }
            catch (Exception ex)
            {
                Debug.WriteLine(ex);
            }
            finally
            {
                IsBusy = false;
            }

            await Shell.Current.GoToAsync("..");
        }
    }
```

2. Replace the using statements with the following:

```
using HoneyDo.Models;
using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Xamarin.Forms;
```

3. In the HoneyDo project, in App.xaml.cs, add the following code.

```
public static HoneyDoItemViewModel SelectedItemViewModel;
```

4. Replace the using statements with the following:

```
using HoneyDo.Services;
using HoneyDo.ViewModels;
using Xamarin.Forms;
```

5. In the Views folder, add a HoneyDoItemPage.xaml.

6. Replace the existing StackLayout with the following:

```
<StackLayout Spacing="20"
             Padding="15">

    <Label Text="Name"
           FontSize="Medium" />
    <Entry Text="{Binding HoneyDoItem.Description}"
           FontSize="Large"
           FontAttributes="Bold"
           Margin="0,-10,0,0" />

    <Label Text="Assigned To"
           FontSize="Medium" />
    <Entry Text="{Binding HoneyDoItem.AssignedTo}"
           FontSize="Medium"
           Margin="0,-10,0,0" />

    <Label Text="Priority"
           FontSize="Medium" />
    <Entry Text="{Binding HoneyDoItem.Priority}"
           FontSize="Medium"
           Margin="0,-10,0,0" />

    <Label Text="Category"
           FontSize="Medium" />
    <Entry Text="{Binding HoneyDoItem.Category}"
           FontSize="Medium"
           Margin="0,-10,0,0" />

    <Label Text="Due"
           FontSize="Medium" />
    <DatePicker HorizontalOptions="Start"
                Date="{Binding HoneyDoItem.DueDate}"
                Margin="0,-10,0,0" />

    <StackLayout Orientation="Horizontal">
        <Button x:Name="SaveButton"
                Text="Save"
                HorizontalOptions="Start"
                Margin="25,0,0,25"
                WidthRequest="150"
                HeightRequest="50"
                Command="{Binding SaveItemCommand}" />
        <Button x:Name="CompletedButton"
                Text="Completed"
                HorizontalOptions="Start"
                Margin="25,0,0,25"
                WidthRequest="150"
                HeightRequest="50"
                Command="{Binding DeleteItemCommand}" />
    </StackLayout>

</StackLayout>
```

7. Add the following code in bold to HoneyDoItemPage.xaml.cs

```
HoneyDoItemViewModel viewModel;

public HoneyDoItemPage()
```

```
{
    InitializeComponent();

    BindingContext = viewModel = App.SelectedItemViewModel;
}
```

8. Replace the using statements with the following

```
using HoneyDo.ViewModels;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
```

9. Add the following code in bold to HoneyDoItemsViewModel.cs

```
public Command LoadItemsCommand { get; set; }
public Command<HoneyDoItem> ItemTapped { get; }

public HoneyDoItemsViewModel()
{
    HoneyDoItems = new ObservableCollection<HoneyDoItem>();

    DataStore.Initialize();

    LoadItemsCommand = new Command(async () => await ExecuteLoadItemsCommand());
    ItemTapped = new Command<HoneyDoItem>(OnItemSelected);
}
```

10. Replace the OnAppearing method with the following code:

```
public void OnAppearing()
{
    IsBusy = true;
    SelectedItem = null;
}

public HoneyDoItem SelectedItem
{
    get => selectedItem;
    set
    {
        SetProperty(ref selectedItem, value);
        OnItemSelected(value);
    }
}

async void OnItemSelected(HoneyDoItem item)
{
    if (item == null)
        return;

    App.SelectedItemViewModel = new HoneyDoItemViewModel();
    App.SelectedItemViewModel.HoneyDoItem = item;
    // This will push the HoneyDoItemPage onto the navigation stack
    await Shell.Current.GoToAsync($"{nameof(HoneyDoItemPage)}");
}
```

11. Replace the using statements with the following.

```
using HoneyDo.Models;
using HoneyDo.Views;
using System;
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.Threading.Tasks;
using Xamarin.Forms;
```

12. Add the following code in bold to HoneyDoItemsPage.xaml:

