IOS115

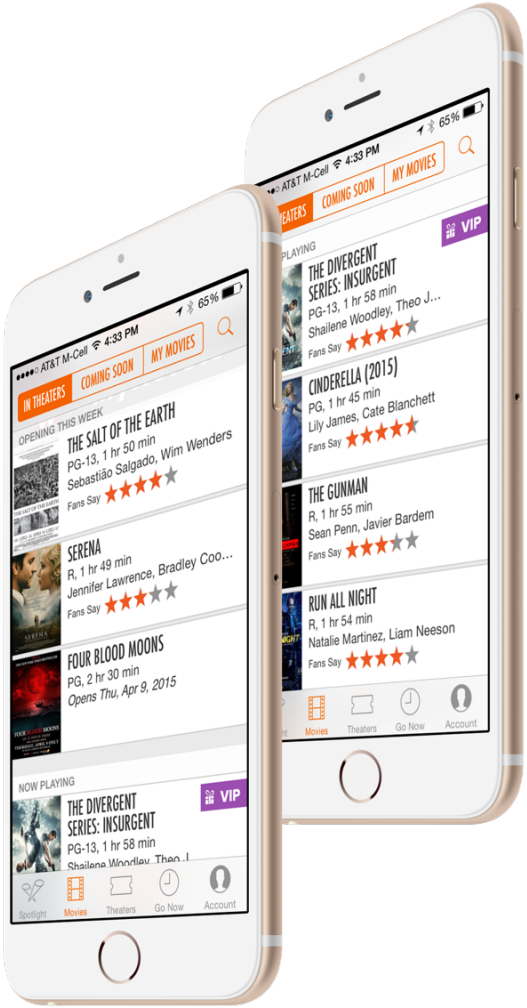
Customizing Table Views

- Lecture will begin shortly
- Download class materials from university.xamarin.com

Xamarin University

Objectives

1. Customize table view cells in code
2. Customize table view cells in the designer
3. Group data in the table view

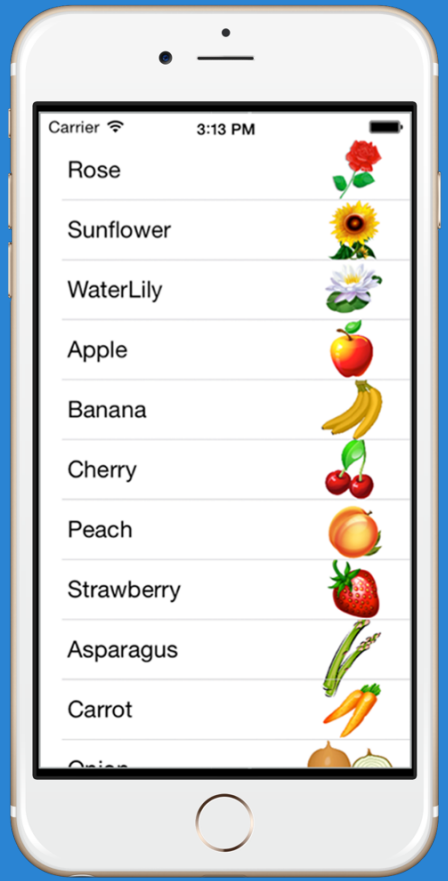


Customize table view cells in code

Xamarin University

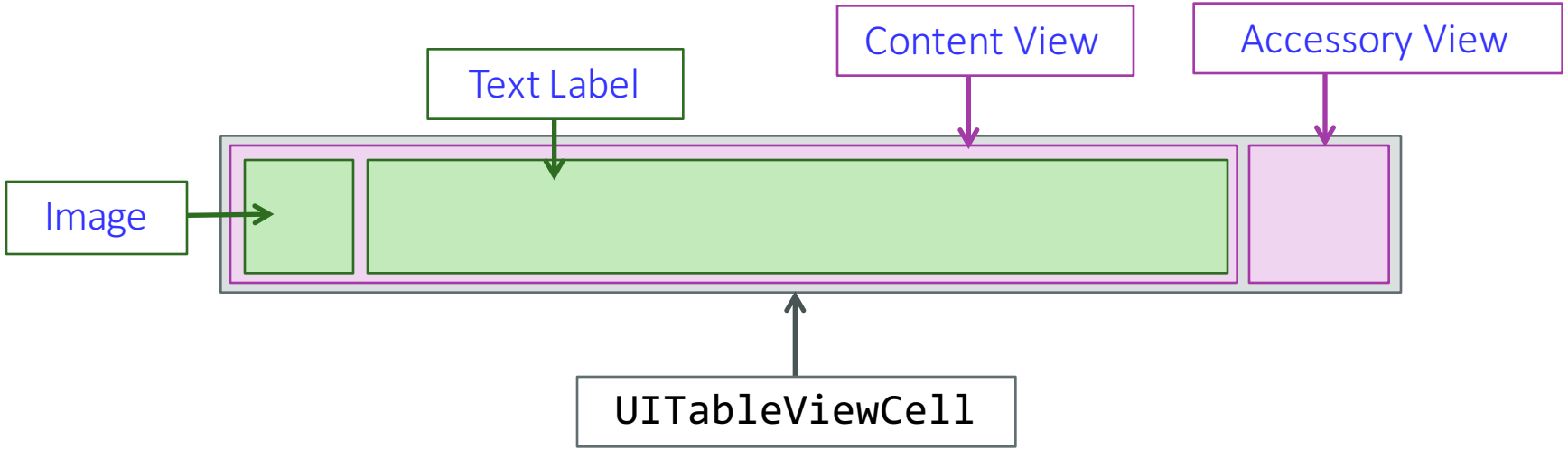
Tasks

1. View the anatomy of a cell
2. Customize the default cell
3. Identify the steps to creating a custom cell
4. Create a custom cell



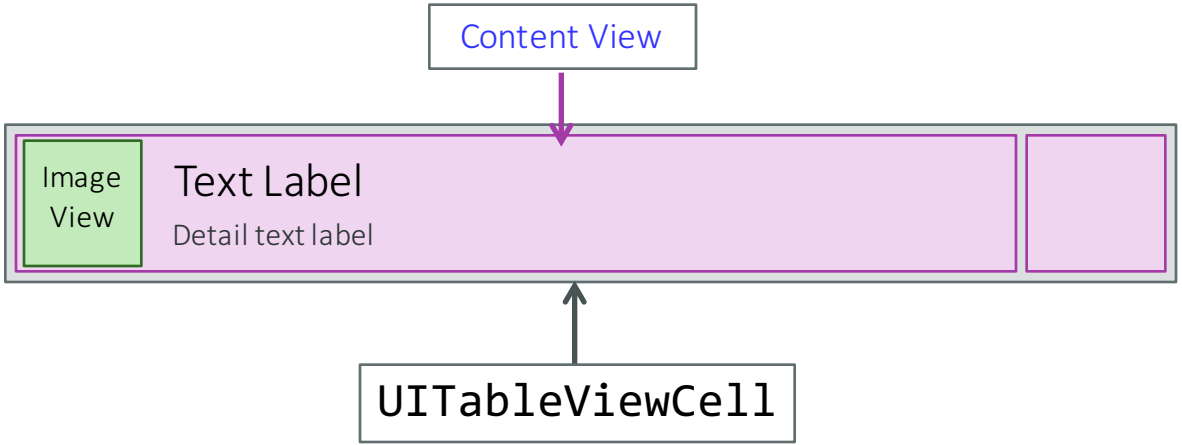
Anatomy of a default cell

- The default **`UITableViewCell`** is composed of the cell and several subviews, which allows for a high degree of customization out of the box



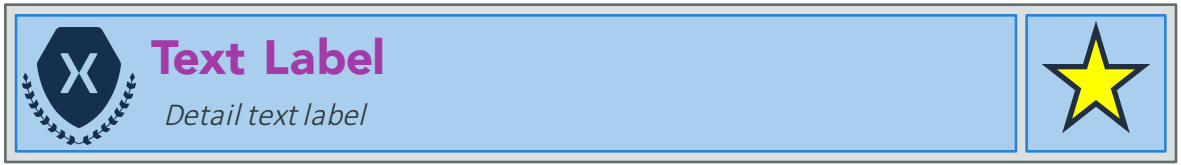
Subviews

- One way to create a custom cell is to style the default content view, taking advantage of the built-in classes to adjust fonts, colors, and change the accessory image



Customize the default views

- We can change the properties on the built-in subviews in the `GetCell` implementation

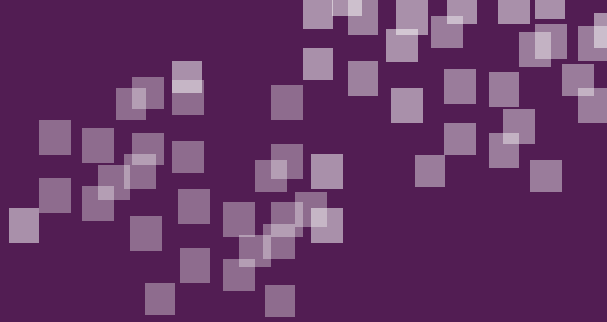


All default cells all contain the `DetailTextLabel`, and `ImageView` properties but they will be `null` for styles that don't support these visualizations

Accessory view

- The accessory view is used to indicate state or behavior when a cell is tapped – you can customize the image or replace it with a custom `UIView`

The accessory view shows additional information like state (checkmark) or it indicates behavior (chevron for navigation). *It shouldn't be used for cell content.*



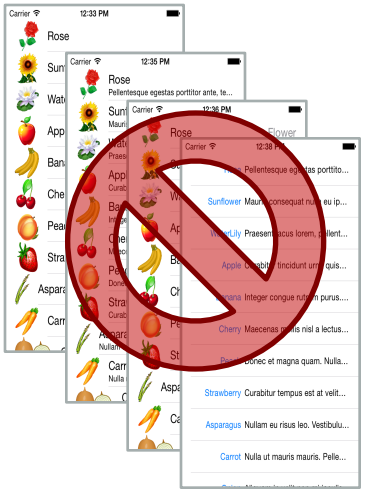
Individual Exercise

Customize a default table cell

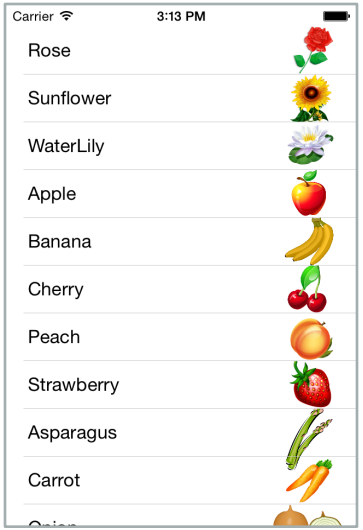
Xamarin University

Custom Table View cells

- Built-in cell styles cover common scenarios, but sometimes you need to display information in ways that are not supported by default - when this happens you can turn to a **custom table view cell**



What if we want the image on the right side?

