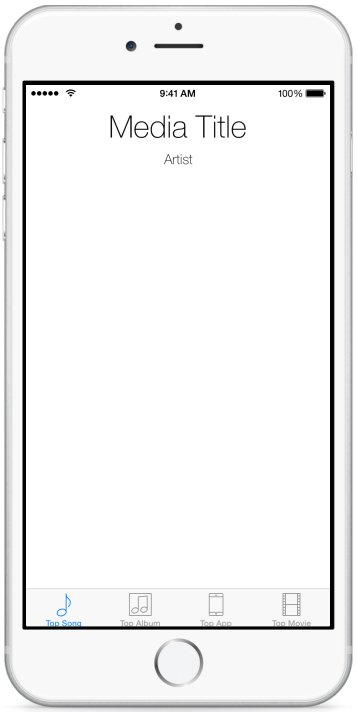
RSSReader
Lesson 4



Description

Add labels to the controller views, and bind the view controllers to a custom view controller implementation.

Learning Outcomes

- Analyze app behavior to identify opportunities to encapsulate common code reuse.
- Practice adding interface components to a view, and establishing auto layout constraints.
- Practice adding a new controller class to an Xcode project, and binding it to view controllers in a storyboard.
- Practice declaring and connecting controller outlet properties.
- Observe how ARC declaration modifiers allude to object ownership within a property declaration.
- Practice modifying outlet properties with controller code.

Vocabulary

class definition	image view	auto layout constraint
identity inspector	storyboard scene	outlet
property	declaration modifier	implicitly unwrapped optional

Materials

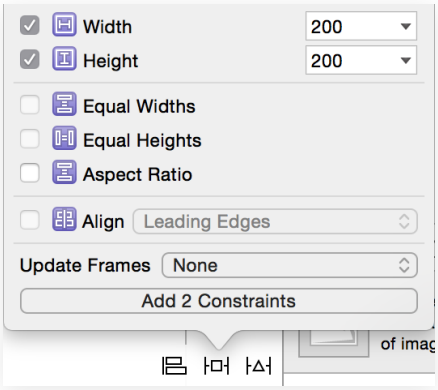
- **RSSReader Lesson 4** Xcode project

Opening

How might we build our four views while minimizing repetitive code?

Agenda

- Discuss how each view controller will present the title, artist, and image of a top song, album, app or movie.
- Discuss the possibility of implementing individual TopSong, TopAlbum, TopApp and TopMovie controller classes, and how each controller definition would perform the same work but with different data.
- Using Interface Builder and the Object Library (⌥⌘L), drag a Text Label onto the Top Song view, change the contents of the label to **Title**, and use the Attributes Inspector (⌥⌘4) to customize the text alignment and font. Control-drag upwards from the label to the view to add constraints for the vertical spacing and horizontal alignment; and use the menu item *Editor* > *Resolve Auto Layout Issues* > *Update Frames* (⌥⌘=) to match the size constraint.
- Using Interface Builder and the Object Library (⌥⌘L), drag a second text label onto the Top Song view, change the contents of the label to **Artist**, and use the Attributes Inspector (⌥⌘4) to customize the alignment and font. Control-drag from the Artist label to the Title label to establish a vertical spacing constraint; control-drag from the Artist label to the view to add a horizontal alignment constraint; and use the menu item *Editor* > *Resolve Auto Layout Issues* > *Update Frames* (⌥⌘=) to match the size constraint.
- Using Interface Builder and the Object Library (⌥⌘L), drag an Image View onto the Top Song view, position the image centered, below the Artist label. Use the Size Inspector (⌥⌘5) to set the height and width of the image view to 200x200.
- Add constraints to the image view by Control-dragging from the image view to the Artist label to add a vertical spacing constraint; Control-dragging downwards from the label to the main view to add a horizontal alignment constraint; and use the Pin control (⊢□⊣) to add constraints for the width and height.