```
    </StackLayout>
    <StackLayout.GestureRecognizers>
        <TapGestureRecognizer NumberOfTapsRequired="1"
                              Command="{Binding Source={RelativeSource
                              AncestorType={x:Type local:HoneyDoItemsViewModel}}, Path=ItemTappe
d}"
                              CommandParameter="{Binding .}">
        </TapGestureRecognizer>
    </StackLayout.GestureRecognizers>
</StackLayout>
```
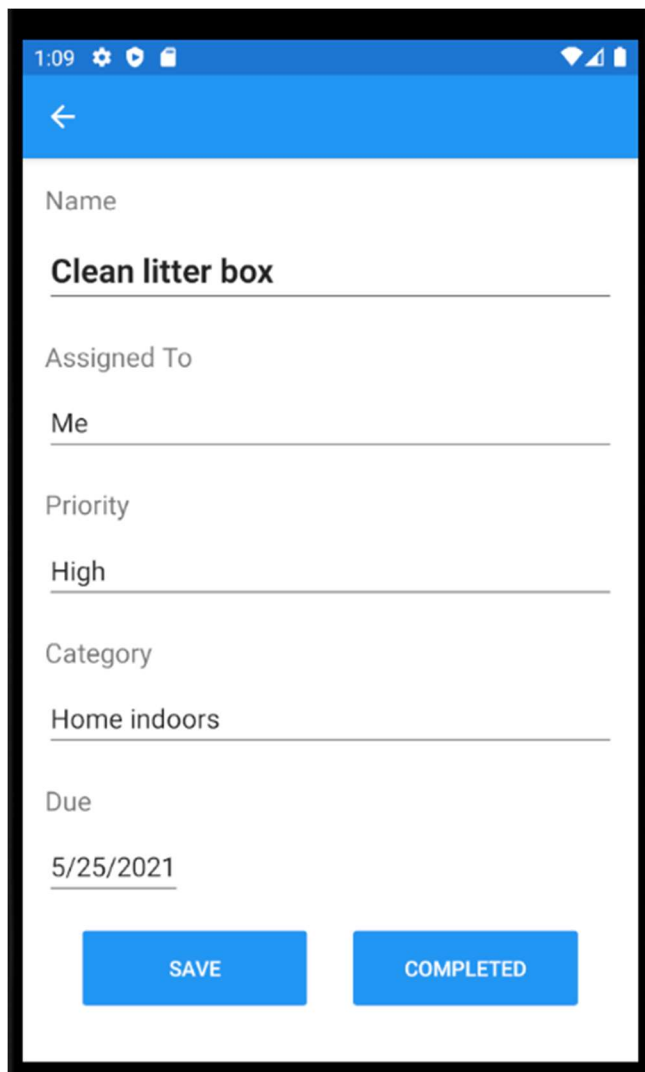
13. In AppShell.xaml.cs, replace the commented out Routing.RegisterRoute code with the following.

```
Routing.RegisterRoute(nameof(HoneyDoItemPage), typeof(HoneyDoItemPage));
```

14. Replace the using statements with the following.

```
using HoneyDo.Views;
using Xamarin.Forms;
```

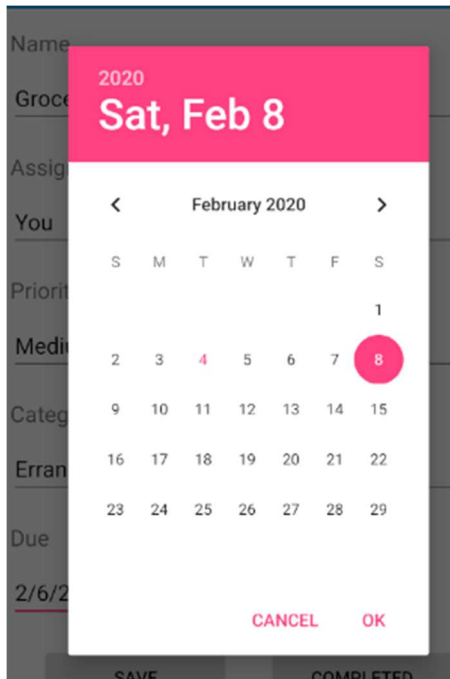15. Run the app. Select one of the honey do items.

16. Change the priority and then change the due date using the date picker.

17. Click Save. The app returns to the main page.

NOTE: The current version of the app is saved in GitHub at https://github.com/rogreen/Xamarin-Workshop-App-V1.

Add pickers for assigned to, priority and category

1. In HoneyDoItemPage.xaml, replace the Entry element for Assigned to with the following. Leave the Label element there.

```xml
<Picker x:Name="AssignedToPicker"
        Title="Assigned to"
        Margin="0,-10,0,0"
        SelectedIndexChanged="OnAssignedToPickerSelectedIndexChanged">
    <Picker.Items>
        <x:String>Me</x:String>
        <x:String>You</x:String>
        <x:String>Us</x:String>
        <x:String>Nobody</x:String>
    </Picker.Items>
</Picker>
```

2. Replace the Entry element for Priority with the following.

```xml
<Picker x:Name="PriorityPicker"
        Title="Priority"
        Margin="0,-10,0,0"
        SelectedIndexChanged="OnPriorityPickerSelectedIndexChanged">
    <Picker.Items>
        <x:String>Low</x:String>
        <x:String>Medium</x:String>
        <x:String>High</x:String>
    </Picker.Items>
</Picker>
```

3. Replace the Entry element for Category with the following.

20

```xml
<Picker x:Name="CategoryPicker"
        Title="Category"
        Margin="0,-10,0,0"
        SelectedIndexChanged="OnCategoryPickerSelectedIndexChanged">
    <Picker.Items>
        <x:String>Home indoors</x:String>
        <x:String>Home outdoors</x:String>
        <x:String>Errand</x:String>
    </Picker.Items>
</Picker>
```

4. Add the following method to HoneyDoItemPage.xaml.cs.

```csharp
private void SetPickers()
{
    switch (viewModel.HoneyDoItem.AssignedTo)
    {
        case "Me":
            AssignedToPicker.SelectedIndex = 0;
            break;
        case "You":
            AssignedToPicker.SelectedIndex  = 1;
            break;
        case "Us":
            AssignedToPicker.SelectedIndex  = 2;
            break;
        case "Nobody":
            AssignedToPicker.SelectedIndex  = 3;
            break;
        default:
            break;
    }

    switch (viewModel.HoneyDoItem.Priority)
    {
        case "Low":
            PriorityPicker.SelectedIndex = 0;
            break;
        case "Medium":
            PriorityPicker.SelectedIndex = 1;
            break;
        case "High":
            PriorityPicker.SelectedIndex = 2;
            break;
        default:
            break;
    }

    switch (viewModel.HoneyDoItem.Category)
    {
        case "Home indoors":
            CategoryPicker.SelectedIndex = 0;
            break;
        case "Home outdoors":
            CategoryPicker.SelectedIndex = 1;
            break;
        case "Errand":
            CategoryPicker.SelectedIndex = 2;
            break;
        default:
            break;
    }
}
```

5. Add the following method to set the pickers when the page appears.