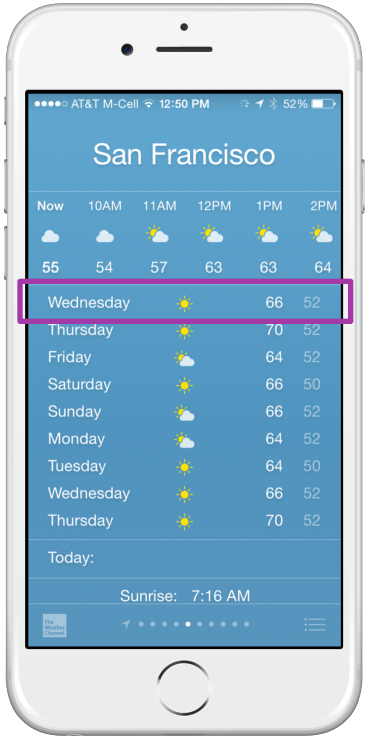


Creating a custom table view cell

- Custom cells can be created either in code, or in the Storyboard designer

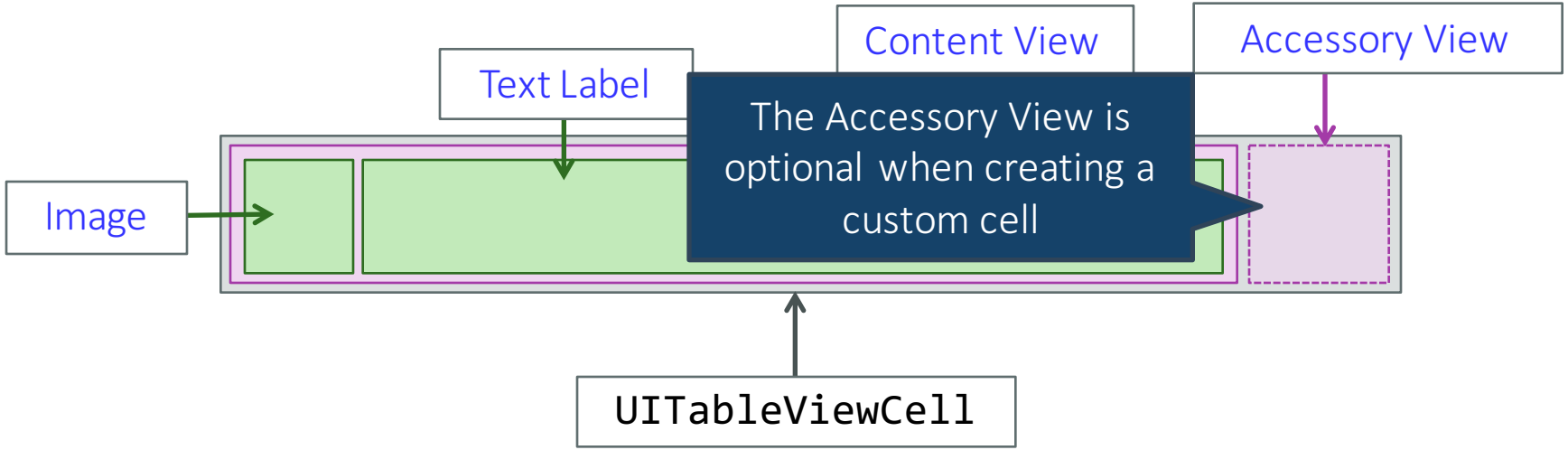
Completely customized cells

- Sometimes the data you want to display doesn't fit within the confines of the default cell
- When this happens, it is necessary to make a *custom cell*



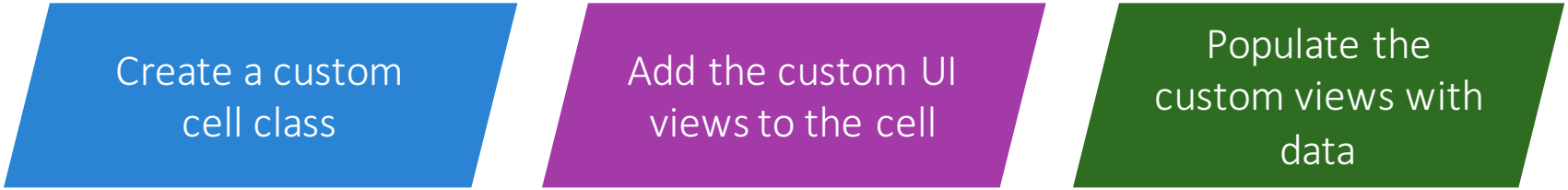
Anatomy of a custom cell

- The content view is blank in a custom cell, it's up to you to populate it with custom controls and visuals



Steps to creating a custom cell

- There are three steps to creating a custom cell



Create a custom cell class

- A custom cell class derives from **`UITableViewCell`** and defines the UI and behavior of the cell

```
public class PlantTableViewCell : UITableViewCell
{
    ...
}
```

Create a custom cell class

- If the cell is used within a Table View created in the designer, the constructor must but updated

```
public class PlantTableViewCell : UITableViewCell
{
    public PlantTableViewCell(IntPtr handle) : base (handle)
    {
    }
    ...
}
```

Constructor is passed a native handle and must forward the call to the base class

Add the custom UI visuals to the cell

- Create custom subview(s) to display data within the cell and add them to the `ContentView`

```
UILabel plantName;  // hold reference to update

public PlantTableViewCell (IntPtr handle) : base (handle)
{
    plantName = new UILabel();
    ContentView.AddSubview (plantName);
}
```

Layout the cell

- Override the `LayoutSubviews` method to size and position the child views in your cell

```
UILabel plantName;  // hold reference to update

public override void LayoutSubviews()
{
    base.LayoutSubviews ();
    plantName.Frame = new CGRect(10, 18, 100, 20);
    ...
}
```

Register the cell with the UITableView

- Register the cell for *reuse* using the **`RegisterClassForCellReuse`** method on your **`UITableView`**

```
public class PlantTVC : UITableViewController
{
    public PlantTVC ()
    {
        TableView.RegisterClassForCellReuse(
           typeof(PlantCellView), "PlantCellId");
        ...
    }
}
```

Recall that reusing cells optimizes the memory and performance of your application – you should always utilize this iOS feature

Visualize the data in the cell

- We expose a public method to update the content of the child views

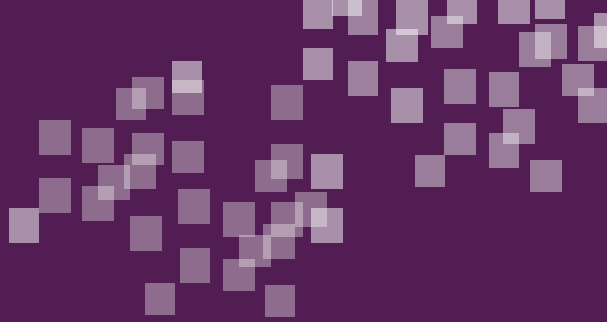
```
public class PlantTableViewCell : UITableViewCell
{
    ...
    public void UpdateCell (Plant plant)
    {
        plantName.Text = plant.Name;
        plantImage.Image = LoadImageFromUrl(plant.ImageUrl);
    }
}
```

Visualize the data in the cell

- Call the update method on the custom cell from the `GetCell` method in the table view controller

```
public override UITableViewCell GetCell(UITableView tableView,
                                        NSIndexPath indexPath)
{
    Plant plant = plants[indexPath.Row];
    var cell = tableView.DequeueReusableCell("PlantCellId");

    cell.UpdateCell(plant);
    return cell;
}
```



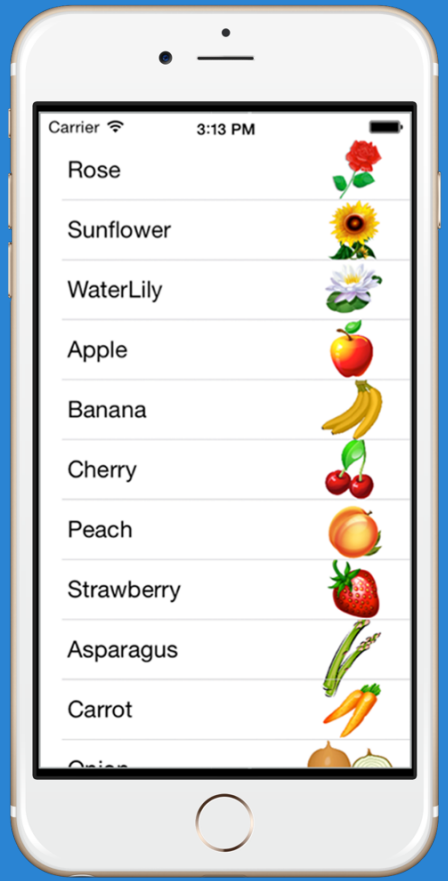
Individual Exercise

Create a custom table view cell in code

Xamarin University

Summary

1. View the anatomy of a cell
2. Customize the default cell
3. Identify the steps to creating a custom cell
4. Create a custom cell



