Android, iOS and Hybrid Applications

Mobile-Development

DAY 3

- Dialogs
- Styling
- Inversion of Control (IOC)
- Testing

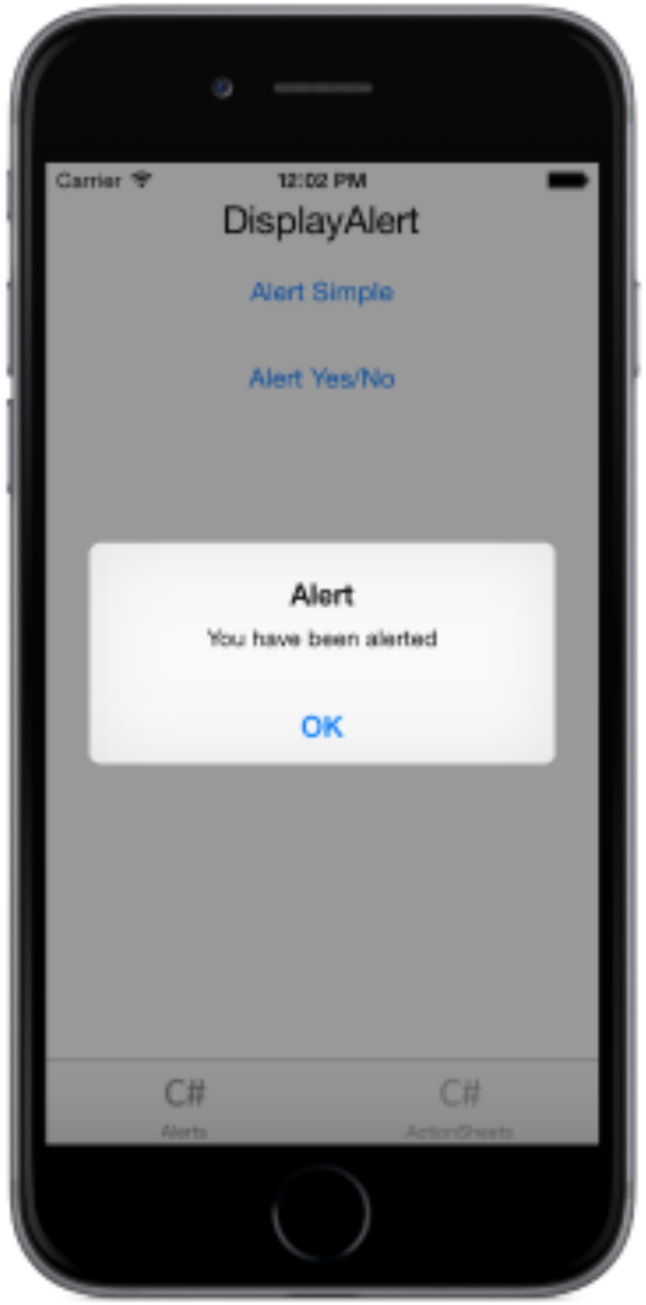
DIALOGS (POP-UPS)

- Call DisplayAlert(“”, “”) on any Page
- Ask “questions” with the overloads
- “Await” the result