```csharp
protected override void OnAppearing()
{
```

```
        base.OnAppearing();
        SetPickers();
}
```

6. Add the following code to update the honey do item after a selection is made from a picker.

```
private void OnAssignedToPickerSelectedIndexChanged(Object sender, EventArgs e)
{
    viewModel.HoneyDoItem.AssignedTo = ((Picker)sender).SelectedItem.ToString();
}

private void OnPriorityPickerSelectedIndexChanged(Object sender, EventArgs e)
{
    viewModel.HoneyDoItem.Priority = ((Picker)sender).SelectedItem.ToString();
}

private void OnCategoryPickerSelectedIndexChanged(Object sender, EventArgs e)
{
    viewModel.HoneyDoItem.Category = ((Picker)sender).SelectedItem.ToString();
}
```
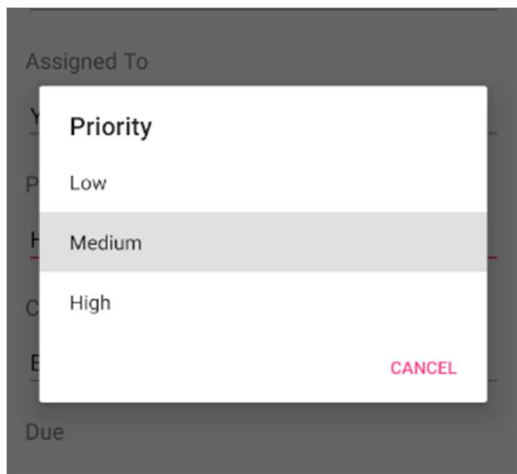
7. Add the following to the using statements:

```
using System;
```

8. Run the app. Select one of the honey do items. Change the priority using the picker. Click Save.

Add the ability to create a new honey do item.

1. Add the following code in bold to HoneyDoItemsViewModel.cs

```
public Command<HoneyDoItem> ItemTapped { get; }
public Command AddItemCommand { get; }

public HoneyDoItemsViewModel()
{
    HoneyDoItems = new ObservableCollection<HoneyDoItem>();

    DataStore.Initialize();

    LoadItemsCommand = new Command(async () => await ExecuteLoadItemsCommand());
    ItemTapped = new Command<HoneyDoItem>(OnItemSelected);
    AddItemCommand = new Command(OnAddItem);
}
```

2. Add the following method.

22

```
private async void OnAddItem(object obj)
{
    var newHoneyDoItem = new HoneyDoItem()
    {
        AssignedTo = "Nobody",
        Priority = "Medium",
        Category = "Errand",
        DueDate = DateTime.Today.AddDays(7)
    };
    App.SelectedItemViewModel = new HoneyDoItemViewModel();
    App.SelectedItemViewModel.HoneyDoItem = newHoneyDoItem;

    await Shell.Current.GoToAsync($"{nameof(HoneyDoItemPage)}");
}
```

3. Add the following XAML in bold to HoneyDoItemsPage.xaml above the RefreshView element:

```
<ContentPage.ToolbarItems>
    <ToolbarItem Text="Add"
                 Command="{Binding AddItemCommand}" />
</ContentPage.ToolbarItems>

<RefreshView x:DataType="local:HoneyDoItemsViewModel"
```

4. Run the app. Click Add on the main page. Create a new honey do item and click Save.
5. Click Completed to delete an item.

Introduction to Styles

1. Create a new Xamarin app, using the Blank template. Run the app.
2. Add the following to MainPage below the labels.

```
<StackLayout Orientation="Horizontal"
             Margin="0,25,0,0">
    <Button x:Name="SaveButton"
            Text="Save"
            BackgroundColor="LightBlue"
            HorizontalOptions="Start"
            Margin="25,0,0,25"
            WidthRequest="150"
            HeightRequest="50" />
    <Button x:Name="CompletedButton"
            Text="Completed"
            BackgroundColor="LightBlue"
            HorizontalOptions="Start"
            Margin="25,0,0,25"
            WidthRequest="150"
            HeightRequest="50" />
</StackLayout>
```

3. Add the following to each of the labels outside the Frame.

```
TextColor="Red"
```

4. Add the following at the top of the page above the first StackLayout element.