Customize table view cells in the designer

Xamarin University

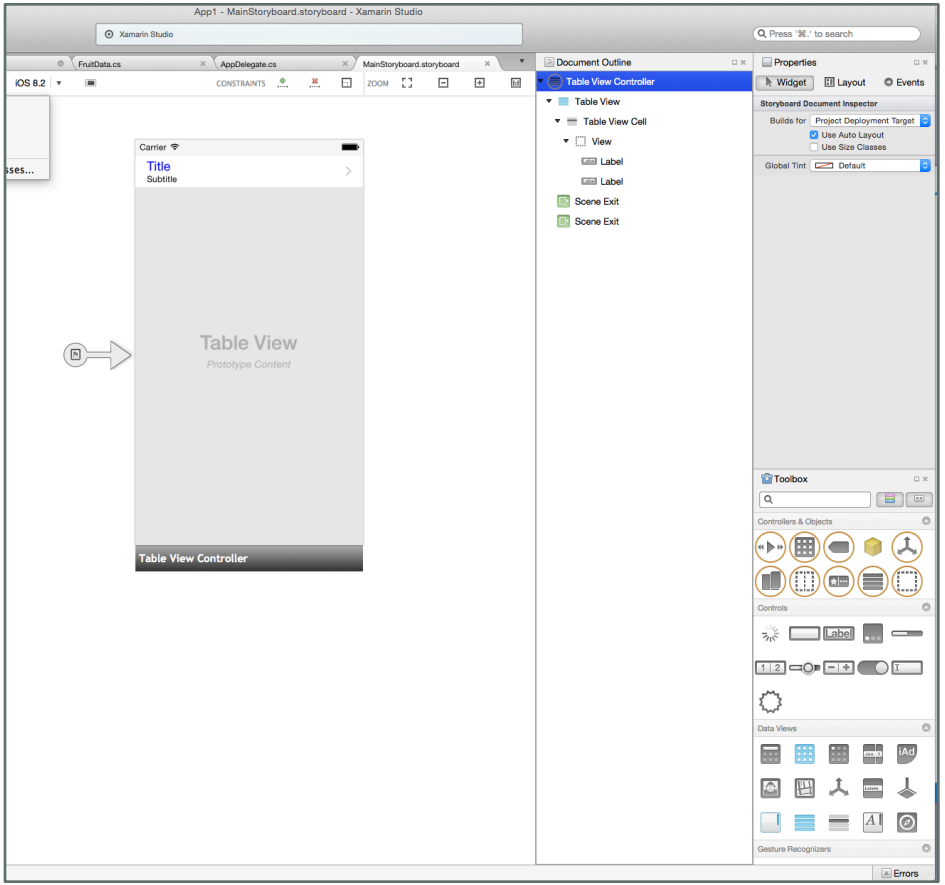
Tasks

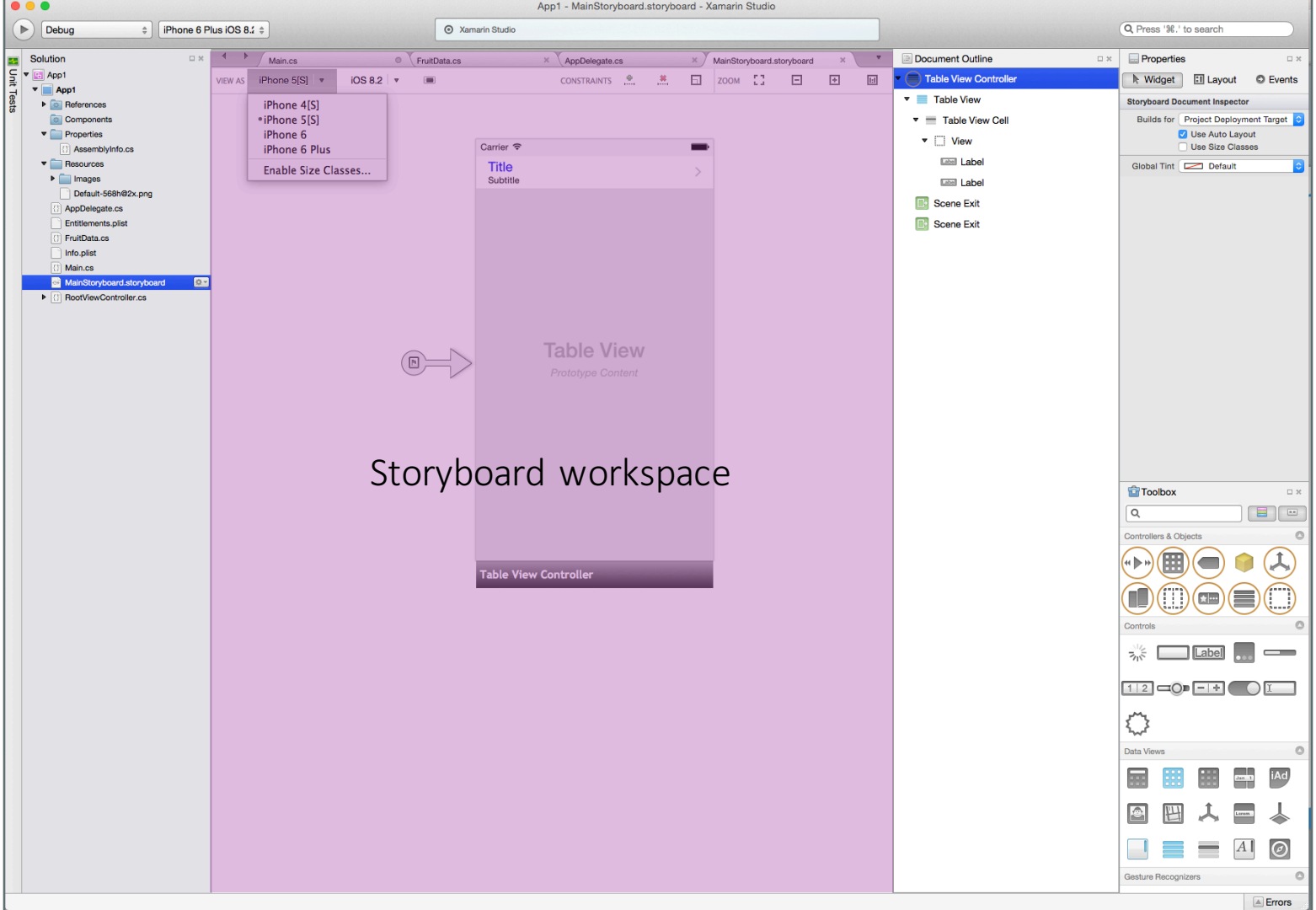
1. Walkthrough the designer
2. Compare static and dynamic cells
3. Design a custom cell using the designer

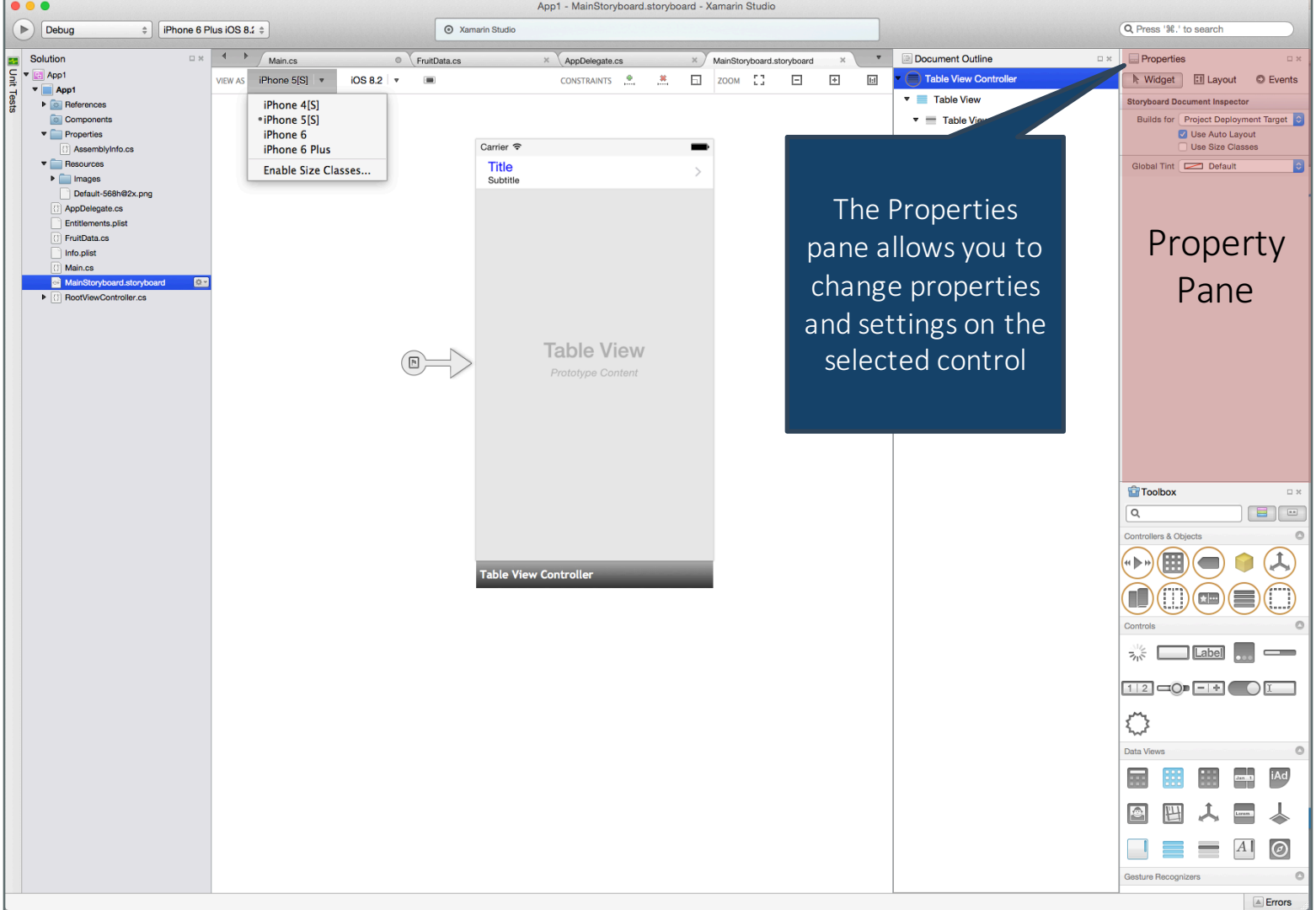
The iOS Designer

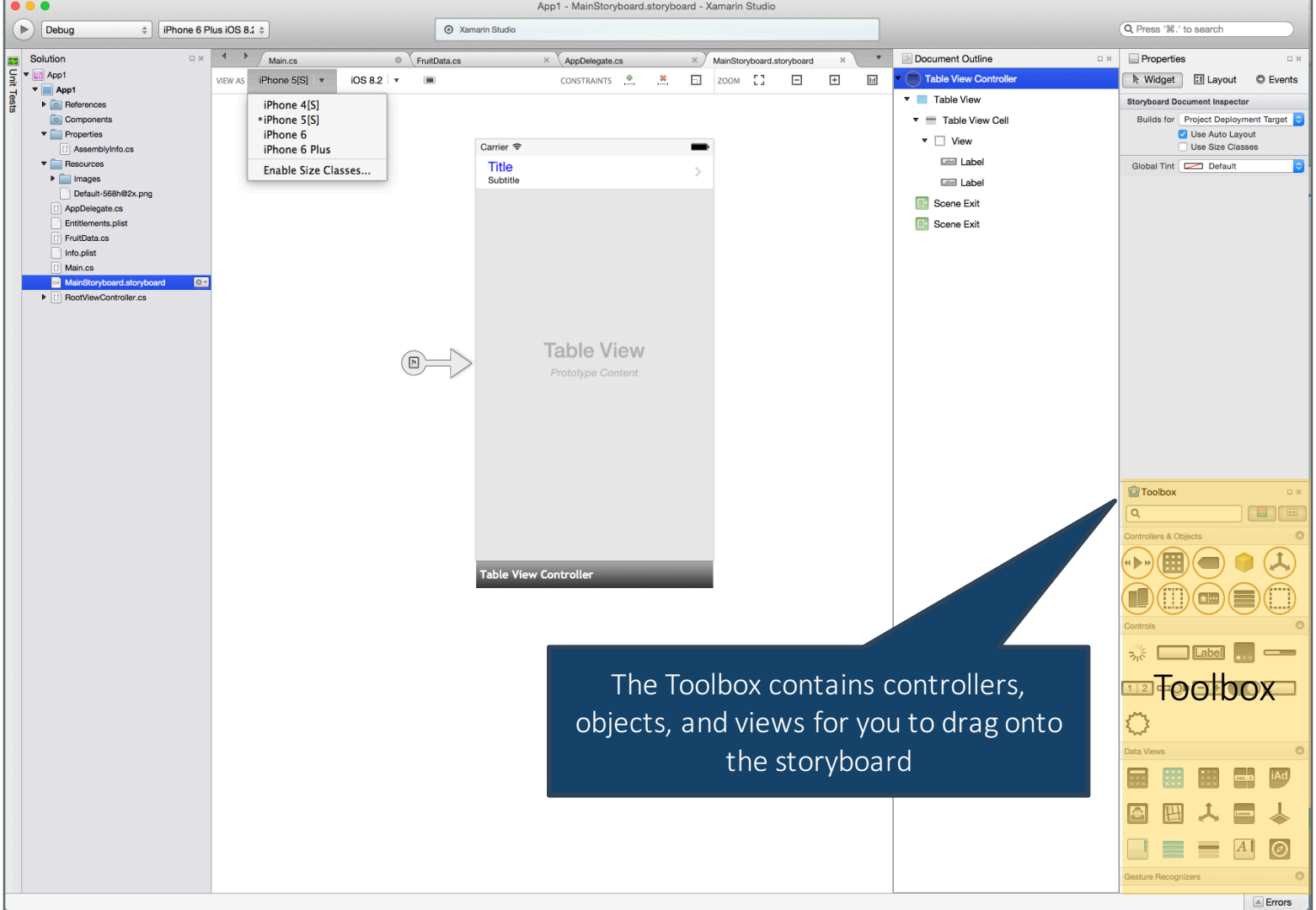
- The Xamarin.iOS designer allows you design and see all of your screens and how they relate to one another