- Action Sheets for a “DropDown” like behaviour

https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/navigation/pop-ups

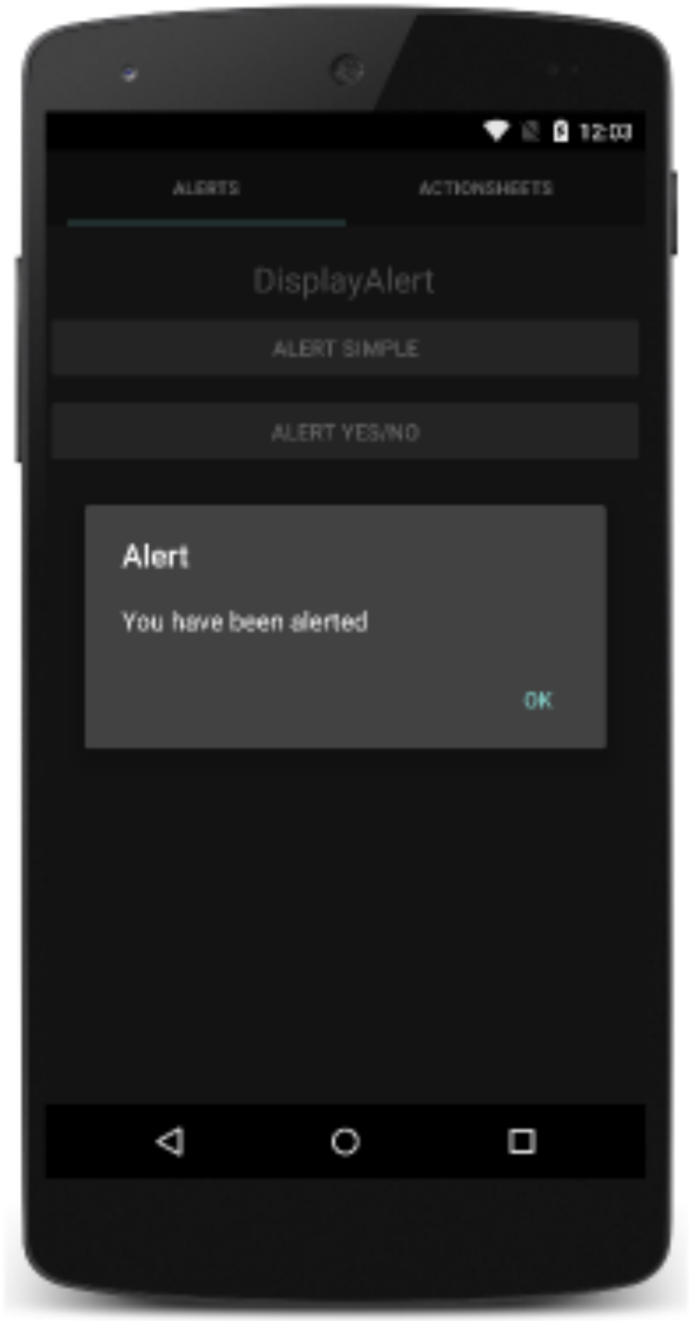
DIALOGS



Alert

You have been alerted

OK

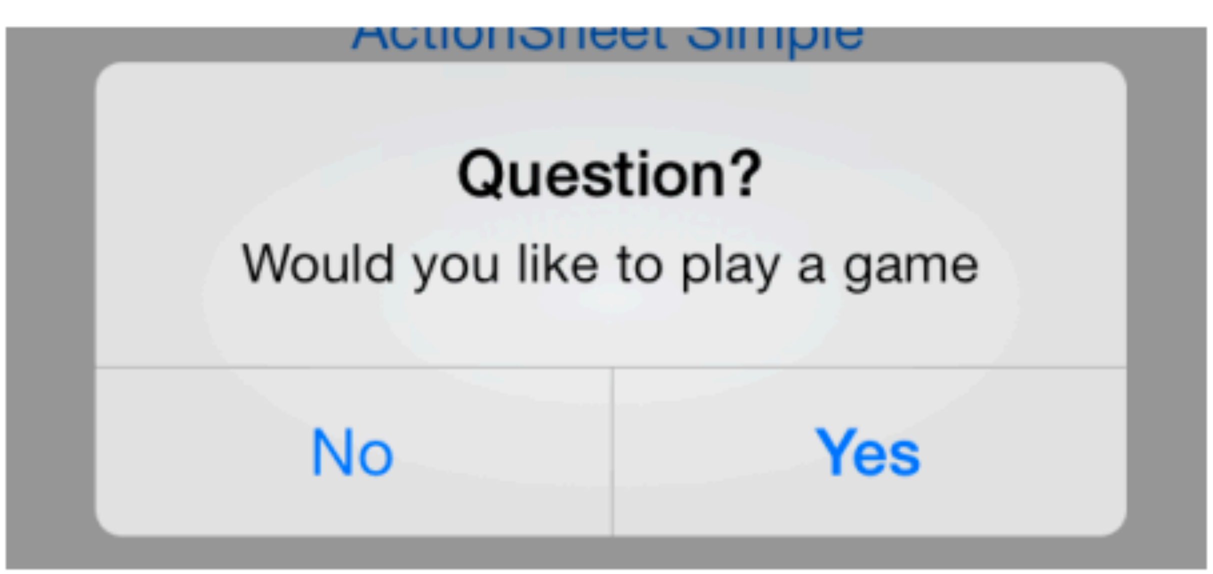