```
<ContentPage.Resources>
    <Style TargetType="Label">
        <Setter Property="TextColor"
                Value="Red" />
    </Style>
</ContentPage.Resources>
```

5. Remove the TextColor attributes you added to the labels. The labels are all red unless you have specified a different color, such as in the Frame.
6. Add the following to the ContentPage Resources

```
<Style TargetType="Button">
    <Setter Property="BackgroundColor"
            Value="LightBlue" />
    <Setter Property="HorizontalOptions"
            Value="Start" />
    <Setter Property="Margin"
            Value="25,0,0,25" />
    <Setter Property="WidthRequest"
            Value="150" />
    <Setter Property="HeightRequest"
            Value="50" />
</Style>
```

7. Remove those attributes from the buttons.
8. Add a third button. It looks just like the others.
9. Change the text color of the labels and background color of the buttons by making one change rather than multiple changes.
10. Add another page to the app.
11. Copy the Frame, Labels and Buttons from MainPage.
12. Copy the ContentPage Resources.
13. Make the following change in bold in App.xaml.cs.

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

    MainPage = new AnotherPage();
}
```

14. Run the app.
15. Copy the Style settings from either page and paste them in App.xaml

```
<Application.Resources>
    <Style TargetType="Label">
        <Setter Property="TextColor"
                Value="Purple" />
    </Style>
    <Style TargetType="Button">
        <Setter Property="BackgroundColor"
                Value="Wheat" />
        <Setter Property="HorizontalOptions"
                Value="Start" />
        <Setter Property="Margin"
                Value="25,0,0,25" />
        <Setter Property="WidthRequest"
                Value="150" />
        <Setter Property="HeightRequest"
                Value="50" />
    </Style>
</Application.Resources>
```

16. Comment out or remove the Style settings from both pages. Run the app.

Improve UI of HoneyDo app

1. Return to the HoneyDo app.

2. Explore the Application and Shell level resources already defined in App.xaml and AppShell.xaml. This is why the app has a particular UI to it. Those resources were auto created when you created the app.
3. Notice in AppShell that Shell.BackgroundColor is set to {StaticResource Primary}. Primary is defined as blue in App.xaml.
4. Remove the HorizontalOptions, Margin, WidthRequest and HeightRequest attributes from the buttons in HoneyDoItemPage.xaml
5. Add the following code in bold to the Button Style defined in App.xaml

```xml
<Style TargetType="Button">
    <Setter Property="TextColor"
            Value="White"></Setter>
    <Setter Property="HorizontalOptions"
            Value="Start" />
    <Setter Property="Margin"
            Value="25,0,0,25" />
    <Setter Property="WidthRequest"
            Value="150" />
    <Setter Property="HeightRequest"
            Value="50" />
    <Setter Property="CornerRadius"
            Value="75" />
    <Setter Property="VisualStateManager.VisualStateGroups">
```

6. Run the Android app.

7. Run the Windows app.

8. Comment out the CornerRadius setting in the App.xaml.
9. In HoneyDoItemPage.xaml.cs, add the following in bold to the constructor to set these in code for the Android app.

```
public HoneyDoItemPage()
{
    InitializeComponent();

    BindingContext = viewModel = App.SelectedItemViewModel;

    if (Device.RuntimePlatform == Device.Android)
    {
        SaveButton.CornerRadius = 75;
        CompletedButton.CornerRadius = 75;
    }
    if (Device.RuntimePlatform == Device.UWP)
    {
        SaveButton.CornerRadius = 15;
        CompletedButton.CornerRadius = 15;
    }
}
```

10. Run both the Windows and Android apps.

Enable deleting a task from the main page

1. Add the following code in bold to HoneyDoItemsViewModel:

```
public Command AddItemCommand { get; }
public Command DeleteItemCommand { get; }

public HoneyDoItemsViewModel()
{
    HoneyDoItems = new ObservableCollection<HoneyDoItem>();

    DataStore.Initialize();
```

```
        LoadItemsCommand = new Command(async () => await ExecuteLoadItemsCommand());
        ItemTapped = new Command<HoneyDoItem>(OnItemSelected);
        AddItemCommand = new Command(OnAddItem);
        DeleteItemCommand = new Command(OnDeleteItem);
    }
```

2. Add the following method:

```
private async void OnDeleteItem(object obj)
{
    IsBusy = true;

    try
    {
        await DataStore.DeleteItemAsync(App.SelectedItemViewModel.HoneyDoItem.Id);
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex);
    }
    finally
    {
        IsBusy = false;
    }
}
```

3. In HoneyDoItemsPage.xaml, make the following changes in bold:

```
<DataTemplate>
    <Grid Margin="0,10,0,0">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="50" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>

        <CheckBox x:Name="CompletedCheckBox"
                  Grid.Column="0"
                  HorizontalOptions="Center"
                  VerticalOptions="CenterAndExpand"
                  CheckedChanged="CompletedCheckBox_CheckedChanged" />

        <StackLayout Padding="10"
                     Grid.Column="1"
                     x:DataType="model:HoneyDoItem">
....

        </StackLayout>
    </Grid>
</DataTemplate>
```

4. Add the CompletedCheckBox_CheckedChanged method to HoneyDoItemsPage.xaml.cs:

```
private async void CompletedCheckBox_CheckedChanged(System.Object sender, CheckedChang
edEventArgs e)
{
    var honeyDoItemViewModel = new HoneyDoItemViewModel();
    honeyDoItemViewModel.HoneyDoItem = ((HoneyDoItem)((CheckBox)sender).BindingContext
);

    await honeyDoItemViewModel.ExecuteDeleteItemCommand();

    ((CheckBox)sender).IsChecked = false;
    viewModel.LoadItemsCommand.Execute(null);
```
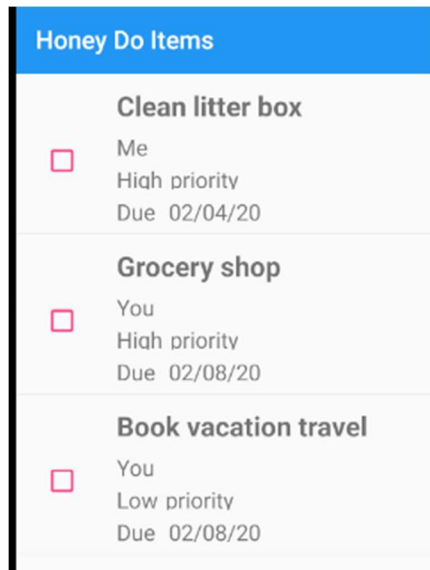
```
}
```