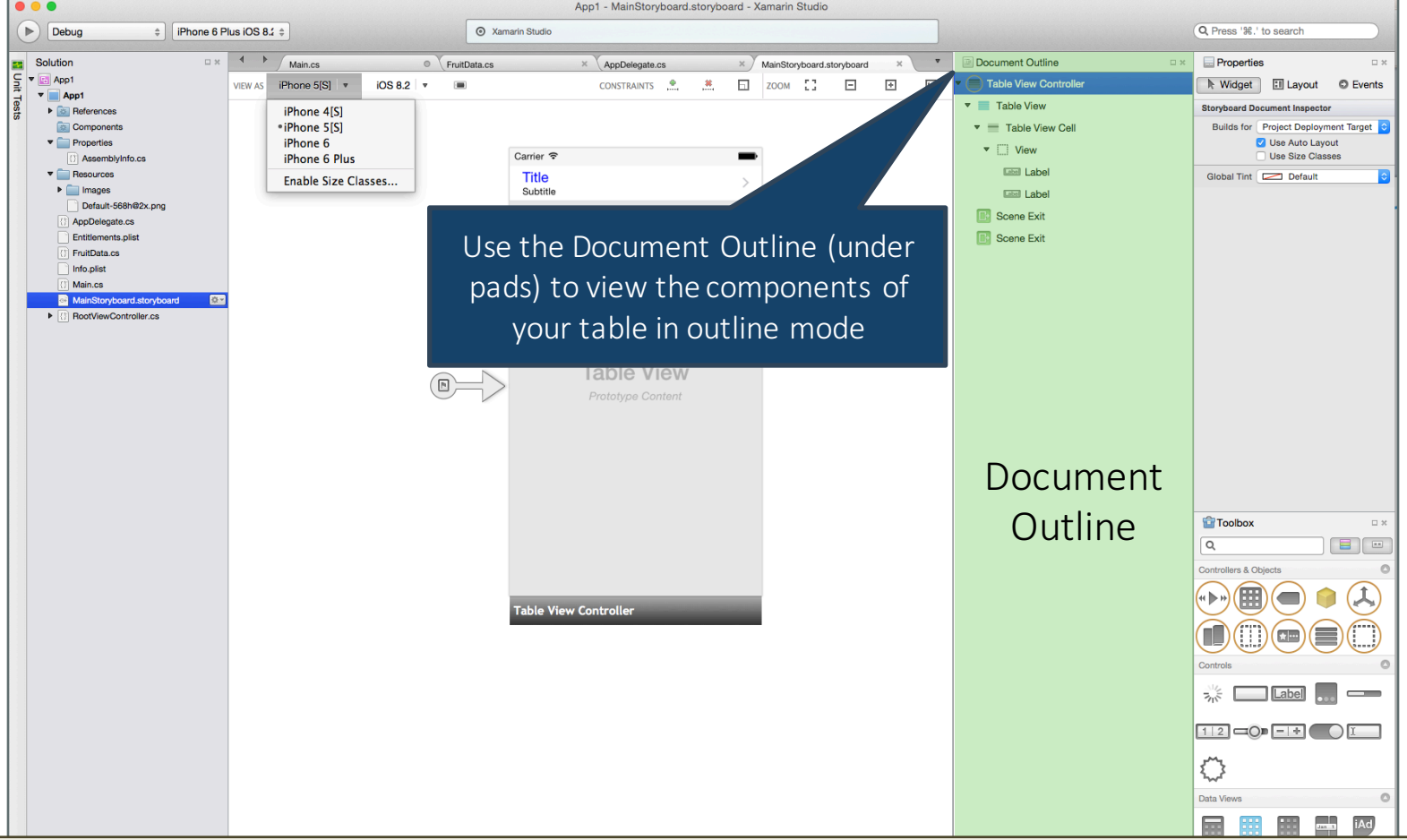
- You can create and customize many parts of the table view in the designer



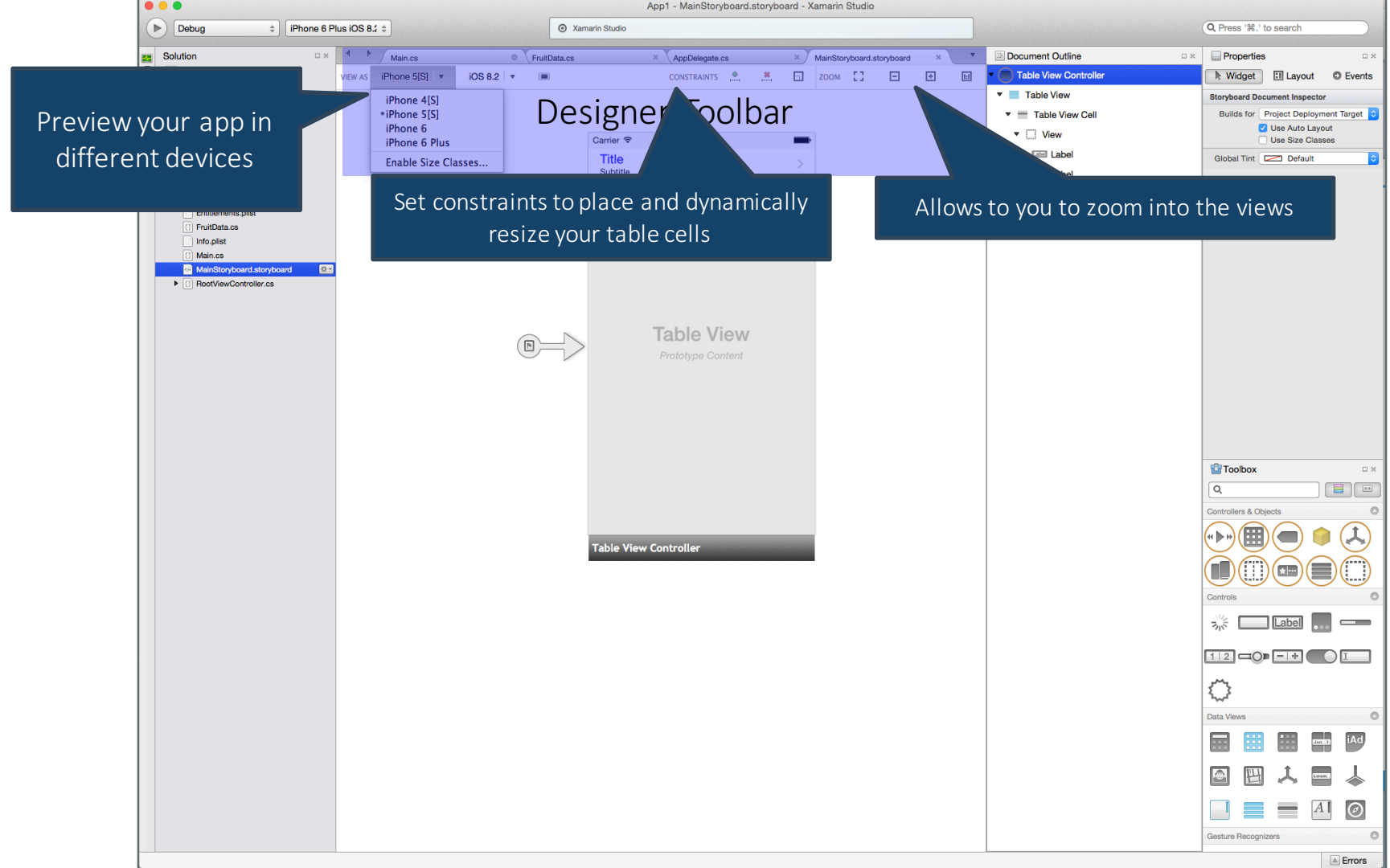

App1 - MainStoryboard.storyboard - Xamarin Studio
Debug
iPhone 6 Plus iOS 8.1
Xamarin Studio
Press '⌘.' to search
Solution
App1
References
Components
Properties
AssemblyInfo.cs
Resources
Images
Default-568h@2x.png
AppDelegate.cs
Entitlements.plist
FruitData.cs
Info.plist
Main.cs
MainStoryboard.storyboard
RootViewController.cs
Unit Tests
Main.cs
FruitData.cs
AppDelegate.cs
MainStoryboard.storyboard
VIEW AS
iPhone 5[S]
iOS 8.2
CONSTRAINTS
ZOOM
iPhone 4[S]
iPhone 5[S]
iPhone 6
iPhone 6 Plus
Enable Size Classes...
Carrier
Title
Subtitle
Table View
Prototype Content
Table View Controller
Storyboard workspace
Document Outline
Table View Controller
Table View
Table View Cell
View
Label
Label
Scene Exit
Scene Exit
Properties
Widget
Layout
Events
Storyboard Document Inspector
Builds for
Project Deployment Target
Use Auto Layout
Use Size Classes
Global Tint
Default
Toolbox
Controllers & Objects
Controls
Label
Data Views
Gesture Recognizers
Errors