- Run the app (⌘R) observe the labels appear, and discuss how the image view is empty because no image has been assigned to it for display.
- Select the three interface elements by ⌘-clicking each of them, copy and paste the three elements to the Top Album view, and center the group of elements at the top of the view.
- Discuss how constraints between the elements are preserved, but the constraints relative to the containing view must be recreated.
- Add constraints by Control-dragging upwards from the Title label to the containing view to add a vertical spacing constraint and a horizontal alignment constraint; Control-drag upwards from the Artist label to the containing view to establish a horizontal alignment constraint; and Control-drag downwards from the image view to the containing view to add a horizontal alignment constraint.
- Run the app (⌘R), interact with the Top Song and Top Album tabs, and observe the labels appear.
- Duplicate the copying and pasting of the labels and image view to the Top App and Top Movie scenes, and establish the appropriate constraints.
- Run the app (⌘R), interact with all of the tabs, and observe the labels appear.
- Add a new class (⌘N) to the project called `TopMediaController` that extends `UIViewController`.

```
import UIKit

class TopMediaController: UIViewController {

   override func viewDidLoad() {
      super.viewDidLoad()
   }

}
```

- Using Interface Builder, select each view controller, and use the Identity Inspector (⌥⌘3) to set the Class attribute to `TopMediaController`.
- Discuss how one controller class will encapsulate the retrieval of RSS data, with four separate instances of the controller that will update their respective views.
- Within the `TopMediaController` implementation declare three outlet properties for the user interface components.

```
class TopMediaController: UIViewController {

   @IBOutlet weak var titleLabel: UILabel!
   @IBOutlet weak var artistLabel: UILabel!
   @IBOutlet weak var imageView: UIImageView!
   ...
```

- Discuss that the properties use the `weak` declaration modifier because the parent view, and not the controller itself, "owns" the labels and image view; and that the properties are implicitly unwrapped optional types, because the controller does not assign values to the properties during initialization.
- Using Interface Builder and the Document Outline (), expand the contents of each scene, and Control drag connections from each controller to their respective interface components. For example, Control-drag a connection from the Top Song controller to its Title label, and select the `titleLabel` outlet; demonstrate repeating the action with each controller and its interface components.
- In the `TopMediaController` implementation, update `viewDidLoad` to modify the `titleLabel` `text`.

```
override func viewDidLoad() {
   super viewDidLoad()
   titleLabel.text ="Media Title"
}
```

- Run the app (⌘R), interact with the tabs, and observe how each Title label appears with the contents **Media Title**.

Closing

Why do each of our views display the same title? If we want to use code to change the label text differently in each view, do we need four different controller classes?

Modifications and Extensions

- Use Interface Builder to create a container view that holds the text labels and image view, and rebuild the interface by re-establishing constraints for the interface components relative to the container view, and between the container view and the main view in each scene. Copy and paste the container view to the different scenes, and determine what constraints are preserved, and what constraints must be created within each scene.
- Rather than using Interface Builder to create repetitive views, encapsulate the interface in your own `UIView` child class, representing a container for the labels and image view. Instantiate your custom view in the controller `viewDidLoad` method, and compare your approach with one that just uses Interface Builder.

Resources

Cocoa Application Competencies for iOS: Storyboard https://developer.apple.com/library/ios/documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html

View Controller Catalog for iOS: Tab Bar Controllers https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/TabBarControllers.html

Xcode Overview: Build a User Interface https://developer.apple.com/library/prerelease/ios/documentation/ToolsLanguages/Conceptual/Xcode_Overview/edit_user_interface.html

Auto Layout Guide: Working with Constraints in Interface Builder https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/AutolayoutPG/WorkingwithConstraints/WorkingwithConstraints.html

Interface Builder Help: Creating an Outlet Connection https://developer.apple.com/library/ios/recipes/xcode_help-IB_connections/chapters/CreatingOutlet.html

The Swift Programming Language: Classes and Structures https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ClassesAndStructures.html

The Swift Programming Language: Properties https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Properties.html

The Swift Programming Language: Weak References https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/Swift_Programming_Language/AutomaticReferenceCounting.html#//apple_ref/doc/uid/TP40014097-CH20-ID53