5. Add the following to the using statements.

   ```
   using HoneyDo.Models;
   ```

6. Run the app. Check one of the items and see it disappear.

Honey Do Items

Clean litter box
☐   Me
    High priority
    Due 02/04/20

Grocery shop
☐   You
    High priority
    Due 02/08/20

Book vacation travel
☐   You
    Low priority
    Due 02/08/20

7. In HoneyDoItemsPage.xaml, make the following changes in bold to add a Frame and remove the Grid's Margin:

   ```xml
   <DataTemplate>
       <Frame BorderColor="Black"
              Margin="0,0,0,20"
              Padding="5">
           <Grid >

   ….

           </Grid>
       </Frame>
   </DataTemplate>
   ```

8. Run the app

9. To add the ability to swipe to mark compete, dd the following code in bold in HoneyDoItemsPage.xaml:

```xaml
<Frame BorderColor="Black"
       Margin="0,0,0,20"
       Padding="5">
    <SwipeView>
        <SwipeView.RightItems>
            <SwipeItems Mode="Execute">
                <SwipeItem Text="Completed"
                           BackgroundColor="LightBlue"
                           Invoked="SwipeItem_Invoked" />
            </SwipeItems>
        </SwipeView.RightItems>
        <Grid>
…

    </SwipeView>
</Frame>
```

10. Add the following method to HoneyDoItemsPage.xaml.cs:

```csharp
private async void SwipeItem_Invoked(System.Object sender, System.EventArgs e)
{
    var honeyDoItemViewModel = new HoneyDoItemViewModel();
    honeyDoItemViewModel.HoneyDoItem =
        ((HoneyDoItem)((SwipteItem)sender).BindingContext);

    await honeyDoItemViewModel.ExecuteDeleteItemCommand();
    viewModel.LoadItemsCommand.Execute(null);
}
```

11. Run the app.
12. Swipe right to left on one of the items to mark it complete and remove it from the list.

NOTE: The current version of the app is saved in GitHub at https://github.com/rogreen/Xamarin-Workshop-App-V2.

Use SQLite

1. Right click on the HoneyDo shared code project and select Manage NuGet Packages.
2. In the Browse tab, search for sqlite.
3. Install the sqlite-net-pcl package.
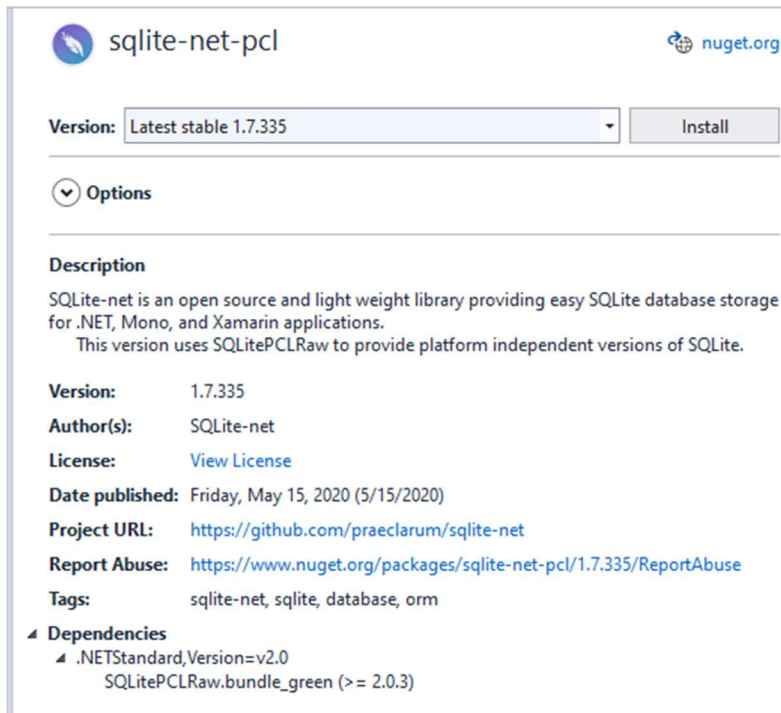
4. In the Services folder add a SQLiteDataStore.cs class.
5. Replace the using statements with the following:

```
using HoneyDo.Models;
using SQLite;
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading.Tasks;
```

6. Make the following change in bold.

```
public class SQLiteDataStore : IDataStore<HoneyDoItem>
```

7. Do not implement the IDataStore interface yet. You will add the various methods next.
8. Add the following code to connect to SQLite and create a new database if one doesn't already exist.

```
SQLiteAsyncConnection database;

public void Initialize()
{
    database = new SQLiteAsyncConnection(
        Path.Combine(Environment.GetFolderPath(
            Environment.SpecialFolder.LocalApplicationData),
            "HoneyDoSQLite.db3"));
    database.CreateTableAsync<HoneyDoItem>().Wait();
}
```