App1 - MainStoryboard.storyboard - Xamarin Studio
Debug
iPhone 6 Plus iOS 8.1
Xamarin Studio
Press '⌘.' to search
Solution
App1
References
Components
Properties
AssemblyInfo.cs
Resources
Images
Default-568h@2x.png
AppDelegate.cs
Entitlements.plist
FruitData.cs
Info.plist
Main.cs
MainStoryboard.storyboard
RootViewController.cs
Unit Tests
Main.cs
FruitData.cs
AppDelegate.cs
MainStoryboard.storyboard
VIEW AS
iPhone 5[S]
iOS 8.2
CONSTRAINTS
ZOOM
iPhone 4[S]
iPhone 5[S]
iPhone 6
iPhone 6 Plus
Enable Size Classes...
Carrier
Title
Subtitle
Table View
Prototype Content
Table View Controller
Document Outline
Table View Controller
Table View
Properties
Widget
Layout
Events
Storyboard Document Inspector
Builds for
Project Deployment Target
Use Auto Layout
Use Size Classes
Global Tint
Default
Property Pane
The Properties pane allows you to change properties and settings on the selected control
Toolbox
Controllers & Objects
Controls
Label
Data Views
Gesture Recognizers
Errors


App1 - MainStoryboard.storyboard - Xamarin Studio
Debug
iPhone 6 Plus iOS 8.1
Xamarin Studio
Press '⌘.' to search
Solution
App1
References
Components
Properties
AssemblyInfo.cs
Resources
Images
Default-568h@2x.png
AppDelegate.cs
Entitlements.plist
FruitData.cs
Info.plist
Main.cs
MainStoryboard.storyboard
RootViewController.cs
Unit Tests
Main.cs
FruitData.cs
AppDelegate.cs
MainStoryboard.storyboard
VIEW AS
iPhone 5[S]
iOS 8.2
CONSTRAINTS
ZOOM
iPhone 4[S]
iPhone 5[S]
iPhone 6
iPhone 6 Plus
Enable Size Classes...
Carrier
Title
Subtitle
Table View
Prototype Content
Table View Controller
Document Outline
Table View Controller
Table View
Table View Cell
View
Label
Label
Scene Exit
Scene Exit
Properties
Widget
Layout
Events
Storyboard Document Inspector
Builds for
Project Deployment Target
Use Auto Layout
Use Size Classes
Global Tint
Default
Toolbox
Controllers & Objects
Controls
Label
Data Views
Gesture Recognizers
Errors
The Toolbox contains controllers, objects, and views for you to drag onto the storyboard
Toolbox

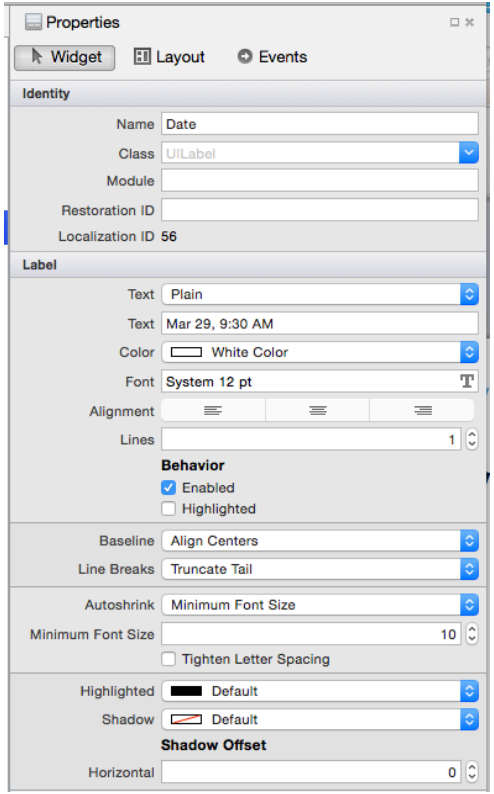


Adding Storyboard IDs to your views makes its much easier to identify specific views when listed in the document outline.

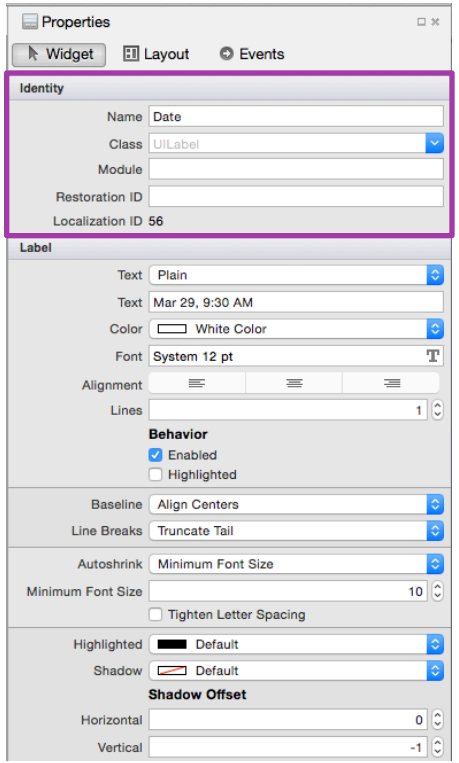

Preview your app in different devices
Set constraints to place and dynamically resize your table cells
Allows to you to zoom into the views
App1 - MainStoryboard.storyboard - Xamarin Studio
Debug
iPhone 6 Plus iOS 8.2
Xamarin Studio
Press '⌘.' to search
Solution
Main.cs
FruitData.cs
AppDelegate.cs
MainStoryboard.storyboard
VIEW AS
iPhone 5[S]
iOS 8.2
CONSTRAINTS
ZOOM
iPhone 4[S]
iPhone 5[S]
iPhone 6
iPhone 6 Plus
Enable Size Classes...
Designer Toolbar
Carrier
Title
Subtitle
Table View
Prototype Content
Table View Controller
Document Outline
Table View Controller
Table View
Table View Cell
View
Label
Properties
Widget
Layout
Events
Storyboard Document Inspector
Builds for
Project Deployment Target
Use Auto Layout
Use Size Classes
Global Tint
Default
Toolbox
Controllers & Objects
Controls
Label
Data Views
Gesture Recognizers
Errors
FruitData.cs
Info.plist
Main.cs
MainStoryboard.storyboard
RootViewController.cs

Style the custom cell in the designer

- Once you have dragged your views onto the storyboard, use the properties pane to style them

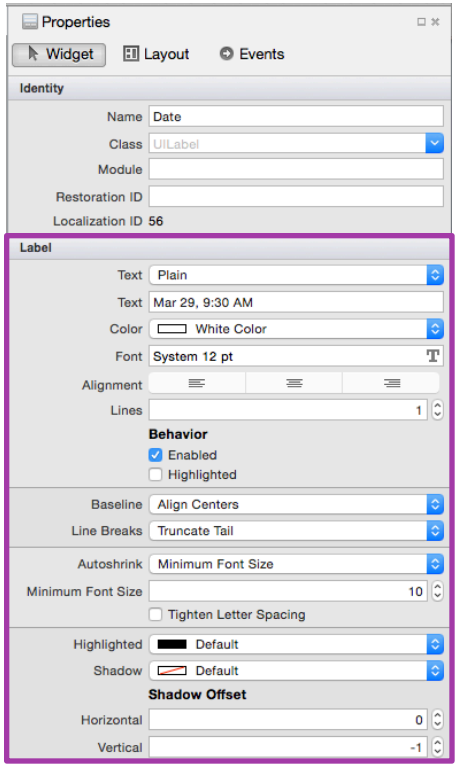


Changing properties



- Use the **Identity** tab to assign a **Name** and **Class** to your UI element
  - **Name** assigns a code-behind element to the control
  - **Class** creates the code-behind class used to customize the cell definition

Changing properties



- Second group in the property pane allows you to change the properties that are unique to the selected UI element
  - Text
  - Color
  - Font
  - …

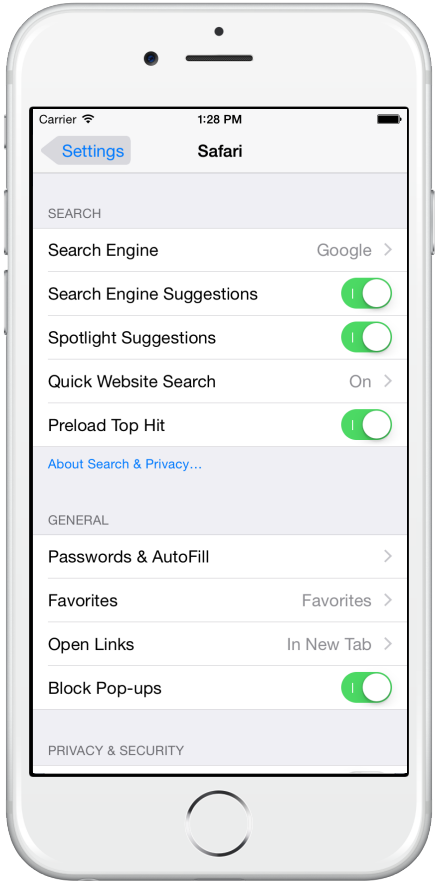
Two types of cells

- Designer supports two types of table view cell designs

Static table view cells

- When you want to display pre-defined data which does not change, you can use **static cells**, these are:
  - hard coded into the table view design
  - not assigned a reuse identifier
  - not populated by a table view source

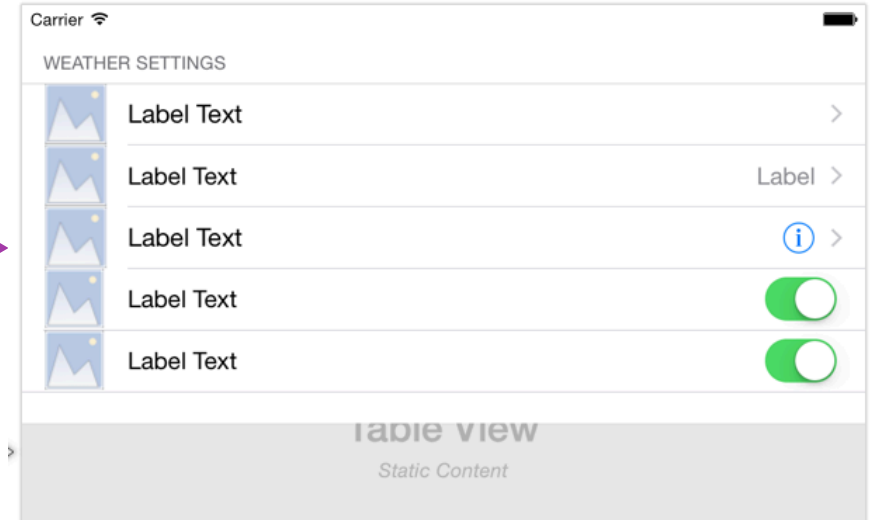
- Typically used when the design and data of the cell is completely known at compile time



Static cells in the designer

- We can create and populate Static prototype cells using the designer

Static Prototype cell



Populating static cells

- To update the contents of the static cells at runtime, name the cells in the designer and access the child views in the view controller's code behind

```
partial class SettingsViewController : UITableViewController
{
    ...

    public override void ViewDidLoad ()
    {
        CellDefaultCity.DetailTextLabel.Text = "Vancouver";
        ...
    }
}
```