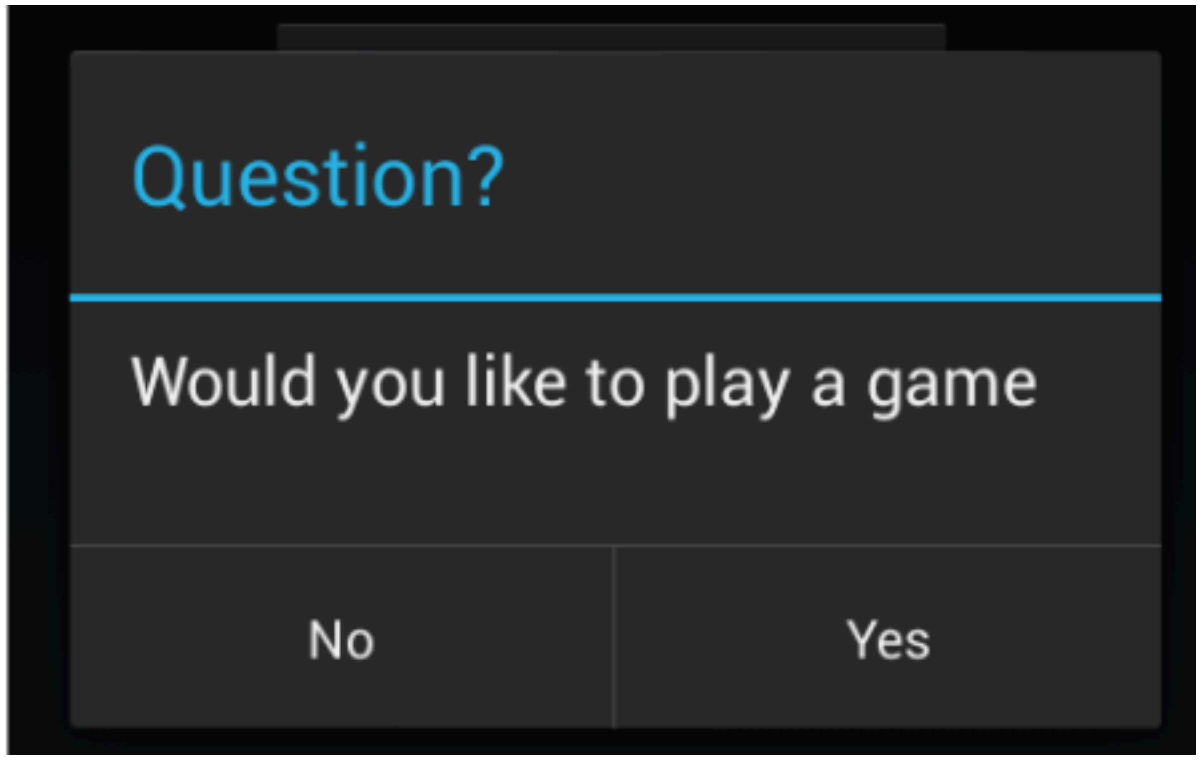
Alert

You have been alerted

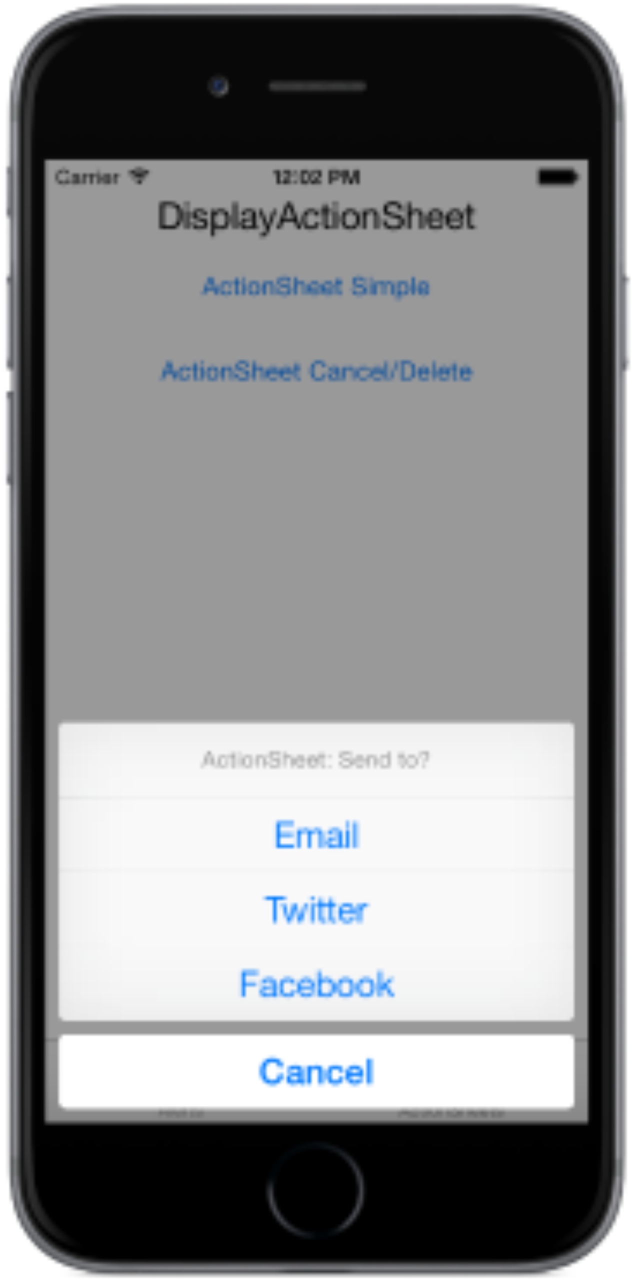
OK

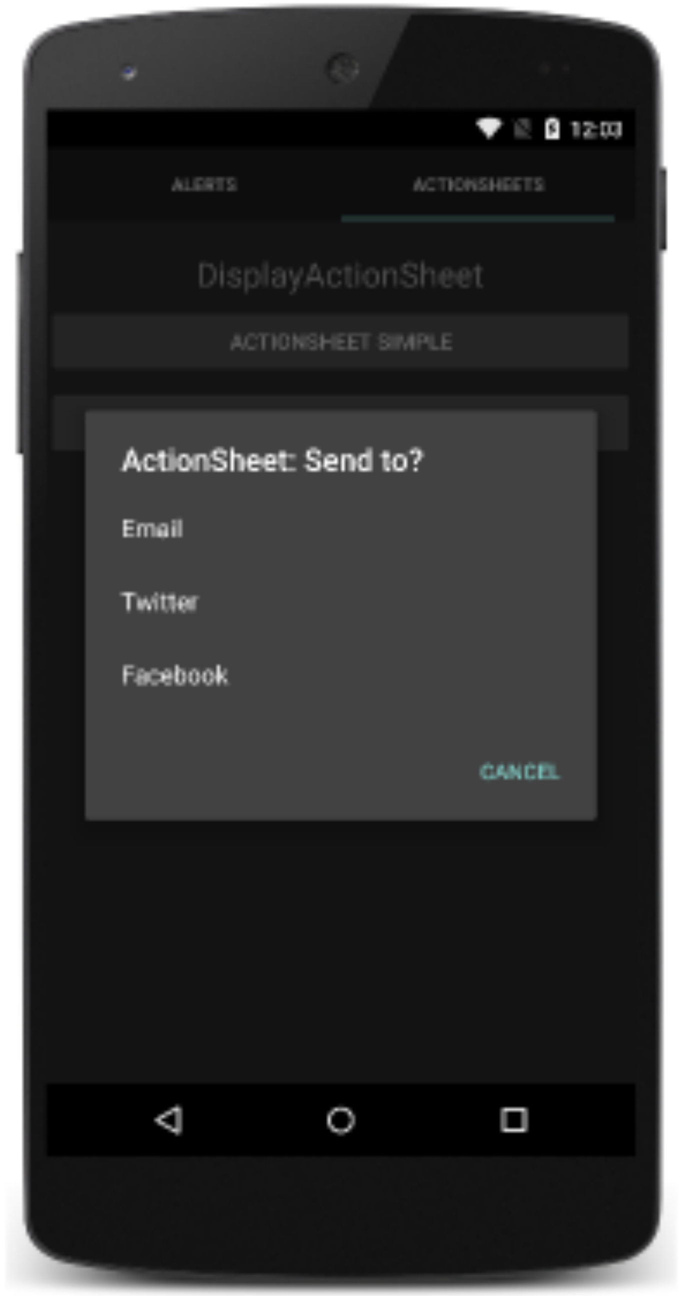
DIALOGS


ActionSheet Simple
Question?
Would you like to play a game
No
Yes


Question?
Would you like to play a game
No
Yes

Dialogs


Carrier
12:02 PM
DisplayActionSheet
ActionSheet Simple
ActionSheet Cancel/Delete
ActionSheet: Send to?
Email
Twitter
Facebook
Cancel


12:03
ALERTS
ACTIONSHEETS
DisplayActionSheet
ACTIONSHEET SIMPLE
ActionSheet: Send to?
Email
Twitter
Facebook
CANCEL