9. Add the following using statement to HoneyDoItem.cs.

```
using SQLite;
```

10. Make the following modification in bold.

```
[PrimaryKey, AutoIncrement]
public int Id { get; set; }
```

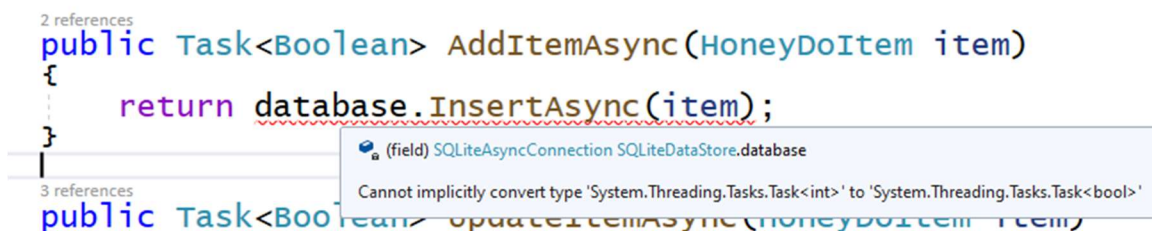11. Return to SQLiteDataStore.cs.

12. Add methods to retrieve all or one honey do item.

```
public Task<List<HoneyDoItem>> GetItemsAsync()
{
    //return database.Table<HoneyDoItem>().ToListAsync();
    return database.QueryAsync<HoneyDoItem>(
        "SELECT * FROM [HoneyDoItem] WHERE [Completed] = 0 " +
        "ORDER BY [DueDate]");
}

public Task<HoneyDoItem> GetItemAsync(Int32 id)
{
    return database.Table<HoneyDoItem>()
        .Where(i => i.Id == id).FirstOrDefaultAsync();
}
```

13. Add methods to add, update and delete.

```
public Task<Boolean> AddItemAsync(HoneyDoItem item)
{
    return database.InsertAsync(item);
}

public Task<Boolean> UpdateItemAsync(HoneyDoItem item)
{
    return database.UpdateAsync(item);
}

public Task<Boolean> DeleteItemAsync(Int32 id)
{
    var item = database.Table<HoneyDoItem>()
        .Where(i => i.Id == id).FirstOrDefaultAsync();

    return database.DeleteAsync(item);
}
```

```
2 references
public Task<Boolean> AddItemAsync(HoneyDoItem item)
{
    return database.InsertAsync(item);
}

3 references
public Task<Boolean> UpdateItemAsync(HoneyDoItem item)
```

🔓 (field) SQLiteAsyncConnection SQLiteDataStore.**database**

Cannot implicitly convert type 'System.Threading.Tasks.Task<int>' to 'System.Threading.Tasks.Task<bool>'

These methods return the number of rows modified, not a true or false value.

```
//
// Summary:
//     Inserts the given object and retrieves its auto incremented primary key if it
//     has one.
//
// Parameters:
//   obj:
//     The object to insert.
//
// Returns:
//     The number of rows added to the table.
public Task<Int32> InsertAsync(Object obj);
```

14. Modify the IDataStore interface to account for this.

```
Task<int> AddItemAsync(T item);
Task<int> UpdateItemAsync(T item);
Task<int> DeleteItemAsync(int id);
```

15. Modify the AddItemAsync, UpdateItemAsync and DeleteItemAsync methods in the MockDataStore class

```
public async Task<int> AddItemAsync(HoneyDoItem honeyDoItem)
{
    …
    return await Task.FromResult(1);
}

public async Task<int> UpdateItemAsync(HoneyDoItem honeyDoItem)
{
    …
    return await Task.FromResult(1);
}

public async Task<int> DeleteItemAsync(int id)
{
    …
    return await Task.FromResult(1);
}
```

16. Modify the same methods in SQLiteDataStore class.

```
public Task<int> AddItemAsync(HoneyDoItem item)
```

```
public Task<int> UpdateItemAsync(HoneyDoItem item)
```
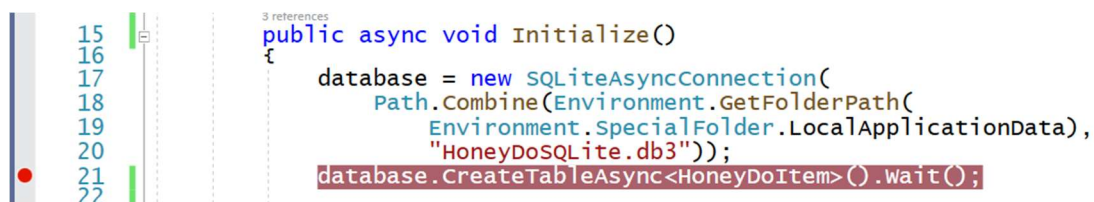
```
public Task<int> DeleteItemAsync(Int32 id)
```

17. Run the app and ensure we haven't broken the mock data code. Exit the app.
18. Make the following change in bold in App.xaml.cs.

```
DependencyService.Register<SQLiteDataStore>();
```

19. Download and install SQLiteStudio from https://github.com/pawelsalawa/sqlitestudio/releases.
20. Set a breakpoint in the Initialize method of the SQLiteDataStore class.

```
3 references
15  public async void Initialize()
16  {
17      database = new SQLiteAsyncConnection(
18          Path.Combine(Environment.GetFolderPath(
19              Environment.SpecialFolder.LocalApplicationData),
20              "HoneyDoSQLite.db3"));
21      database.CreateTableAsync<HoneyDoItem>().Wait();
22
```