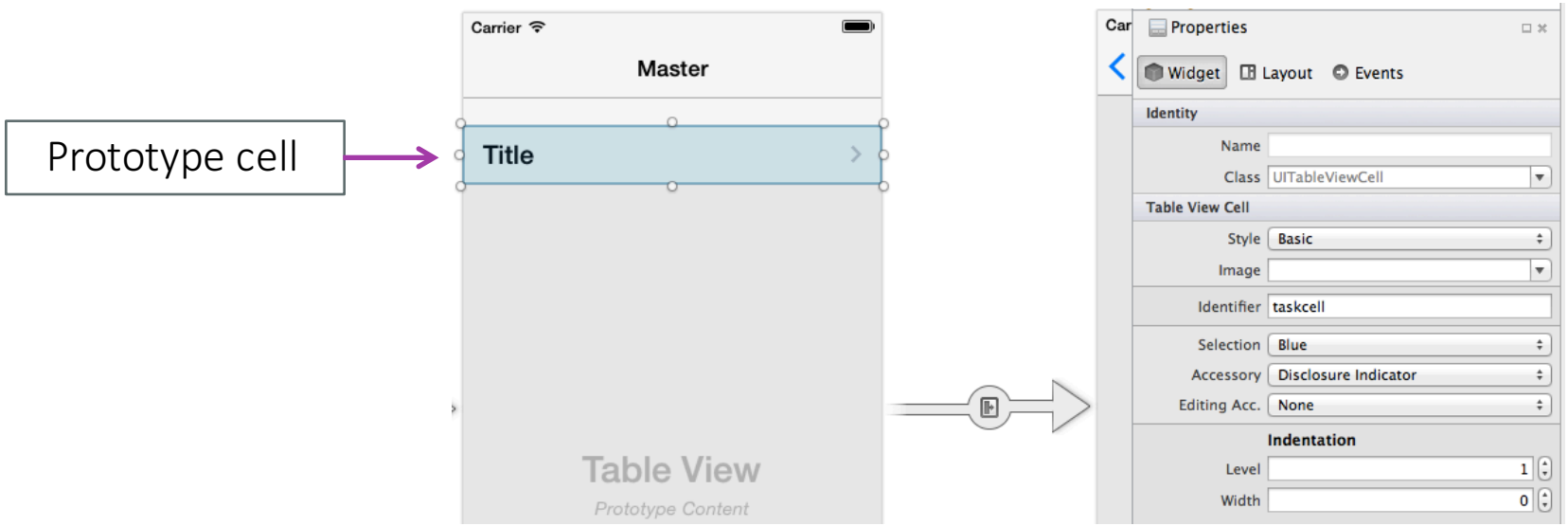
Group Exercise

Create static cells in the Designer

Xamarin University

Dynamic prototype cells

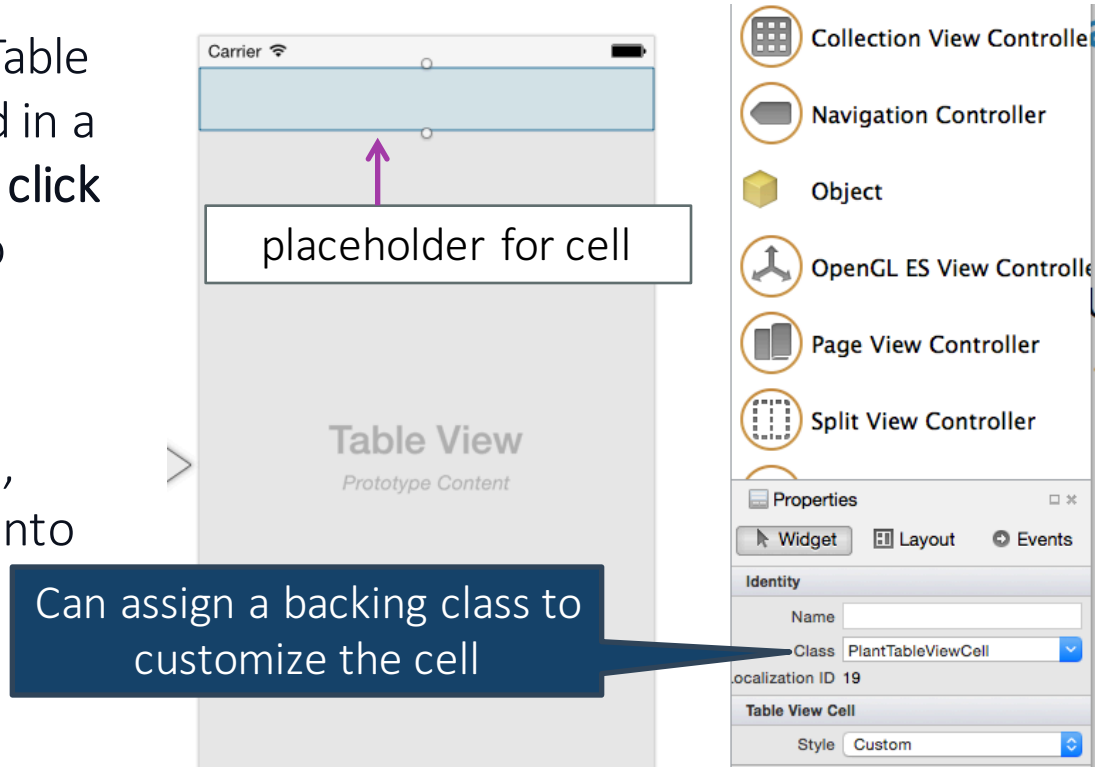
- Custom cells which are populated with runtime data are represented in the designer using a *dynamic prototype* cell definition



Designing a cell in the designer

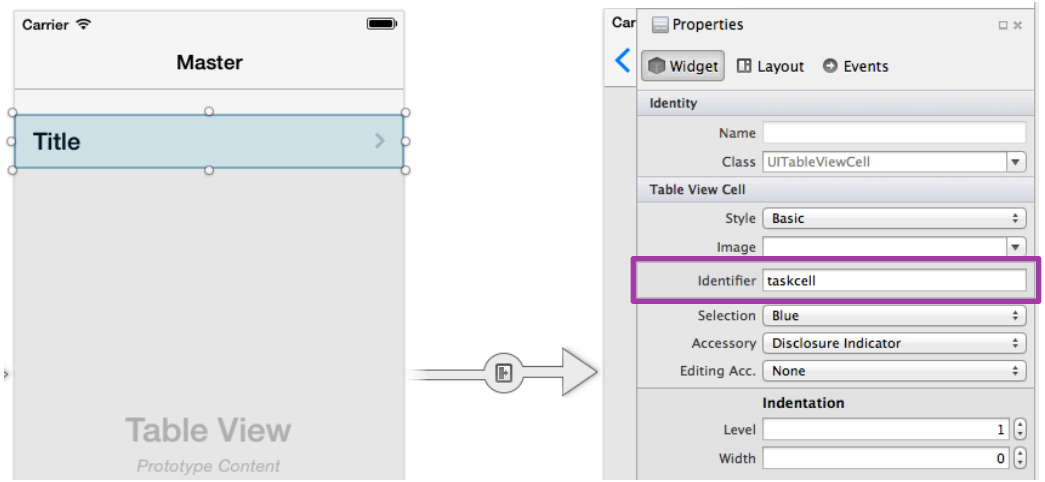
- When the Table View or Table View Controller is created in a Storyboard, then you can **click on the cell placeholder** to adjust the design

- While cell design is active, drag and drop sub-views into the cell container



Set the reuse Identifier

- Set the reuse Identifier when using dynamic prototype cells to enable cell reuse

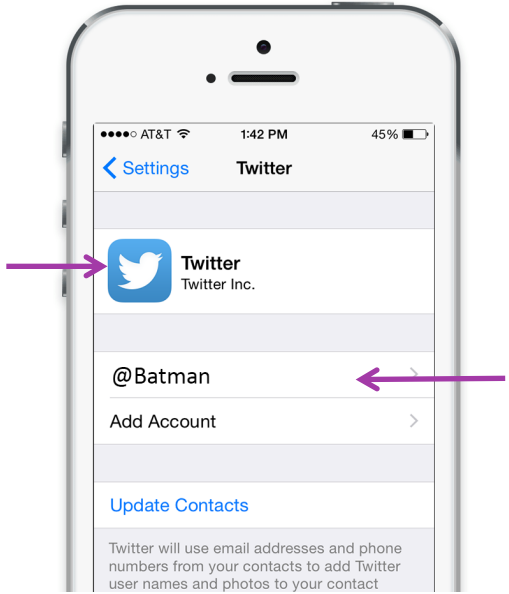


Make sure the reuse identifier set in the storyboard matches the ID used in the `GetCell` method in your table view controller code-behind

Mixing static and dynamic data?

- iOS does not allow you to mix static and dynamic prototype cells

Static cell content
(never changes)



Dynamic cell content
(decided at runtime)

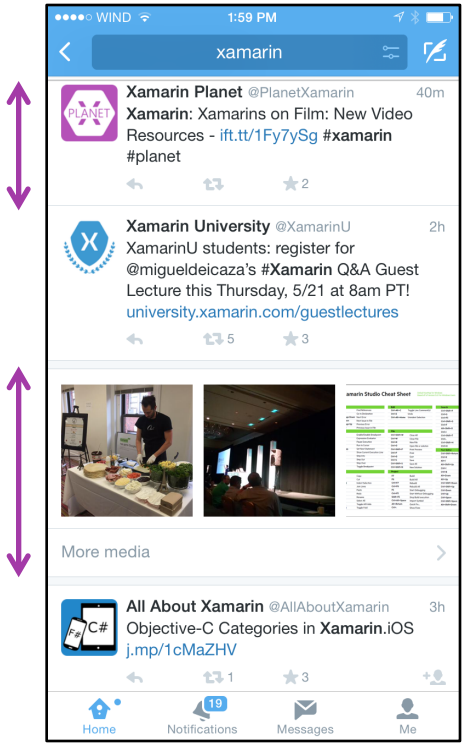
Simulating static content with dynamic cells

- Dynamic prototype cells can behave like static cells when the cell is returned without content from `GetCell`

```
public override  UITableViewCell GetCell (UITableView tableView,
                                          NSIndexPath indexPath)
{
    cell = tableView.DequeueReusableCell (CELL_ID);
    return cell;
}
```

Working with self sizing rows

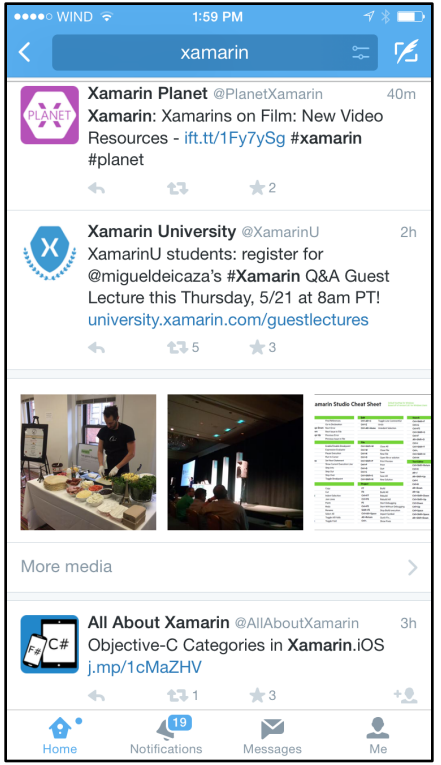
- iOS traditionally uses a fixed size for each row – where every row is always the same height
- What if the row size changes on a row-by-row basis?