DIALOGS – CODE SAMPLES

```
DeleteItemCommmand = new Command<Page>(async (page) =>
{
  var result = await page.DisplayAlert("Confirm", "Are you sure?", "YES", "NO");
  if (result)
  {
    Items.Remove(SelectedItem);
  }
});
```

```
async void OnActionSheetSimpleClicked(object sender, EventArgs e)
{
  string action = await DisplayActionSheet("ActionSheet: Send to?", "Cancel",
                                           null, "Email", "Twitter", "Facebook");
  Debug.WriteLine("Action: " + action);
}
```

DIALOGS - CODE SAMPLES

```
<?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="Todo.Views.TodoListPage"
             xmlns:local="clr-namespace:Todo.Converters;assembly:Todo"
             xmlns:viewModels="clr-namespace:Todo.ViewModels;assembly:Todo"
             x:DataType="viewModels:TodoListViewModel"
             x:Name="Page">
    <ContentPage.Content>

            <Button Text="Delete" WidthRequest="100" Margin="20,0,0,0"
                        Command="{Binding DeleteItemCommand}" CommandParameter="{Reference Page}" />

    </ContentPage.Content>
</ContentPage>
```

DIALOGS - API

```
DisplayAlert(string title, string message, string cancel);
```

```
DisplayAlert(string title, string message,
             string accept, string cancel);
```

```
DisplayActionSheet(string title, string cancel,
                   string destruction, params string[] buttons);
```

Questions?

PRACTICE

- Example
- Use a Dialog in your solution
- Pass the Page via the command parameter

STYLING

- You can use XAML or CSS
- We're going to focus on XAML
- You can check online for what standard properties are supported by the various controls

https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/styles/xaml/index

STYLING – HIERARCHY

- Directly on an Element
- Explicit Styles - set the “Style” directly on an element
- Implicit Styles - a default style applied via the TargetType

STYLING ON THE ELEMENT - EXAMPLE

```
<Label
    Grid.Column="2"
    Text="X"
    TextColor="Red" />
```

STYLING EXPLICIT - EXAMPLE

```
<ContentPage
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="Todo.Views.TodoListPage"
    Title="List">
    <ContentPage.Style>
        <Style>
            <Setter
                Property="BackgroundColor"
                Value="Black" />
        </Style>
    </ContentPage.Style>
```

STYLING EXPLICIT - EXAMPLE

SomePage.xaml

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Todo.Views.TodoListPage"
             x:Name="Page"
             Title="List"
             Style="{StaticResource ContentPageStyle}">
```

App.xaml

```
<Application
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="Todo.App">
    <Application.Resources>
        <ResourceDictionary>
            <Color x:Key="Background">
                Black
            </Color>
           <Style  x:Key="ContentPageStyle" TargetType="ContentPage">
                <Setter Property="BackgroundColor" Value="Black" />
            </Style>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

STYLING IMPLICIT - EXAMPLE

```
<Style
    TargetType="Button">
    <Setter
        Property="BackgroundColor"
        Value="#3541a0" />
    <Setter
        Property="TextColor"
        Value="White" />
    <Setter
        Property="HeightRequest"
        Value="50" />
</Style>
```

STYLING – POSSIBLE VALUES

- Button
  - BackgroundColor
  - BorderRadius
  - BorderWidth
  - BorderColor
  - TextColor

https://docs.microsoft.com/en-gb/dotnet/api/xamarin.forms.button?view=xamarin-forms

STYLING - POSSIBLE VALUES

- Entry
  - TextColor
  - FontSize
  - FontFamily
  - PlaceholderColor

https://docs.microsoft.com/en-gb/dotnet/api/xamarin.forms.entry?view=xamarin-forms

STYLING - POSSIBLE VALUES

- Picker
  - TextColor
  - FontSize
  - FontFamily
  - TitleColor

https://docs.microsoft.com/en-gb/dotnet/api/xamarin.forms.picker?view=xamarin-forms

STYLING – POSSIBLE VALUES

- Label
  - TextColor
  - BackgroundColor
  - FontSize
  - FontFamily
  - TextDecorations

https://docs.microsoft.com/en-gb/dotnet/api/xamarin.forms.label?view=xamarin-forms

STYLING - ADDITIONAL RESOURCES

- Triggers

  https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/triggers

- Custom Renderer

  https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/custom-renderer/

- Effects

  https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/effects/

QUESTIONS?