21. Run the Windows app. When the app breaks, find and copy the database path.

```
HoneyDoSQLite.db3 )),
database.CreateTableAsync<HoneyDoItem>().Wait();
database    {SQLite.SQLiteAsyncConnection}
await       DatabasePath        "C:\\Users\\rgree\\AppData\\Local\\Packages\\bcf68800-9344-498e-9145-accfe1e4960d_4481w3grf8c22\\LocalState\\HoneyDoSQLite.db3"
{           DateTimeStringFormat "yyyy'-'MM'-'dd'T'HH':'mm':'ss'.'fff"
            LibVersionNumber    3026000
            StoreDateTimeAsTicks true
```
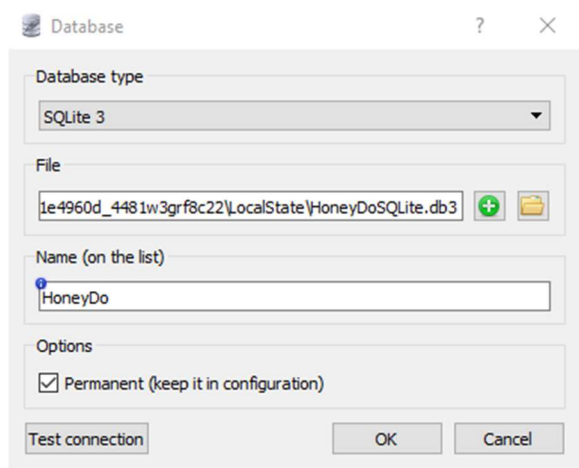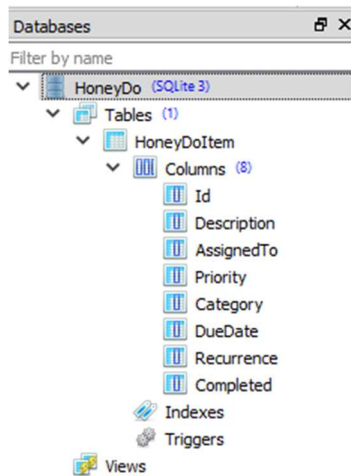
22. Continue the app.
23. Add a honey do item.
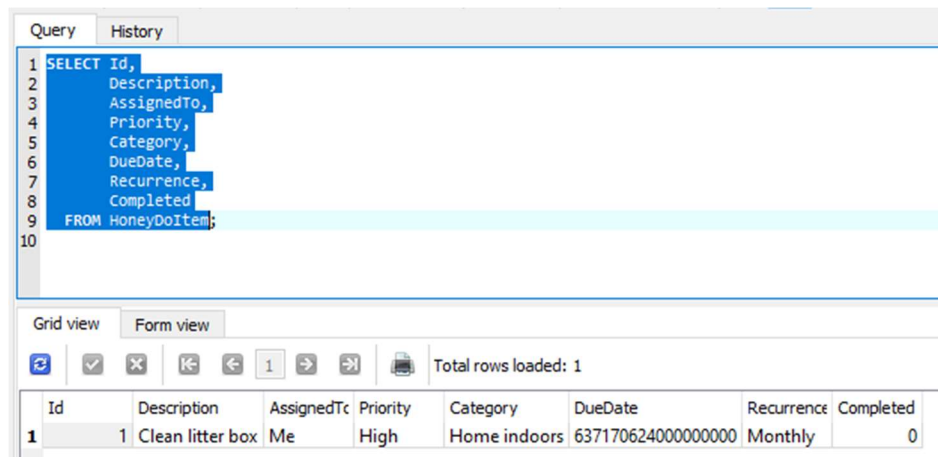24. Open SQLiteStudio.
25. Select Database|Add a database.
26. Paste the database path into the File textbox and set the Name to HoneyDo.

27. Click OK.
28. Right click on HoneyDo and select Connect to the database.

29. Right click on HoneyDoItem and select Generate query for table|SELECT. Execute the query.

30. In the app, add a second honey do item. Exit the app.
31. Launch the app from the Start menu. Three items appear because we are still adding an item in the Initialize.
32. Run the Android app. No items appear because the Android app is using a SQLite database on the Android device, not the database on your computer.
33. Add a new honey do item. Stop the app from within Visual Studio.

34. Navigate to the Home screen in the emulator and tap the app to run it.
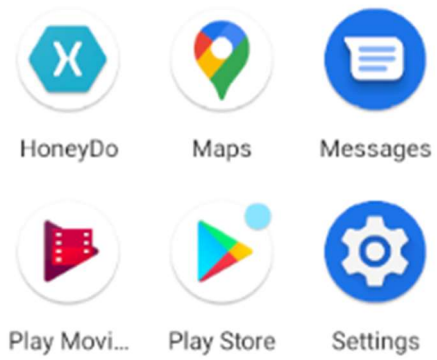
The item you added appears.

35. Test adding, updating and deleting items.

NOTE: The current version of the app is saved in GitHub at https://github.com/rogreen/Xamarin-Workshop-App-V3.

Use Azure REST API

I created HoneyDoService, following the steps in ASP.NET Core Web API with EF Core Code-First Approach.

I created a HoneyDoItems database in SQL Server and then migrated it to Azure SQL.