Self sizing rows (iOS8+)

- To enable self sizing rows in iOS programmatically, first set the row height to **`AutomaticDimension`**

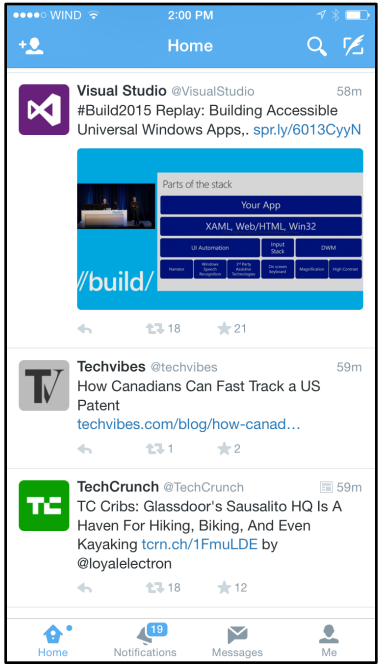
```
public class MessagesTableViewController
        : UITableViewController
{
    public MyTableViewController()
    {
        TableView.RowHeight =
          UITableView.AutomaticDimension;
        ...
    }
}
```



Self sizing rows (iOS8+)

- `EstimatedRowHeight` should be set to an approximate height for the cells

```
public class MessagesTableViewController
        : UITableViewController
{
    public MyTableViewController()
    {
        TableView.RowHeight =
              UITableView.AutomaticDimension;
        TableView.EstimatedRowHeight = 80;
        ...
    }
}
```



Flash Quiz

Xamarin University

Flash Quiz

① Cells which contain pre-defined data are referred to as:

a) Prototype cells

b) Dynamic cells

c) Static cells

Flash Quiz

① Cells which contain pre-defined data are referred to as:

a) Prototype cells

b) Dynamic cells

c) **Static cells**

Flash Quiz

② When creating custom cells, the designer can do everything custom code can

a) True

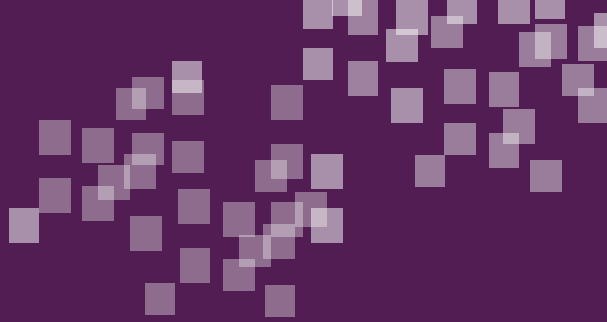
b) False

Flash Quiz

② When creating custom cells, the designer can do everything custom code can

a) True

b) False



Individual Exercise

Create a prototype table view cell using the designer

Xamarin University

Summary

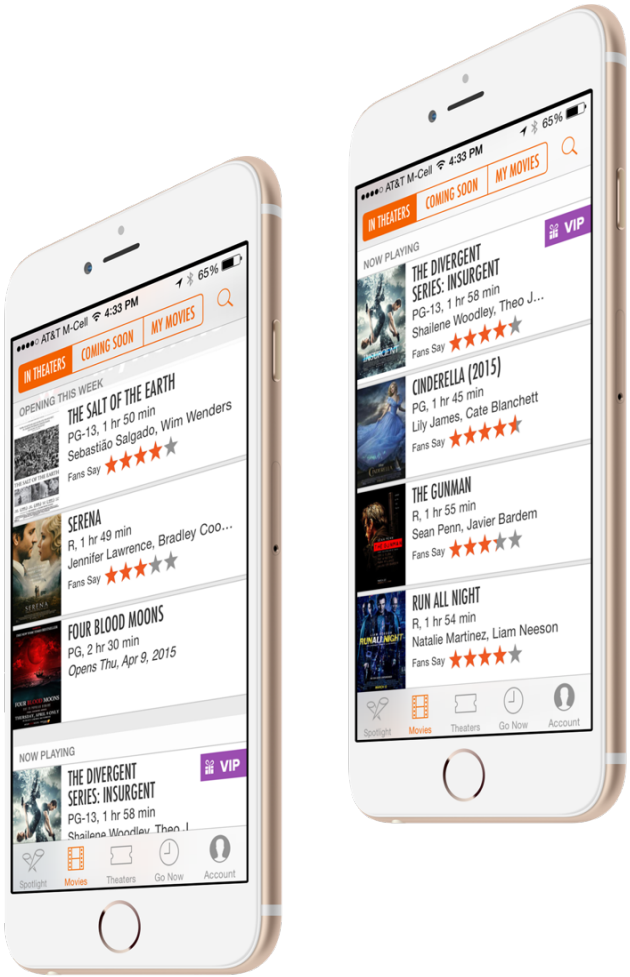
1. Walkthrough the designer
2. Compare static and dynamic cells
3. Design a custom cell using the designer

Group data in the Table View

Xamarin University

Tasks

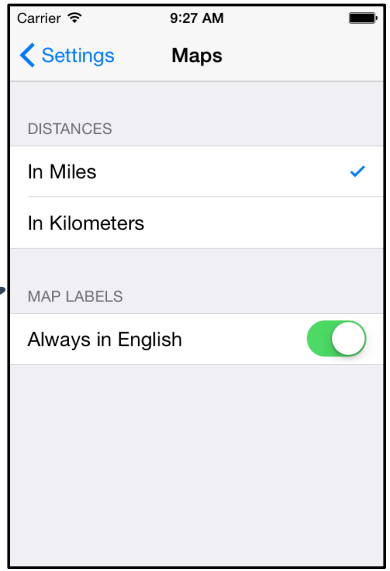
1. Compare plain vs. grouped table views
2. Create an index
3. Add headers and footers
4. Customize headers and footers



Organizing the Table View data

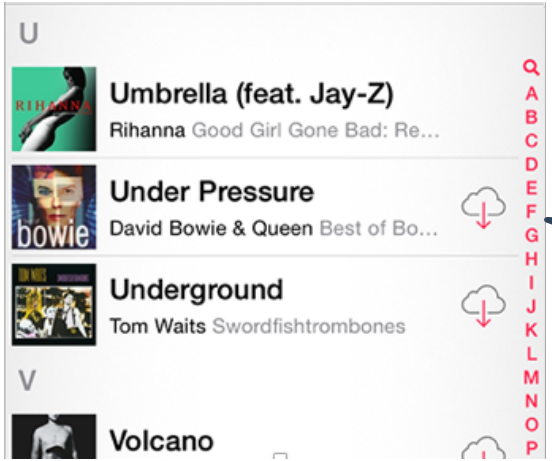
- The Table View has several built-in features which can be used to organize the data display and make it more accessible to the user

Table view cells can be organized into logical groups with headers



Organizing the Table View data

- The Table View has several built-in features which can be used to organize the data display and make it more accessible to the user



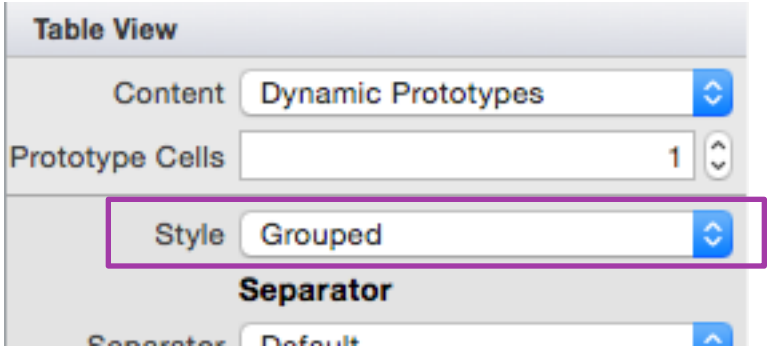
Can add an index for quicker navigation

Setting the table style

- Grouping is enabled on a **`UITableView`** by setting the `Style` property, either in code, or in the designer

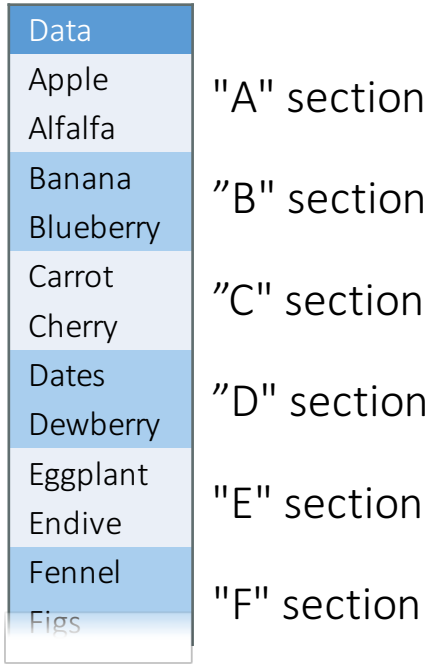
```
var tblView = new UITableView (frame: rc,
                               style: UITableViewStyle.Grouped);
```

The style must be set when the table view is created and cannot be updated



What is a Section?