PRACTICE

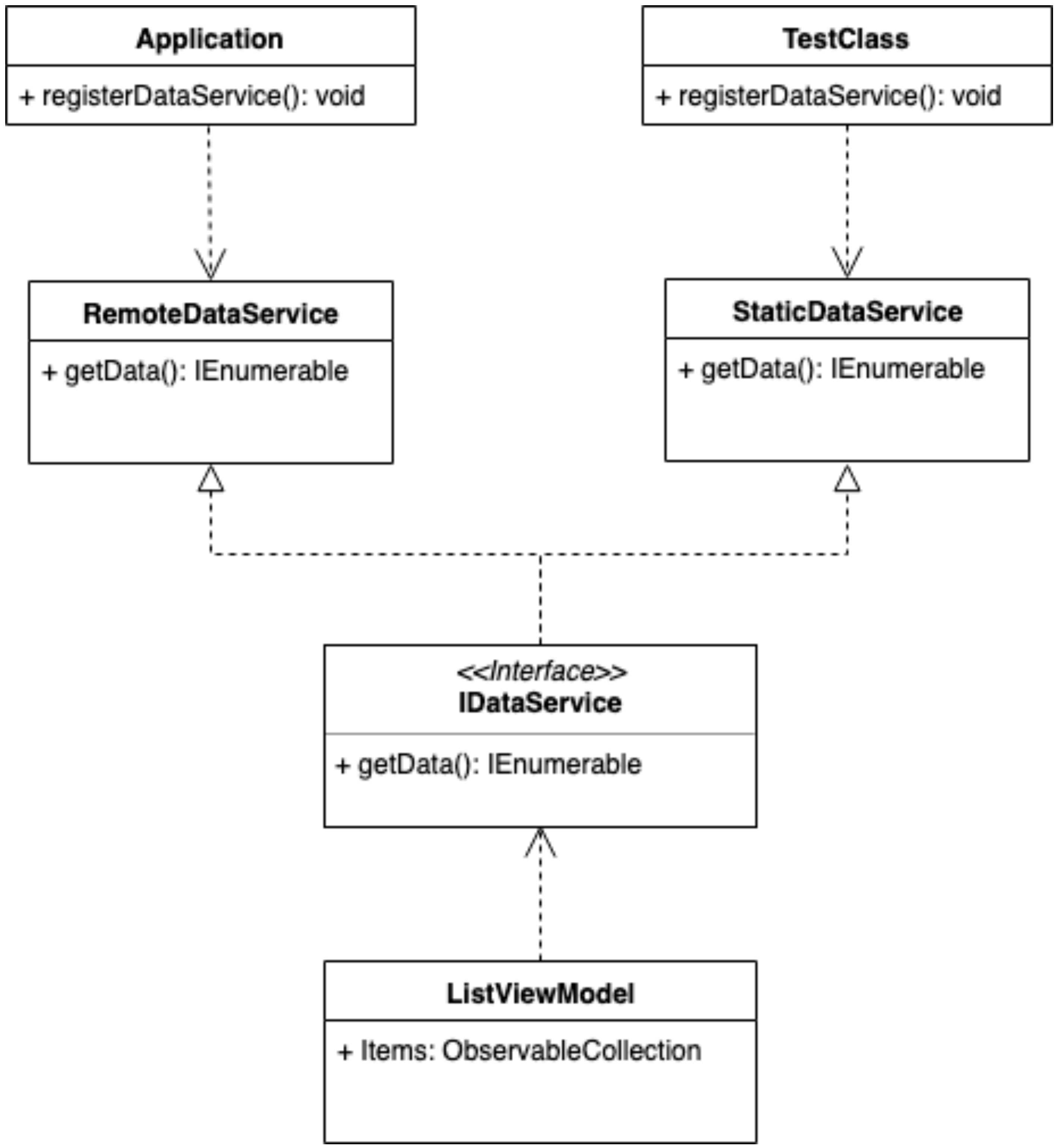
- Example
- Style your App

IOC

- Inversion of Control
- Dependency Injection as a specialised version
- We don’t want to work with concrete implementations
- A container holds the registrations and resolves them

https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/dependency-injection

IOC


Application
+ registerDataService(): void
TestClass
+ registerDataService(): void
RemoteDataService
+ getData(): IEnumerable
StaticDataService
+ getData(): IEnumerable
<<Interface>>
IDataService
+ getData(): IEnumerable
ListViewModel
+ Items: ObservableCollection

IOC – DIALOGSERVICE – INTERFACE

```
public interface IDialogService
{
  Task Show(string title, string message);

  Task<bool> Show(string title, string message, string positive, string negative);
}
```

IOC – DIALOGSERVICE – IMPLEMENTATION

```
public class DialogService : IDialogService
{
  public DialogService(Page page)
  {
    _page = page;
  }

  public async Task Show(string title, string message)
  {
    await _page.DisplayAlert(title, message, "Cancel");
  }

  public async Task<bool> Show(string title, string message, string positive, string negative)
  {
    return await _page.DisplayAlert(title, message, positive, negative);
  }

  private readonly Page _page;
}
```

IOC – DialogService – TestImplementation

```
public class MockDialogService : IDialogService
{
  public Task Show(string title, string message)
  {
    return Task.CompletedTask;
  }

  public Task<bool> Show(string title, string message, string positive, string negative)
  {
    return Task.FromResult(true);
  }
}
```

IOC – DIALOGSERVICE – USAGE

```
DeleteItemCommand = new Command(async () =>
{
  if (await dialogService.Show("Confirm", "Are you sure you want to delete the item?", "Yes", "No"))
  {
    Items.Remove(Items.First(x => x.IsSelected));
  }
});
```

IOC – WORKFLOW

- Register your Services
- Container.Register<Interface, Implementation>()
- Seal the container - no more registrations after this point
- Resolve services using the container
- Container.Resolve<Interface>()

IOC – TRANSIENT VS SINGLETON

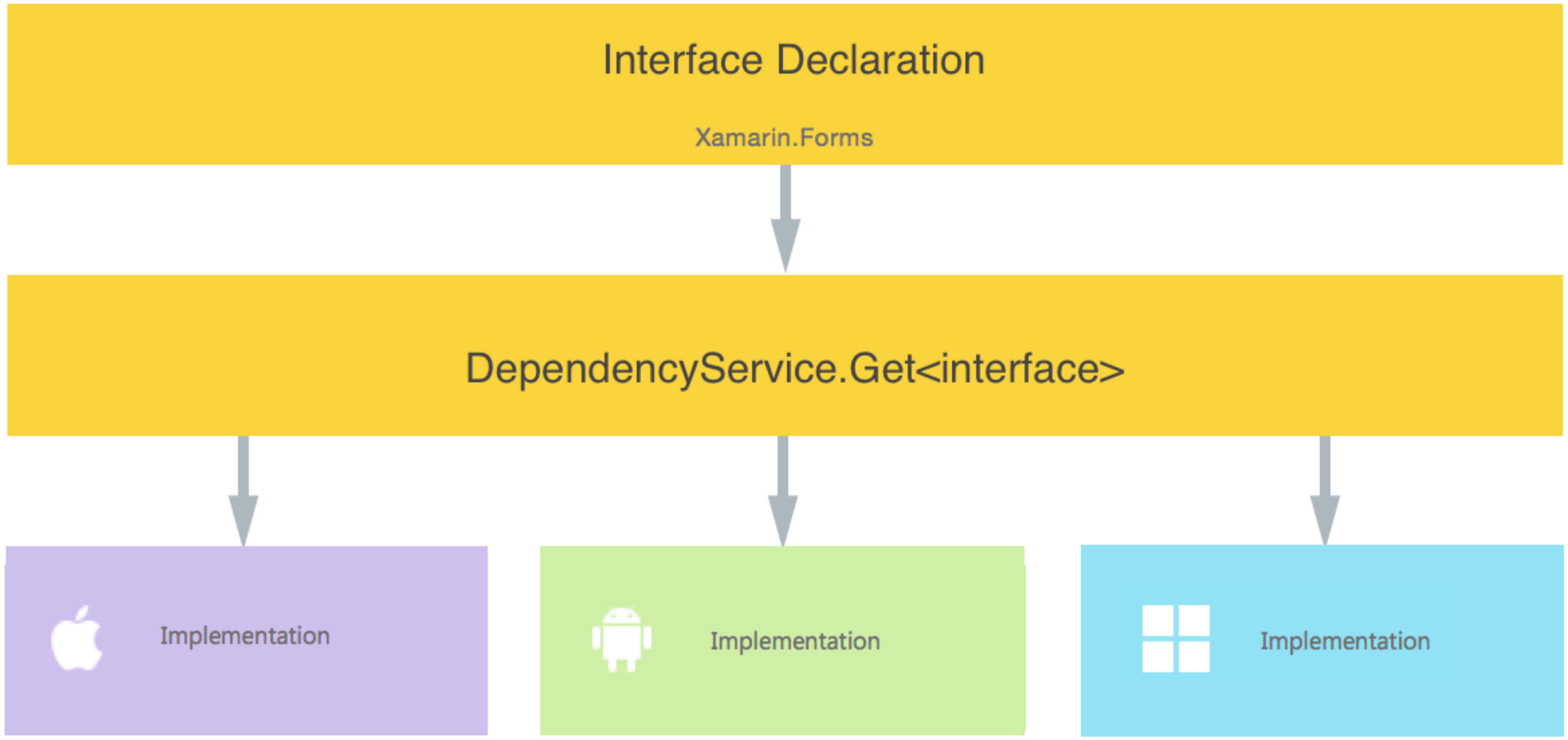
- Singleton exists only once per app/container
- Transient objects are created with every request

- Lifestyle mismatch! Singleton which depends on transient!

IOC – ADVANTAGES

- Replace your services for testing
- No more “new” all through the code
- Use different implementations based on a condition (iOS, Android for example)
- Constructor injection is easy to understand and spot dependencies

IOC – XAMARIN FORMS



XAMARIN FORMS – IOC

```
Shared:
public interface ISomeService
{
    void Foo();
}


Android:
using System;
using FormsTesting.Droid;
using Xamarin.Forms;

// Android specific implementation. Registration via attribute.
[assembly: Dependency(typeof(SomeService))]
namespace FormsTesting.Droid
{
    public class SomeService : ISomeService
    {
        public void Foo()
        {
            throw new NotImplementedException();
        }
    }
}


-> Same for iOS
```

IOC – XAMARIN FORMS

- We can but we don’t need to use it
- It’s a pretty simple container with a lot of limitations
- You **have** to use it for custom controls

IOC – SIMPLE INJECTOR

- There're a lot of IoC frameworks out there
- Cross Platform
- Good documentation
- Used in the sample project

https://simpleinjector.readthedocs.io/en/latest/quickstart.html

IOC – SIMPLE INJECTOR

```
// Register services we need to setup our application.
Services.RegisterInstance(navigationPage.Navigation);
Services.Register<IViewMapper, ViewMapper>(Lifestyle.Singleton);
Services.Register<ITodoViewModelFactory, TodoViewModelFactory>(Lifestyle.Singleton);
Services.Register<ITodoItemProvider, TodoItemProvider>(Lifestyle.Singleton);
Services.Register<MainViewModel>(Lifestyle.Singleton);
Services.Register<TodoListViewModel>(Lifestyle.Singleton);
Services.Register<TodoItemViewModel>(Lifestyle.Transient);
```

```
Services.GetInstance<TodoListViewModel>()
```

```
public TodoListViewModel(INavigation navigation, IViewMapper viewMapper,
ITodoViewModelFactory viewModelFactory, ITodoItemProvider provider)
{
  // Constructor
}
```

TESTING

- Use a standard .NET Core Unit Test project
- Reference your shared project
- One test class per service
- Feel free to create base classes or helper methods

https://docs.microsoft.com/en-us/dotnet/core/testing/

Testing - Setup

```
[TestFixture]
public class Tests
{
  [SetUp]
  public void Setup()
  {
    // Potentially register different services to setup a "predictable" test environment.
    App.Services.RegisterInstance(new NavigationPage().Navigation);
    App.Services.Register<IViewMapper, ViewMapper>(Lifestyle.Singleton);
    App.Services.Register<ITodoViewModelFactory, TodoViewModelFactory>(Lifestyle.Singleton);
    App.Services.Register<ITodoItemProvider, TodoItemProvider>(Lifestyle.Singleton);
    App.Services.Register<MainViewModel>(Lifestyle.Singleton);
    App.Services.Register<TodoListViewModel>(Lifestyle.Singleton);
    App.Services.Register<TodoItemViewModel>(Lifestyle.Transient);

    App.Services.Register<IDialogService, MockDialogService>(Lifestyle.Singleton);
  }
}
```

TESTING - TEST

```
[TestFixture]
public class Tests
{
  // Setup excluded

  [Test]
  public void TestEmptyTodoCantBeSaved()
  {
    var listViewModel = App.Services.GetInstance<TodoListViewModel>();
    var todoItemViewModel = App.Services.GetInstance<ITodoViewModelFactory>()
                                        .Create(new TodoItem(),   listViewModel);

    Assert.That(todoItemViewModel.SaveCommand.CanExecute(null), Is.False);

    todoItemViewModel.Title = "Title";
    Assert.That(todoItemViewModel.SaveCommand.CanExecute(null), Is.True);
  }
}
```

Questions?

TESTING & IoC

- Include an IoC in your app
- Move your dependencies into the IOC
- Register different services for your test scenarios
- Examples:
  - On/Offline service
  - Item Provider or similar that connects to an API/DB
  - Any UI specific/related services

ADDITIONAL TASKS

- Apply some of the additional styling options
- Use CSS to style something
- Expand the tests with TestCases