1. Right click on the HoneyDo shared code project and select Manage NuGet Packages.
2. In the Browse tab, search for newtonsoft.
3. Install the Newtonsoft.Json package.
4. In the Services folder add a RestApiDataStore.cs class.
5. Replace the using statements with the following:

```
using HoneyDo.Models;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
```

6. Make the following change in bold.

```
public class RestApiDataStore: IDataStore<HoneyDoItem>
```

7. Do not implement the IDataStore interface yet. You will add the various methods next.
8. Add the following declarations.

```
string baseAddress;
string honeyDoItemsUrl;
```

9. Add the following code to connect to SQLite and create a new database if one doesn't already exist.

```
public void Initialize()
{
    // The iOS simulator can connect to localhost.
    // However, Android emulators must use the 10.0.2.2 special alias
    // to your host loopback interface.
    //baseAddress = Device.RuntimePlatform ==
    //    Device.Android ? "https://10.0.2.2:58920" : "https://localhost:58920";

    baseAddress = "https://honeydoservice.azurewebsites.net";
    honeyDoItemsUrl = baseAddress + "/api/honeydoitems/{0}";
}
```

10. Add methods to retrieve all or one honey do item.

```
public async Task<List<HoneyDoItem>> GetItemsAsync()
{
    var honeydoItems = new List<HoneyDoItem>();
    var uri = new Uri(string.Format(honeyDoItemsUrl, string.Empty));
    var httpClient = new HttpClient();
    //var httpClient = new HttpClient(new System.Net.Http.HttpClientHandler());

    try
    {
        var response = await httpClient.GetAsync(uri);
        if (response.IsSuccessStatusCode)
        {
            var content = await response.Content.ReadAsStringAsync();
            honeydoItems = JsonConvert.DeserializeObject<List<HoneyDoItem>>(content);
        }
    }
    catch (Exception ex)
    {
```

```csharp
            Debug.WriteLine(@"\tERROR {0}", ex.Message);
        }

        return honeydoItems;

    }

    public async Task<HoneyDoItem> GetItemAsync(Int32 id)
    {
        var honeydoItem = new HoneyDoItem();
        var uri = new Uri(string.Format(honeyDoItemsUrl, id));
        var httpClient = new HttpClient(new System.Net.Http.HttpClientHandler());

        try
        {
            var response = await httpClient.GetAsync(uri);
            if (response.IsSuccessStatusCode)
            {
                var content = await response.Content.ReadAsStringAsync();
                honeydoItem = JsonConvert.DeserializeObject<HoneyDoItem>(content);
            }
        }
        catch (Exception ex)
        {
            Debug.WriteLine(@"\tERROR {0}", ex.Message);
        }

        return honeydoItem;
    }
```

11. Add methods to add, update and delete.

```csharp
    public async Task<Int32> AddItemAsync(HoneyDoItem item)
    {
        var uri = new Uri(string.Format(honeyDoItemsUrl, string.Empty));
        var httpClient = new HttpClient(new System.Net.Http.HttpClientHandler());
        int success = 0;

        try
        {
            var json = JsonConvert.SerializeObject(item);
            var content = new StringContent(json, Encoding.UTF8, "application/json");

            HttpResponseMessage response = null;
            response = await httpClient.PostAsync(uri, content);

            if (response.IsSuccessStatusCode)
            {
                Debug.WriteLine(@"\tHoneyDoItem successfully saved.");
                success = 1;
            }

        }
        catch (Exception ex)
        {
            Debug.WriteLine(@"\tERROR {0}", ex.Message);
        }

        return success;
    }

    public async Task<Int32> UpdateItemAsync(HoneyDoItem item)
    {
        var uri = new Uri(string.Format(honeyDoItemsUrl, item.Id));
        var httpClient = new HttpClient(new System.Net.Http.HttpClientHandler());
        int success = 0;

        try
        {
```

```
            var json = JsonConvert.SerializeObject(item);
            var content = new StringContent(json, Encoding.UTF8, "application/json");

            HttpResponseMessage response = null;
            response = await httpClient.PutAsync(uri, content);

            if (response.IsSuccessStatusCode)
            {
                Debug.WriteLine(@"\tHoneyDoItem successfully saved.");
                success = 1;
            }

        }
        catch (Exception ex)
        {
            Debug.WriteLine(@"\tERROR {0}", ex.Message);
        }

        return success;
    }

    public async Task<Int32> DeleteItemAsync(HoneyDoItem item)
    {
        var uri = new Uri(string.Format(honeyDoItemsUrl, item.Id));
        var httpClient = new HttpClient(new System.Net.Http.HttpClientHandler());
        int success = 0;

        try
        {
            var response = await httpClient.DeleteAsync(uri);

            if (response.IsSuccessStatusCode)
            {
                Debug.WriteLine(@"\tHoneyDoItem successfully deleted.");
                success = 1;
            }

        }
        catch (Exception ex)
        {
            Debug.WriteLine(@"\tERROR {0}", ex.Message);
        }

        return success;
    }
```

12. Make the following change in bold in App.xaml.cs.

```
DependencyService.Register<RestApiDataStore>();
```

13. Run the Windows app. Confirm honey do items appear.
14. Run the Android app. Confirm the same honey do items appear.
15. Make a change to an item in either the Windows or Android app.
16. In the other app, refresh the honey do items list and confirm that the change appears.

NOTE: The current version of the app is saved in GitHub at https://github.com/rogreen/Xamarin-Workshop-App-V4.