- A *section* is a logical group in a list of data – the Table View displays each section in it's own group
- You decide what the sections will be based on your data and it's organization



Section the data

- Sectioning organizes the data into logical groups (i.e. alphabetically)

Data	List position	Section index	Section label
Apple	0	0	A
Alfalfa	1	0	A
Banana	2	0	B
Cherry	3	0	C
Dates	4	1	D
Eggplant	5	1	E
Figs	6	1	F

this is commonly stored in a `Dictionary<K,V>` or an `IGrouping`

A	C	D	E	F

Providing grouped data to the Table View

- The Table View Source must implement two additional methods to support a grouped Table View

`NumberOfSections`

`RowsInSection`

NumberOfSections

- The `NumberOfSections` method should return the number of groups to display – e.g. how many keys are in the dictionary, or how many partitions the data is split into

```
Dictionary<string, string[]> groupedFruit;

public override nint NumberOfSections (UITableView tableView)
{
    return groupedFruit.Keys.Length; // # of groups
}
```

RowsInSection

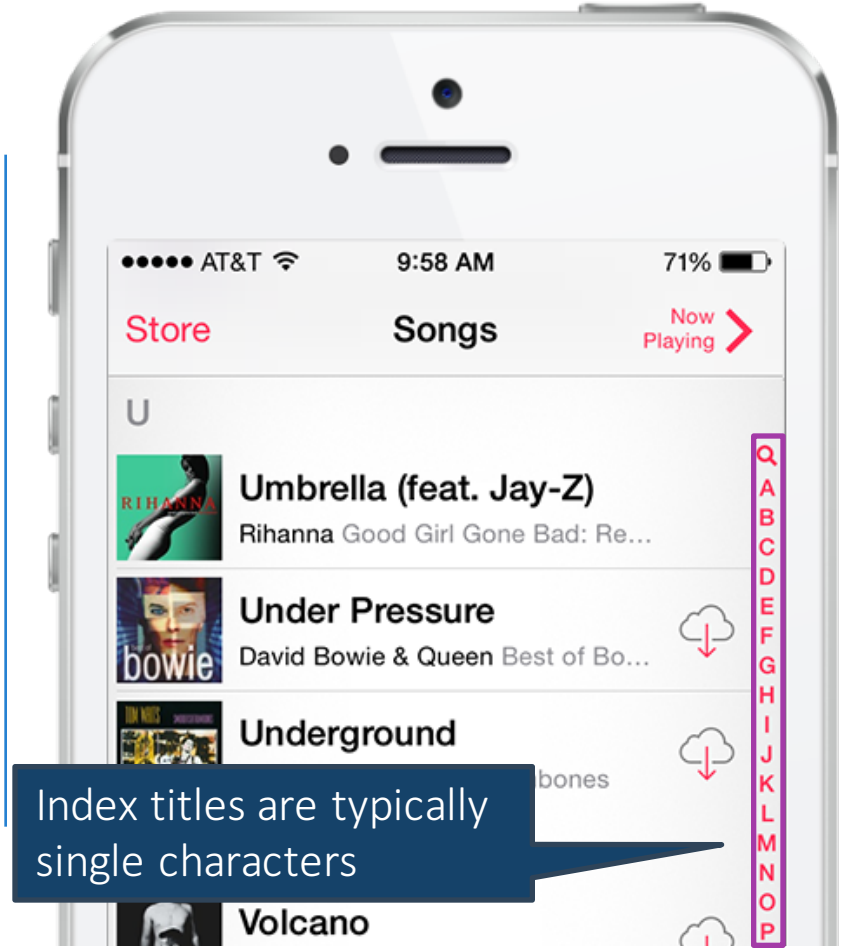
- The **`RowsInSection`** method identifies the number of rows (items) in a given section (group)

```
Dictionary<string, string[]> groupedFruit;

public override nint RowsInSection (UITableView tableview,
                                    nint section)
{
    // # of fruits in group
    keys = indexedTableItems.Keys.ToArray ();
    return groupedFruit.[keys[section]].Count();
}
```

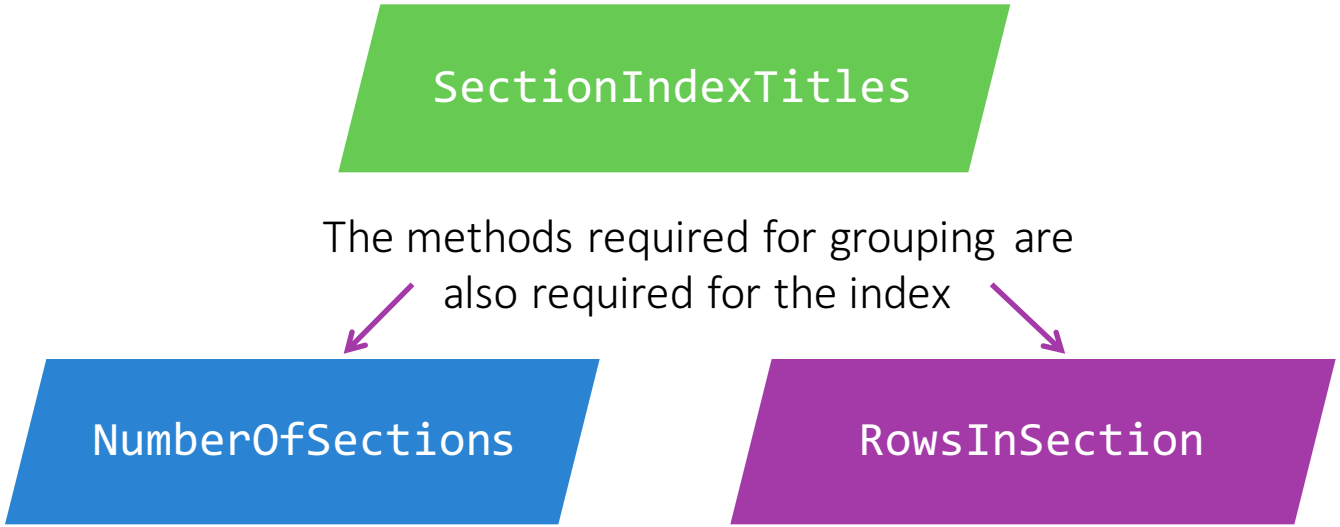
Creating an index

- An *index* can be added to the right side of a Table View for quicker navigation



Populating the index

- To populate the index we need to override the `SectionIndexTitles` in the table view controller



SectionIndexTitle

- The **`SectionIndexTitles`** method returns the array of strings that will be used to display the index

```
public override string[] SectionIndexTitles(UITableView tableView)
{
    var items = groupedFruit.Keys.ToList();
    items.Insert(0, UITableView.IndexSearch);
    return items.ToArray();
}
```

adds magnifying glass when using search

Flash Quiz

Xamarin University

Flash Quiz

① An index can be used in which type of table view?

a) Plain

b) Grouped

c) Both

Flash Quiz

① An index can be used in which type of table view?

a) Plain

b) Grouped

c) **Both**

Flash Quiz

② You must set the Table View style to `Grouped` if `NumberOfSections` returns a value greater than 1

a) True

b) False

Flash Quiz

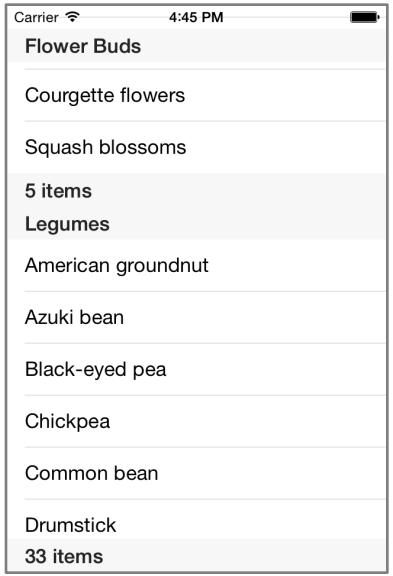
② You must set the Table View style to `Grouped` if `NumberOfSections` returns a value greater than 1

a) True

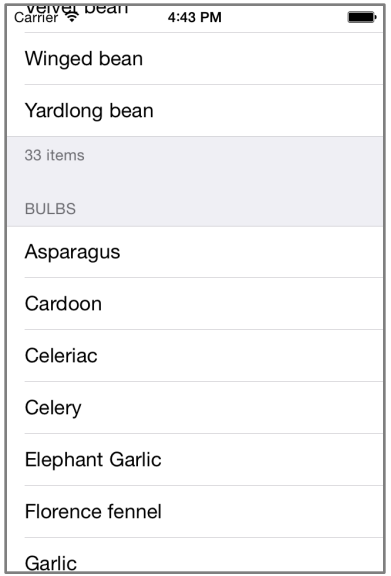
b) **False**

Headers and footers

- Table View supports both headers and footers on grouped sections



Plain



Grouped

Adding Headers and footers

- Displaying headers and footers requires additional methods

TitleForHeader

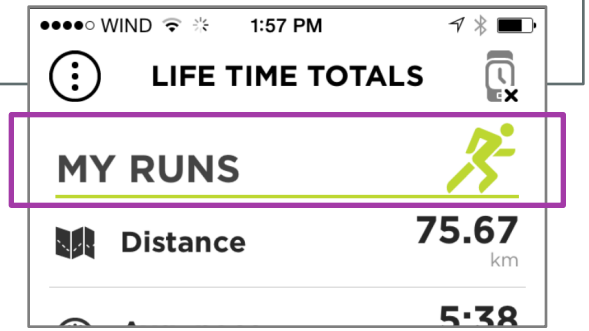
- `TitleForHeader` should return the string to show for the given section

```
public override string TitleForHeader (
     UITableView tableView, nint section)
{
    return keys[section];
}
```

Customize the header

- You can customize the view for the header by using the `GetViewForHeader` method on the Table View source class

```
public override UIView GetViewForHeader (UITableView tableView,
                                         nint section) {
    if(section == 0)
        return BuildCustomHeaderView ("MY RUNS", "runner.png");
    ...
}
```



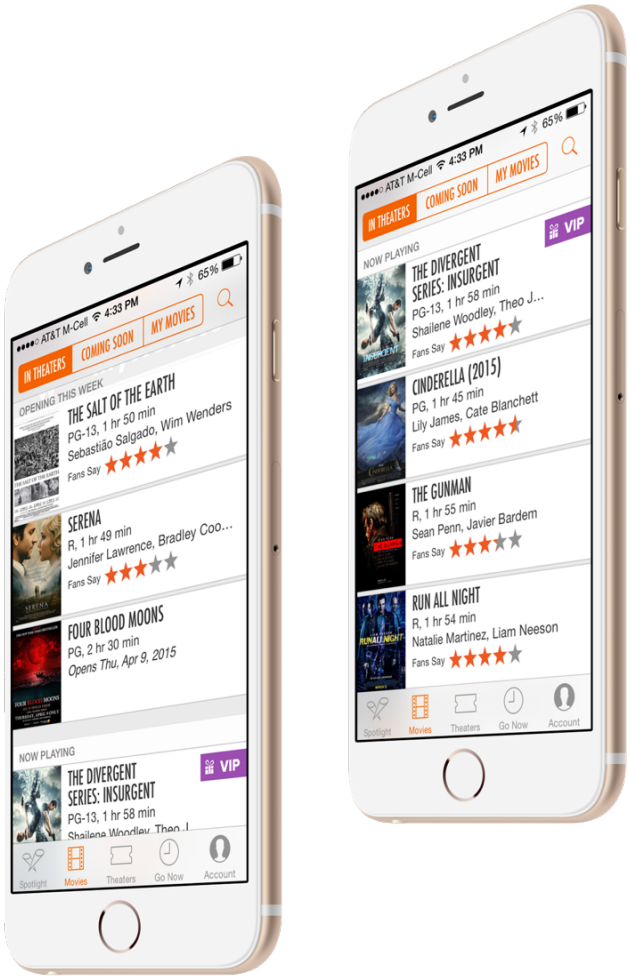
Individual Exercise

Create a grouped table with an index

Xamarin University

Summary

1. Compare plain vs. grouped table views
2. Create an index
3. Add headers and footers
4. Customize headers and footers



Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile