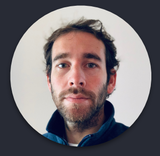
UE LI 385

SORBONNE UNIVERSITÉ

Introduction to iOS development with Swift

Lesson 3



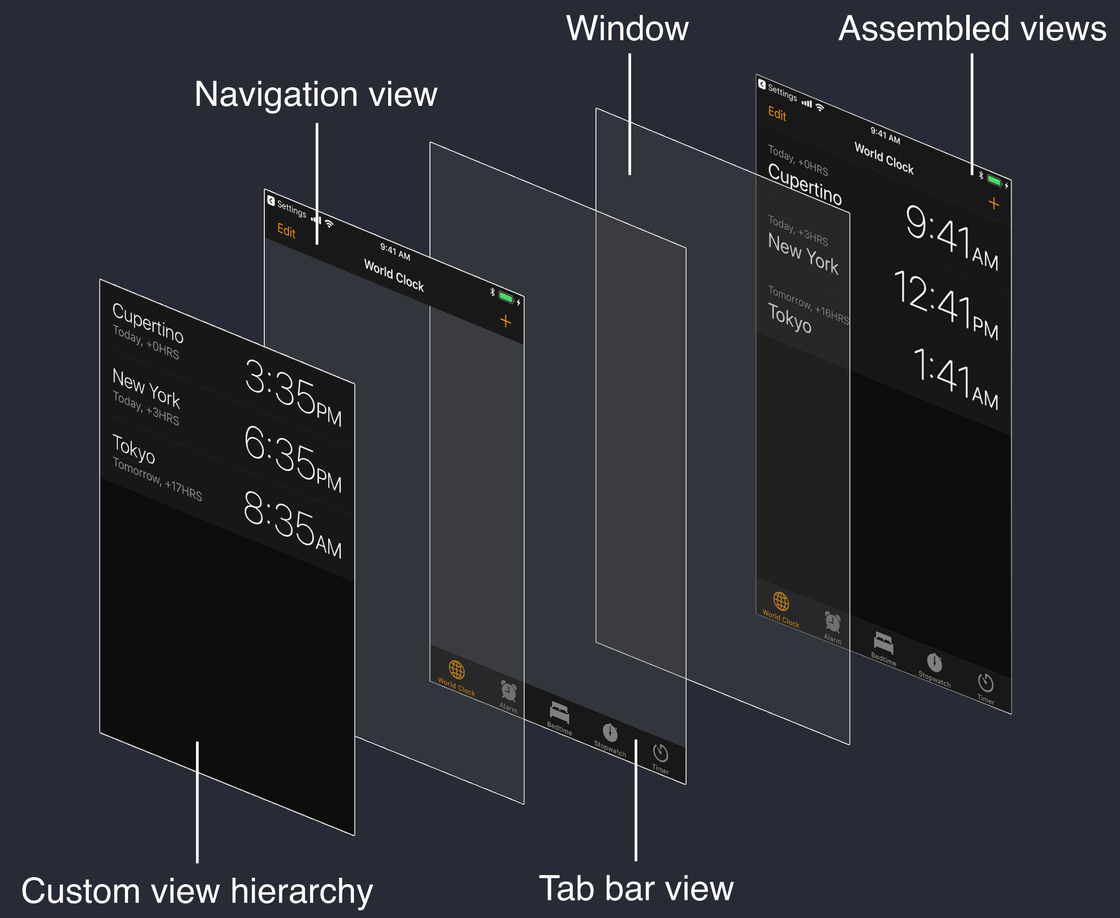
Adrien Humilière
Brut.

adhumi+dant@gmail.com

- Introduction to UIKit
- Auto layout & stack views
- Application lifecycle
- Model — View — Controller
- View controller lifecycle
- Segues & UINavigationController
- UITabBarController
  UIScrollView
  UITableView
- Building simple workflows
- Saving data

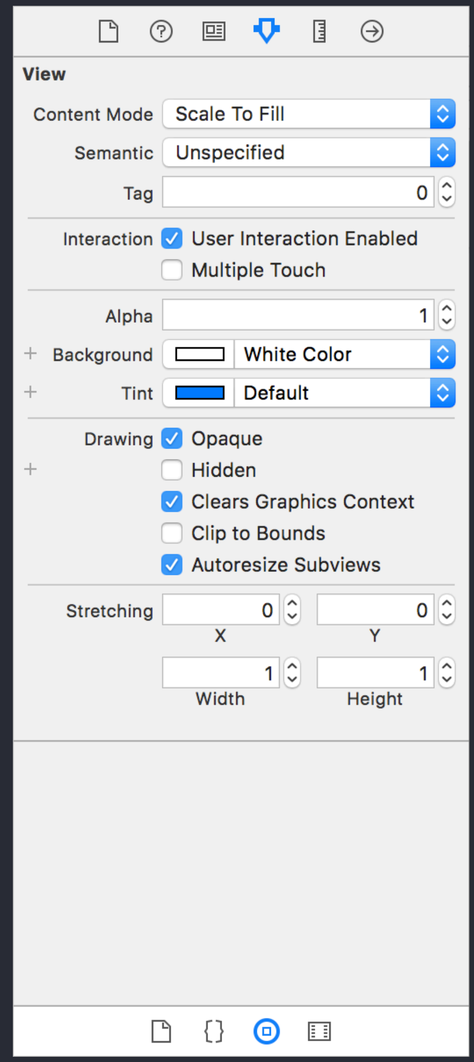
Introduction to UIKit

Common system views

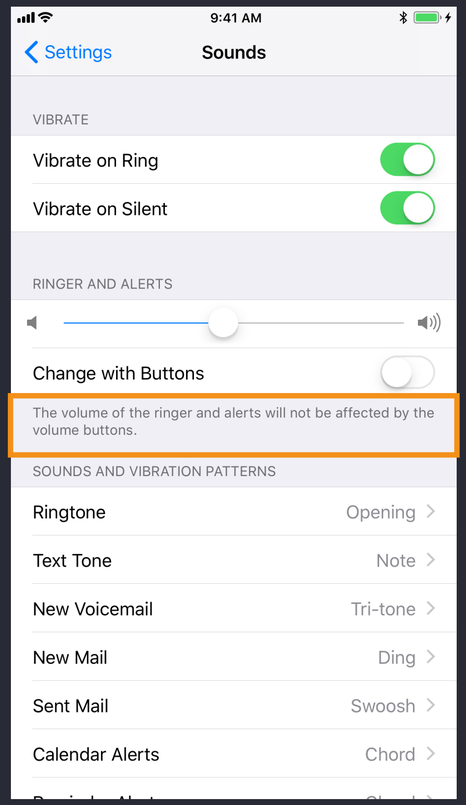


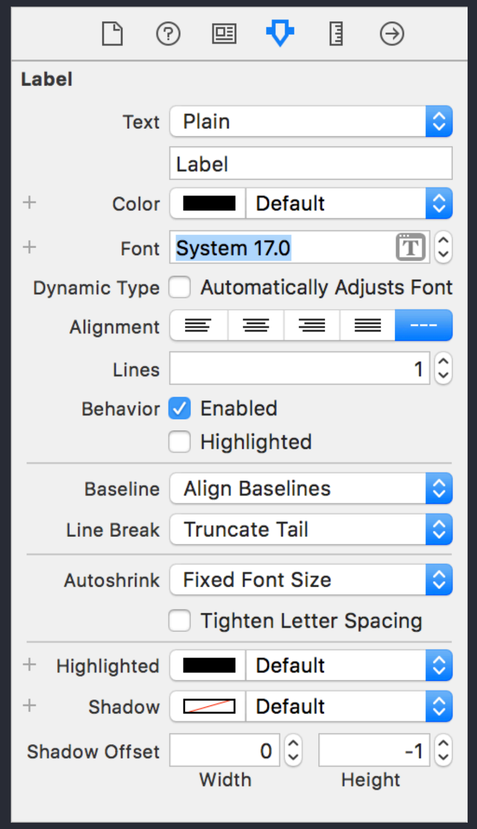
Common system views – UIView

Size	Width and height of the view
Position	Position of the view onscreen
Alpha	Transparency of the view
Background Color	Background color that will be displayed
Tag	Integer that you can use to identify view objects



Label – UILabel





Label – UILabel

Image – UIImageView

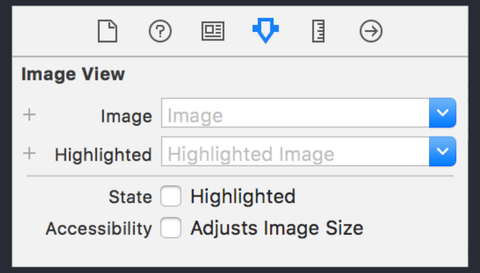
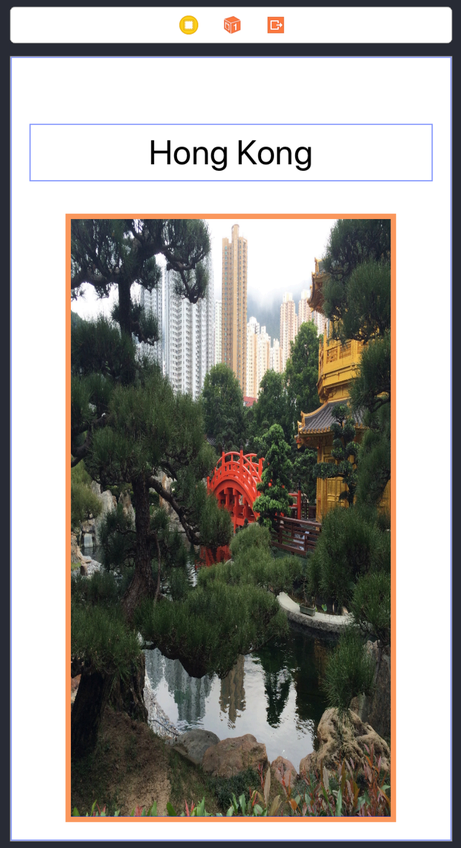
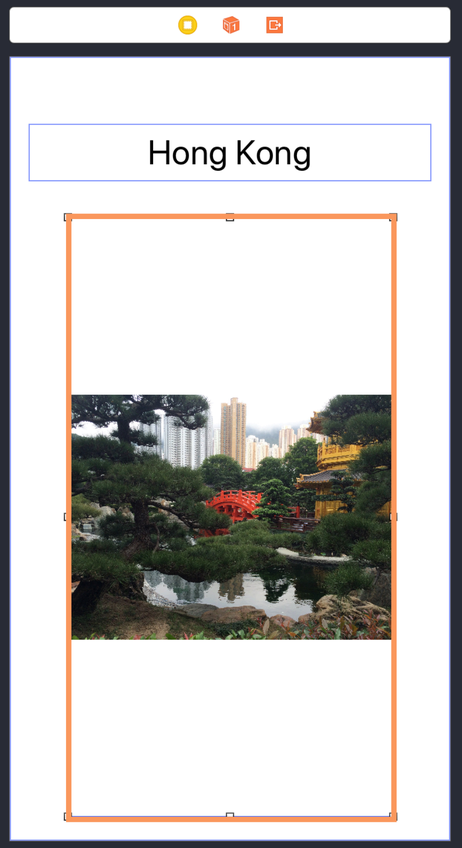


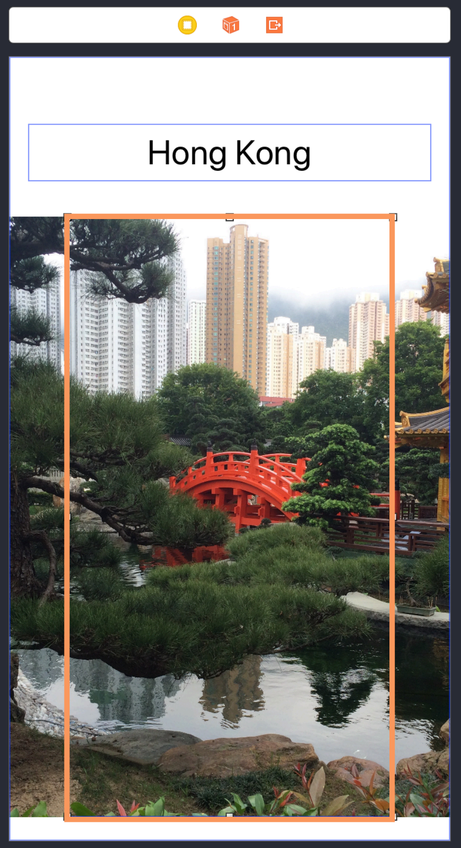
Image – UIImageView



Scale To Fill

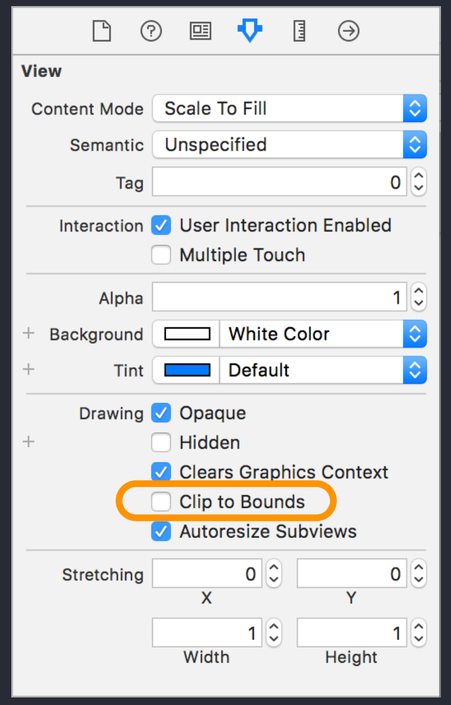


Aspect Fit

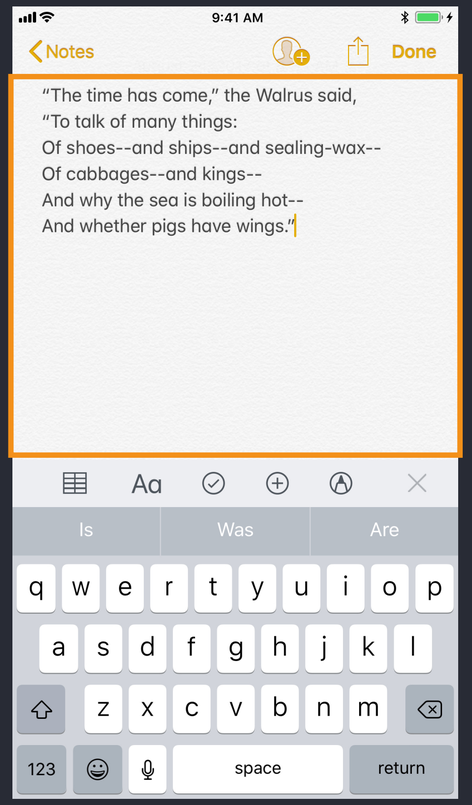


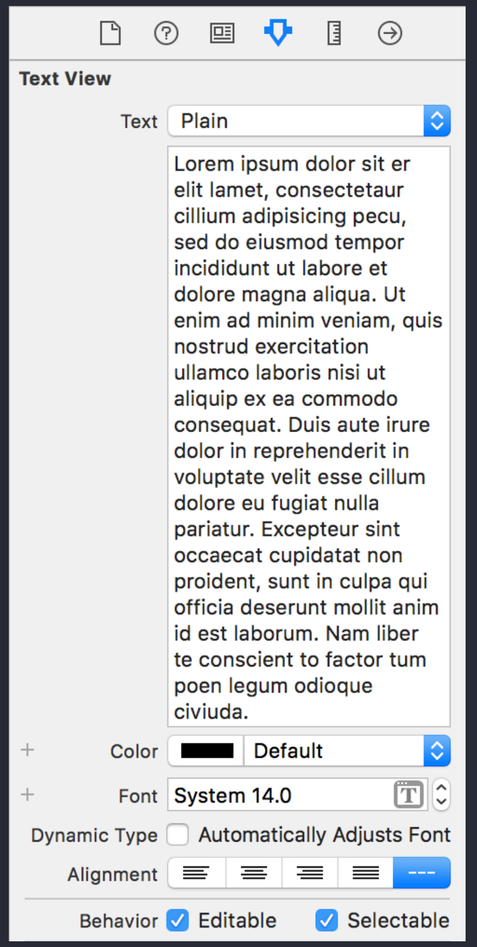
Aspect Fill

Image – UIImageView

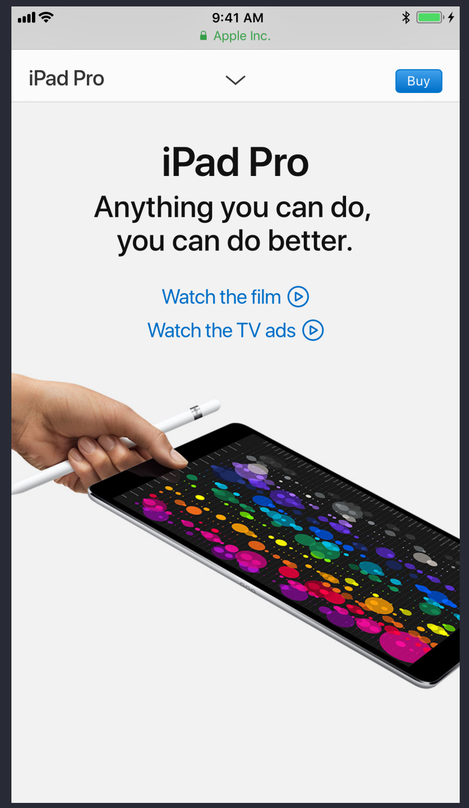


Text – UITextView





Scroll view – UIScrollView



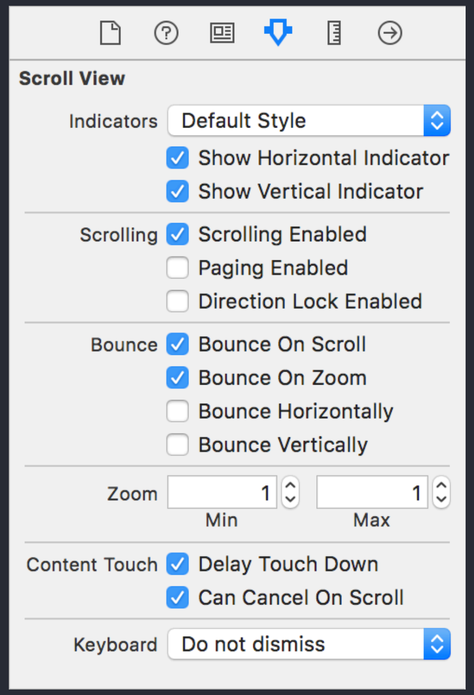
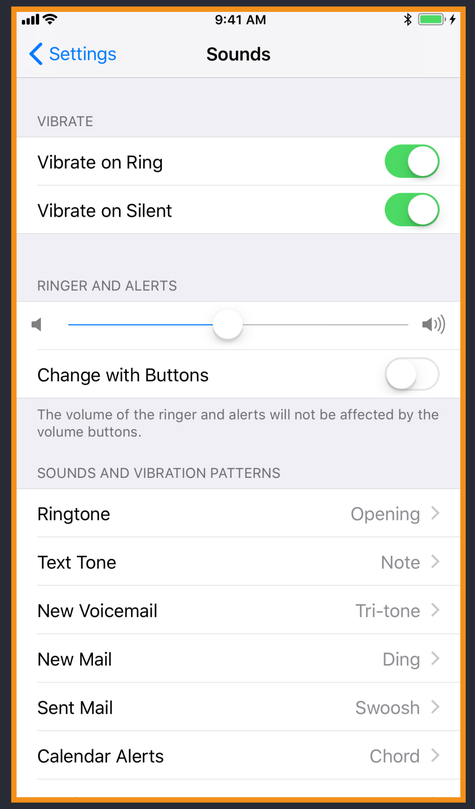
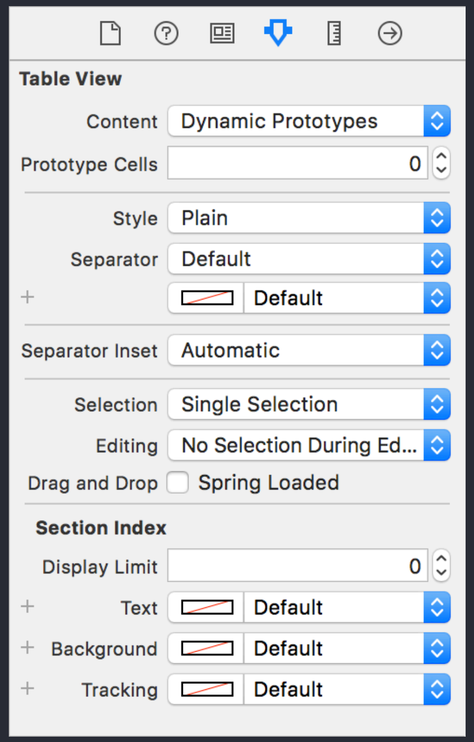
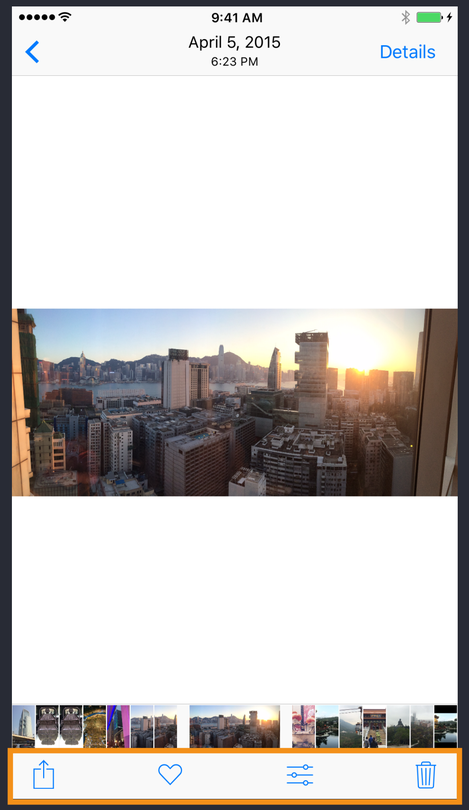


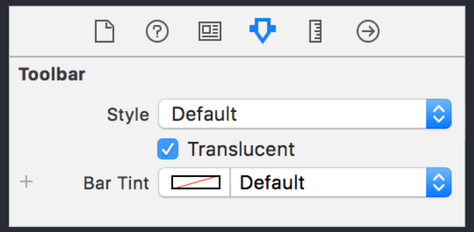
Table – UITableView



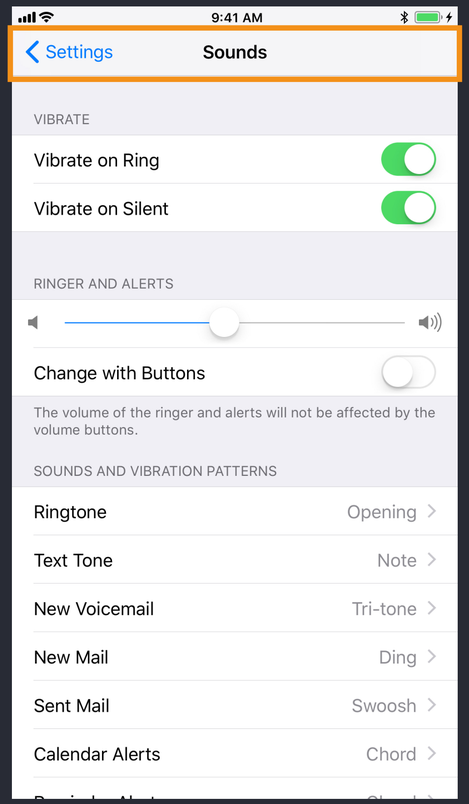


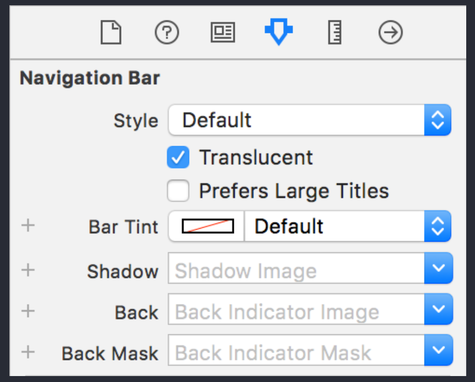
Toolbars – UIToolbar



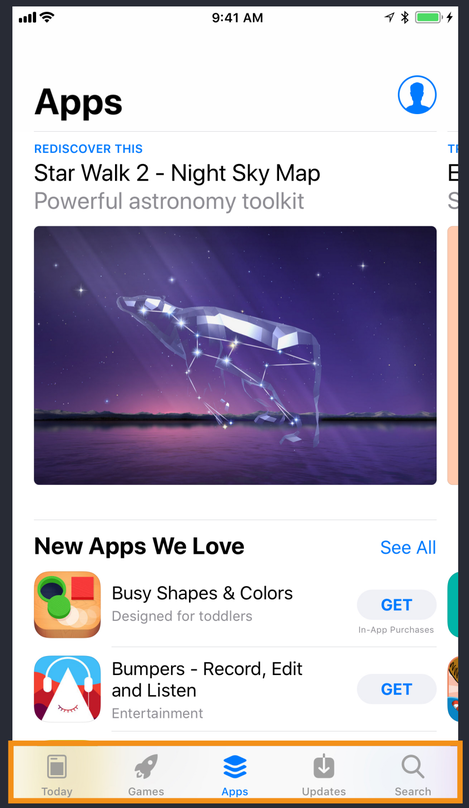


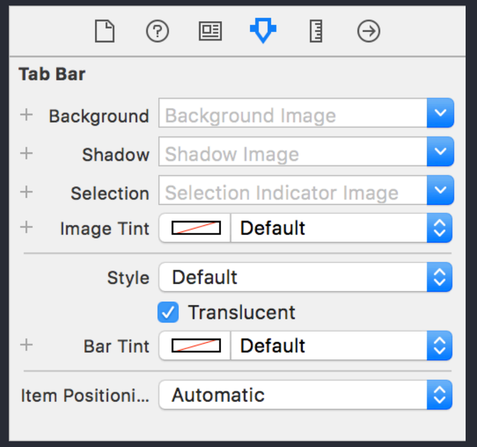
Navigation bar – UINavigationBar



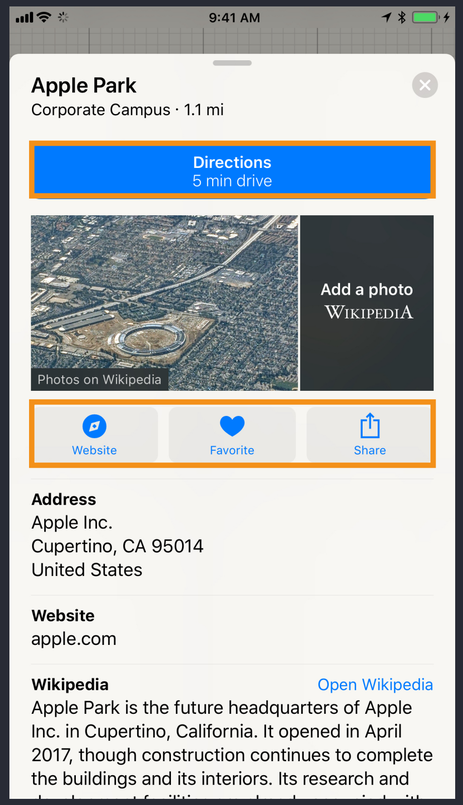


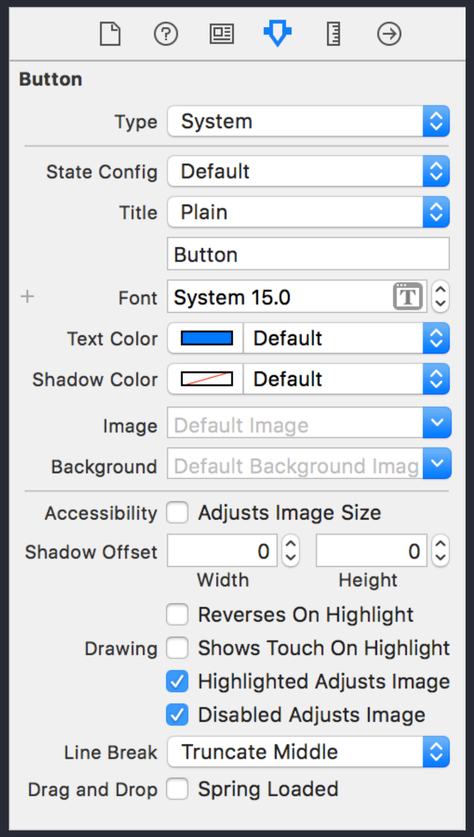
Tabs – UITabBar



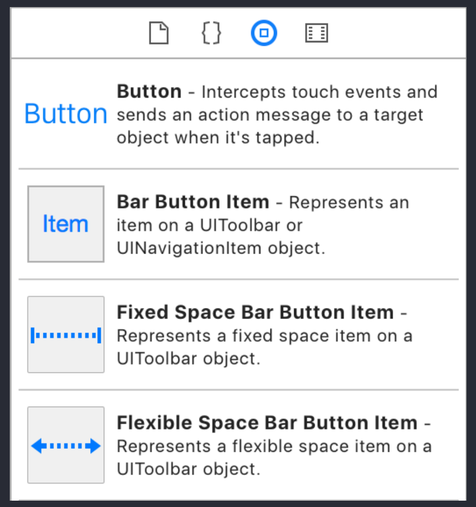


Buttons – UIButton



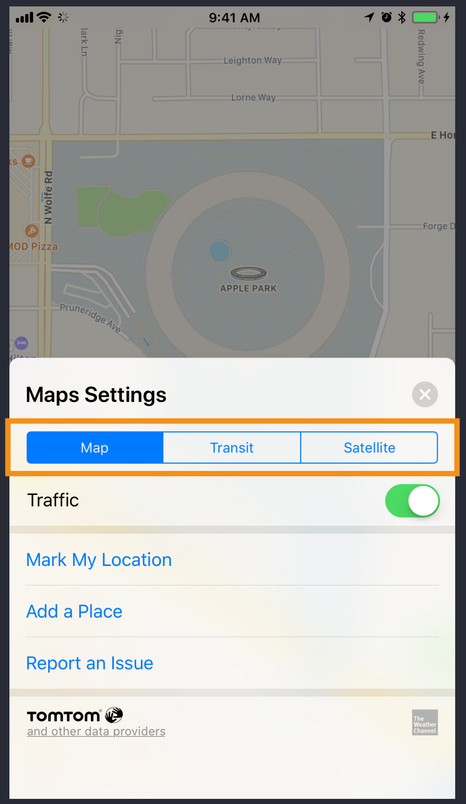


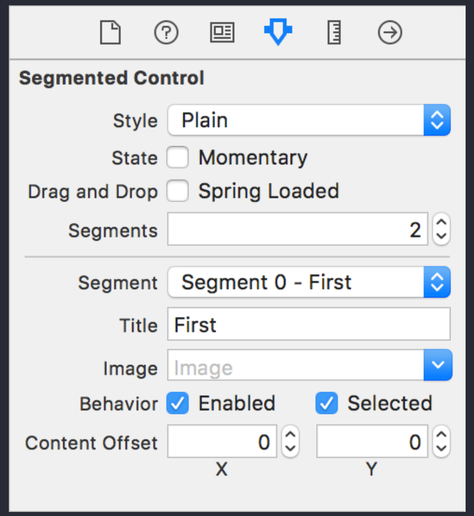
Buttons – UIButton



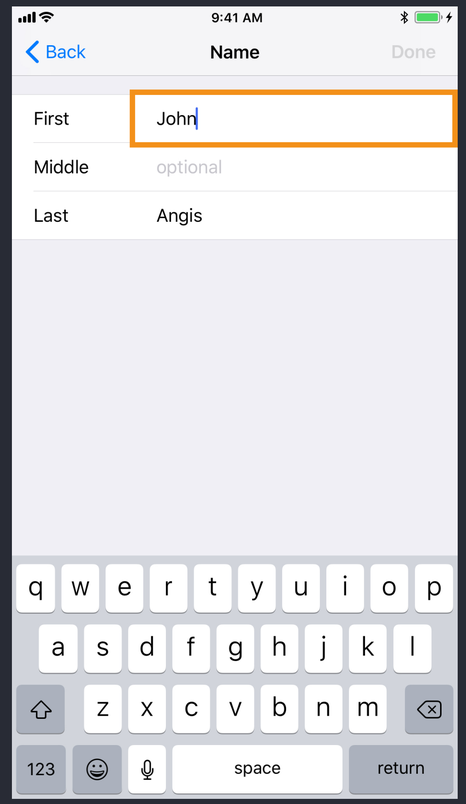

```
@IBAction func buttonTapped(_ sender: Any)
    // Code to respond to button
}
```

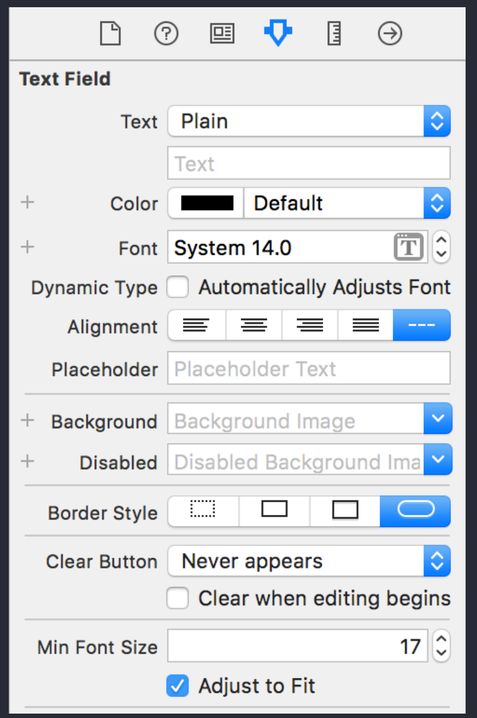
Segmented control – UISegmentedControl



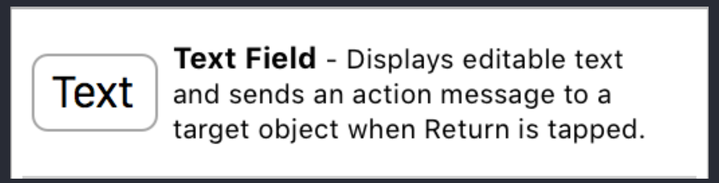


Text fields – UITextField





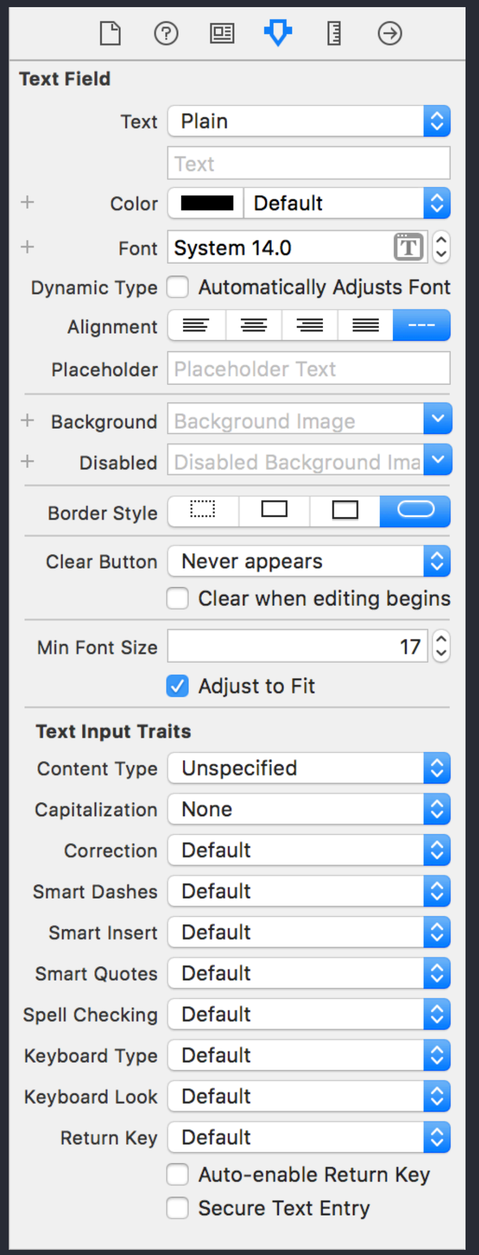
Text fields – UITextField



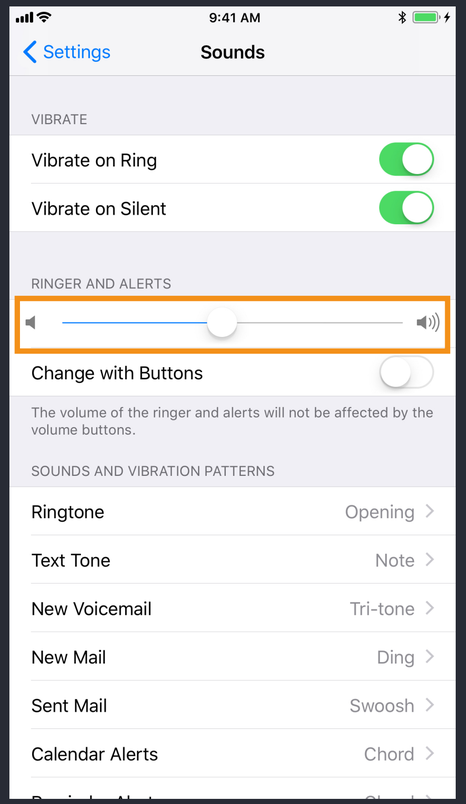

```
@IBAction func keyboardReturnKeyTapped(_
sender: UITextField) {
    if let text = sender.text {
        print(text)
    }
}
```

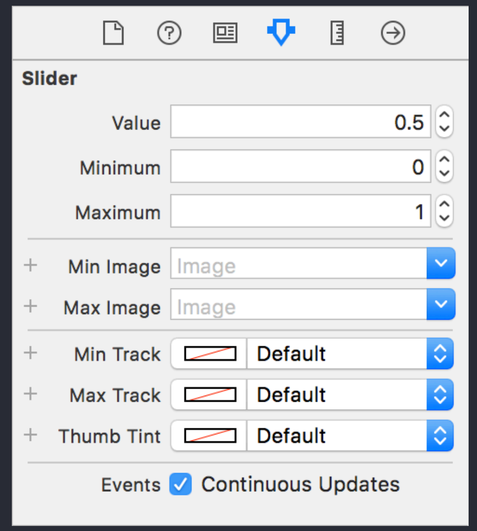
UITextField

Placeholder Text	Text displayed when the text field is empty
Text	Text displayed by the text field
Capitalization	How the keyboard deals with capitalization
Correction	Enables or disables autocorrect
Keyboard	Which keyboard is displayed—for example, email, web, or default
Return Key	Text on the return key
Secure	Specific text fields that don't display their contents, commonly used for passwords

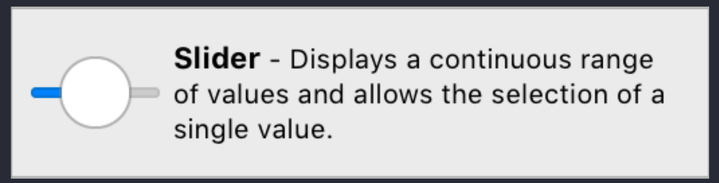


Sliders – UISlider





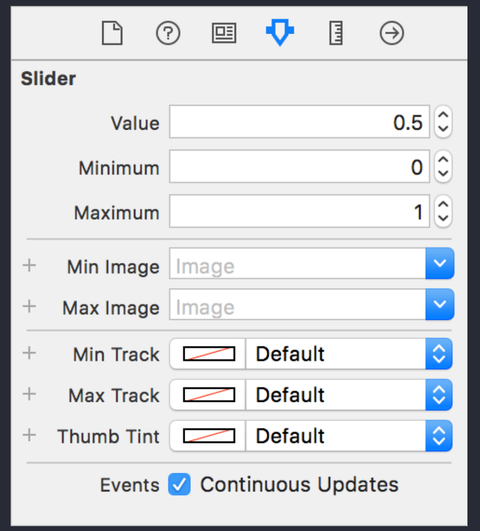
Sliders – UISlider



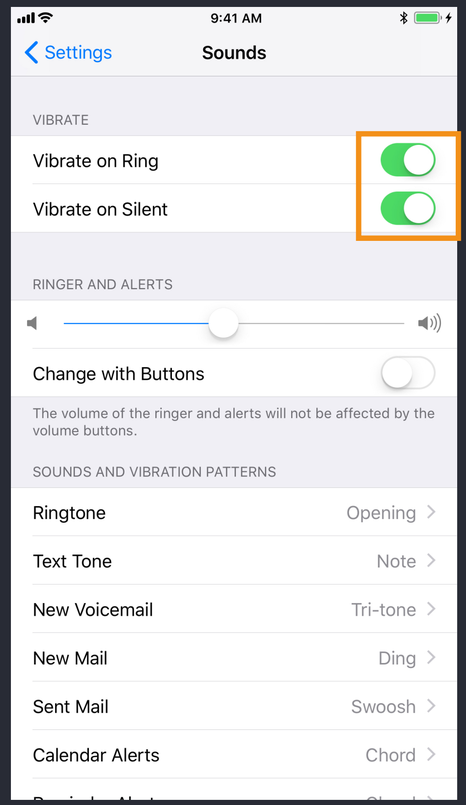

```
@IBAction func sliderValueChanged(_
sender: UISlider) {
    print(sender.value)
}
```

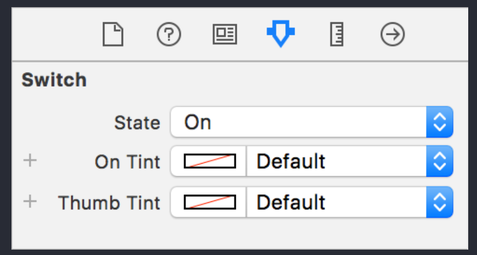
Sliders – UISlider

Minimum Value	Lowest number value the slider may represent
Maximum Value	Highest number value the slider may represent
Current	Starting number value the slider will represent
Min Image	Optional image on the minimum end of the slider
Max Image	Optional image on the maximum end of the slider

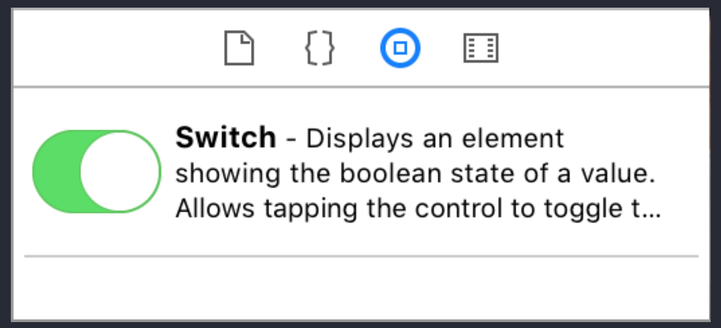


Switches – UISwitch



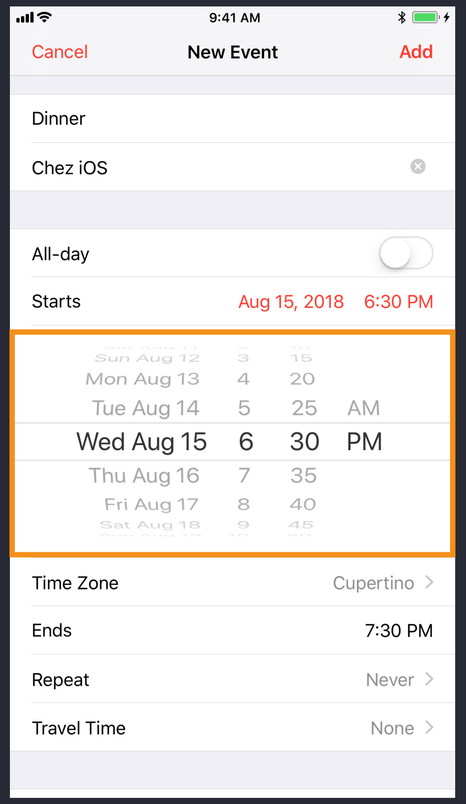


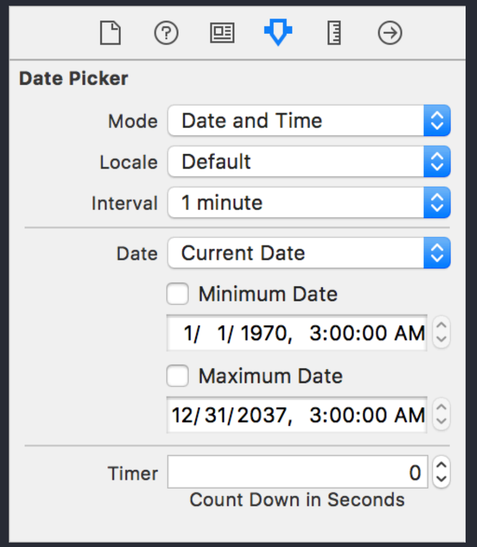
Switches – UISwitch



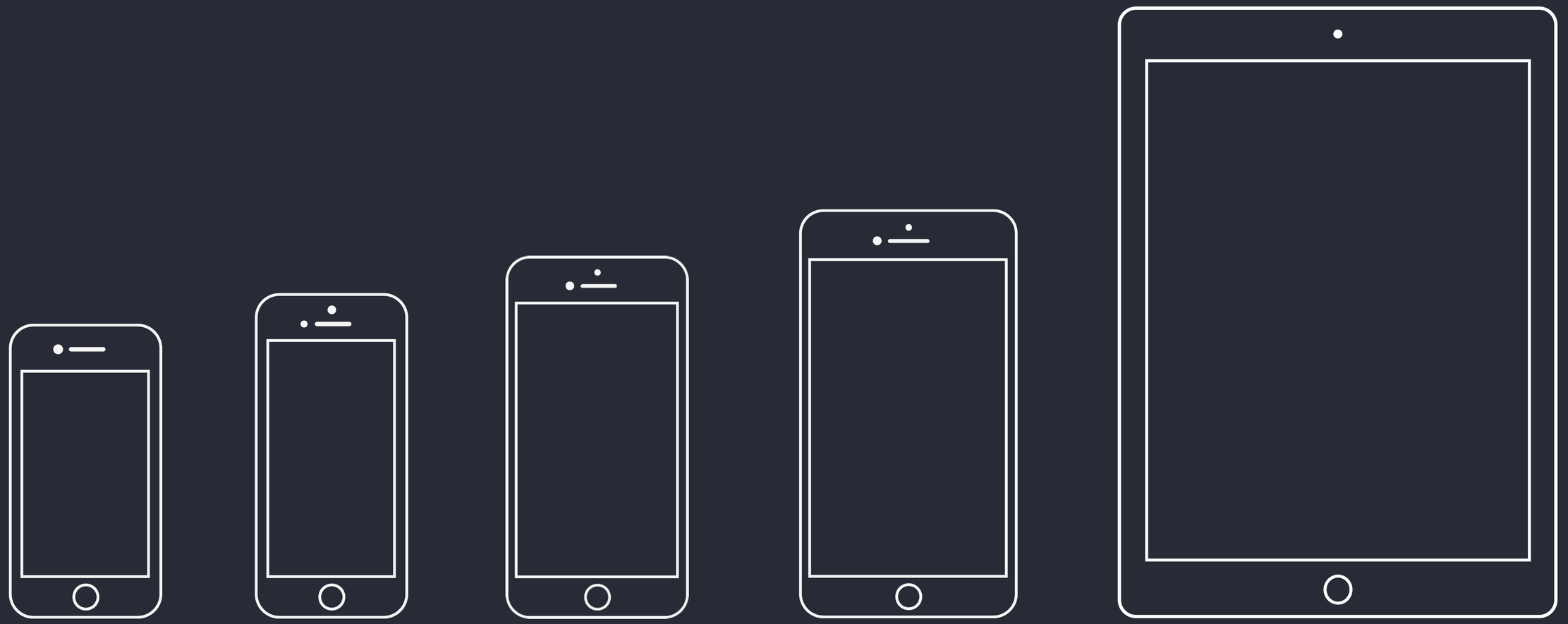

```
@IBAction func switchToggled(_ sender:
UISwitch) {
    if sender.isOn {
        print("The switch is on!")
    } else {
        print("The switch is off.")
    }
}
```

Date pickers – UIDatePicker

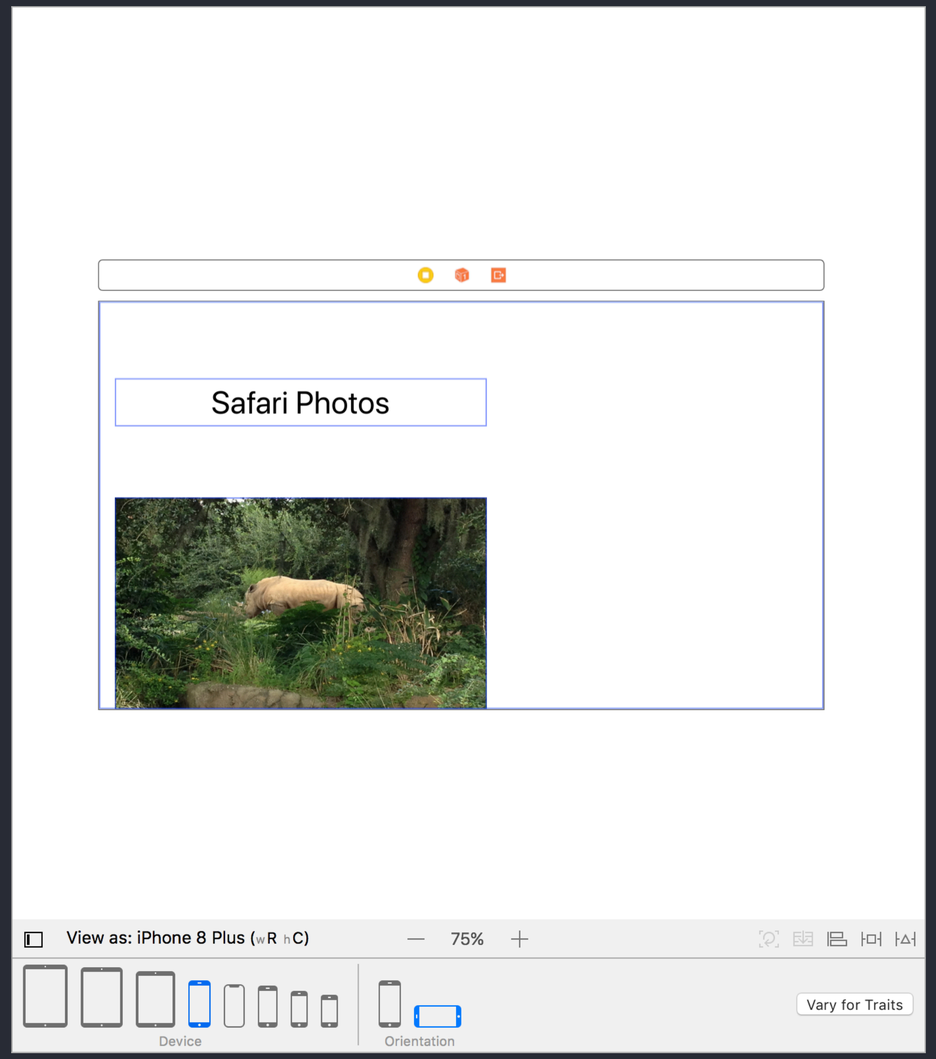




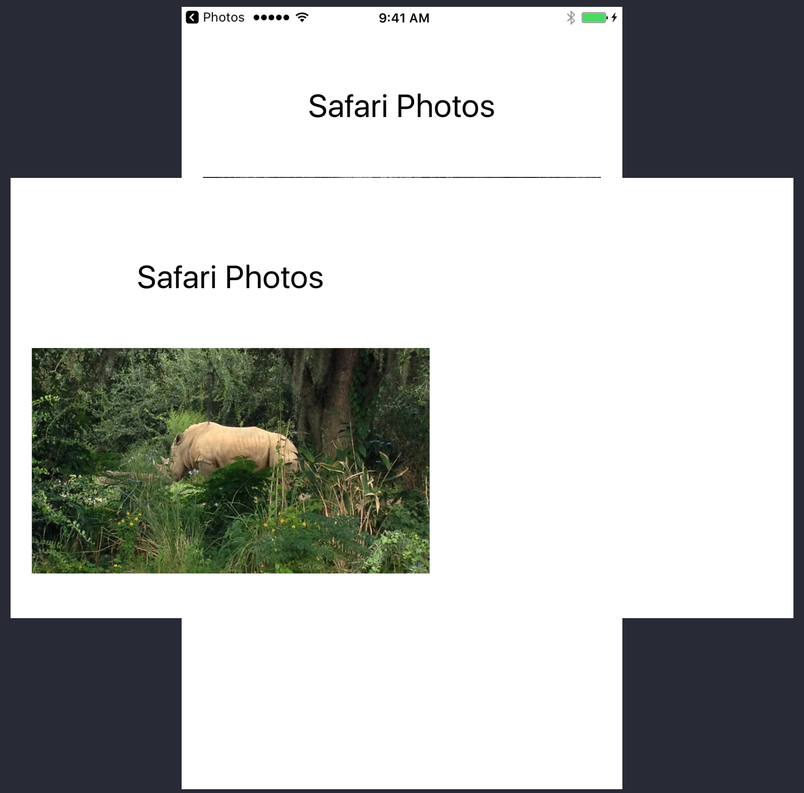
Auto Layout & Stack Views



Interface Builder



Why Auto Layout?

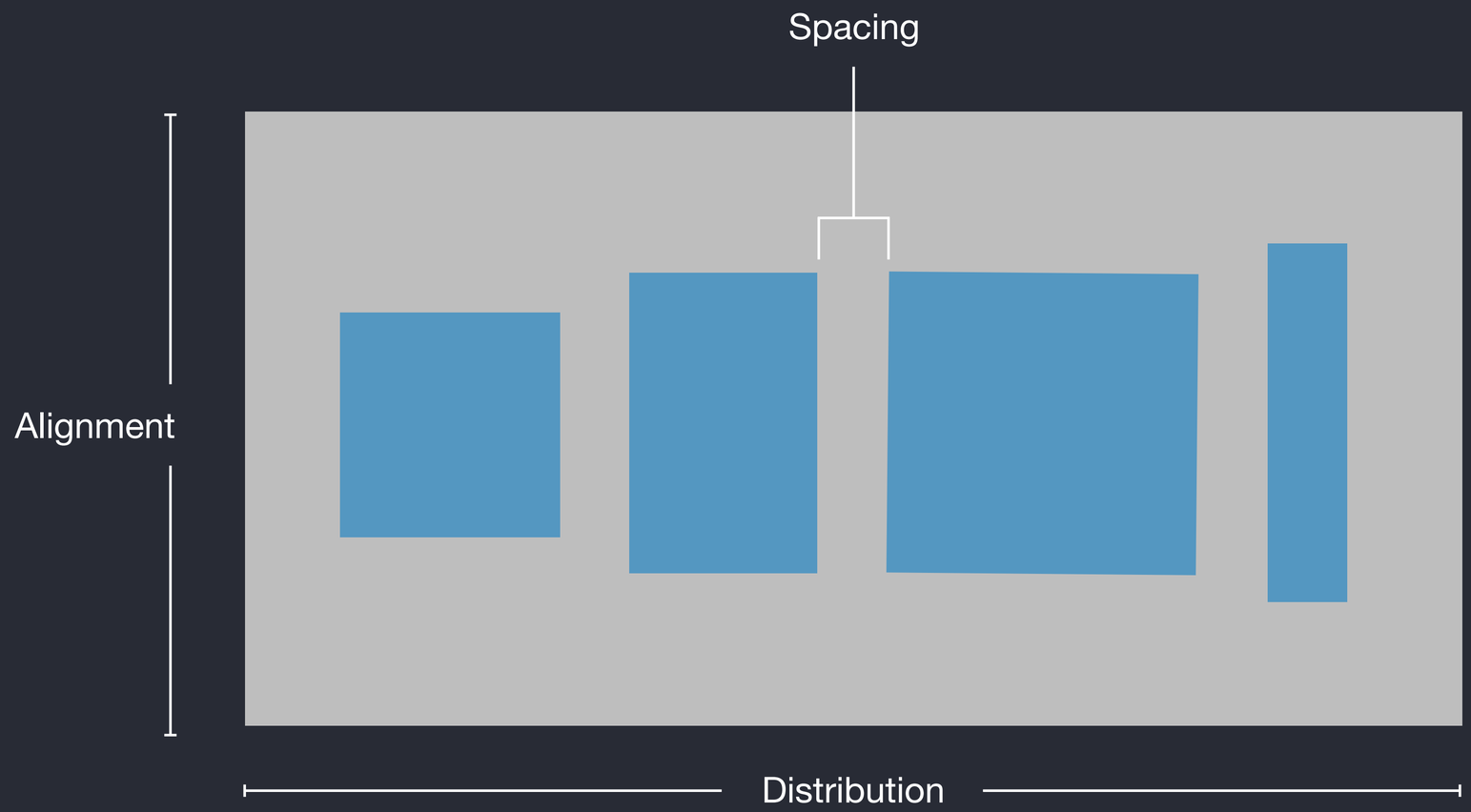


Demo

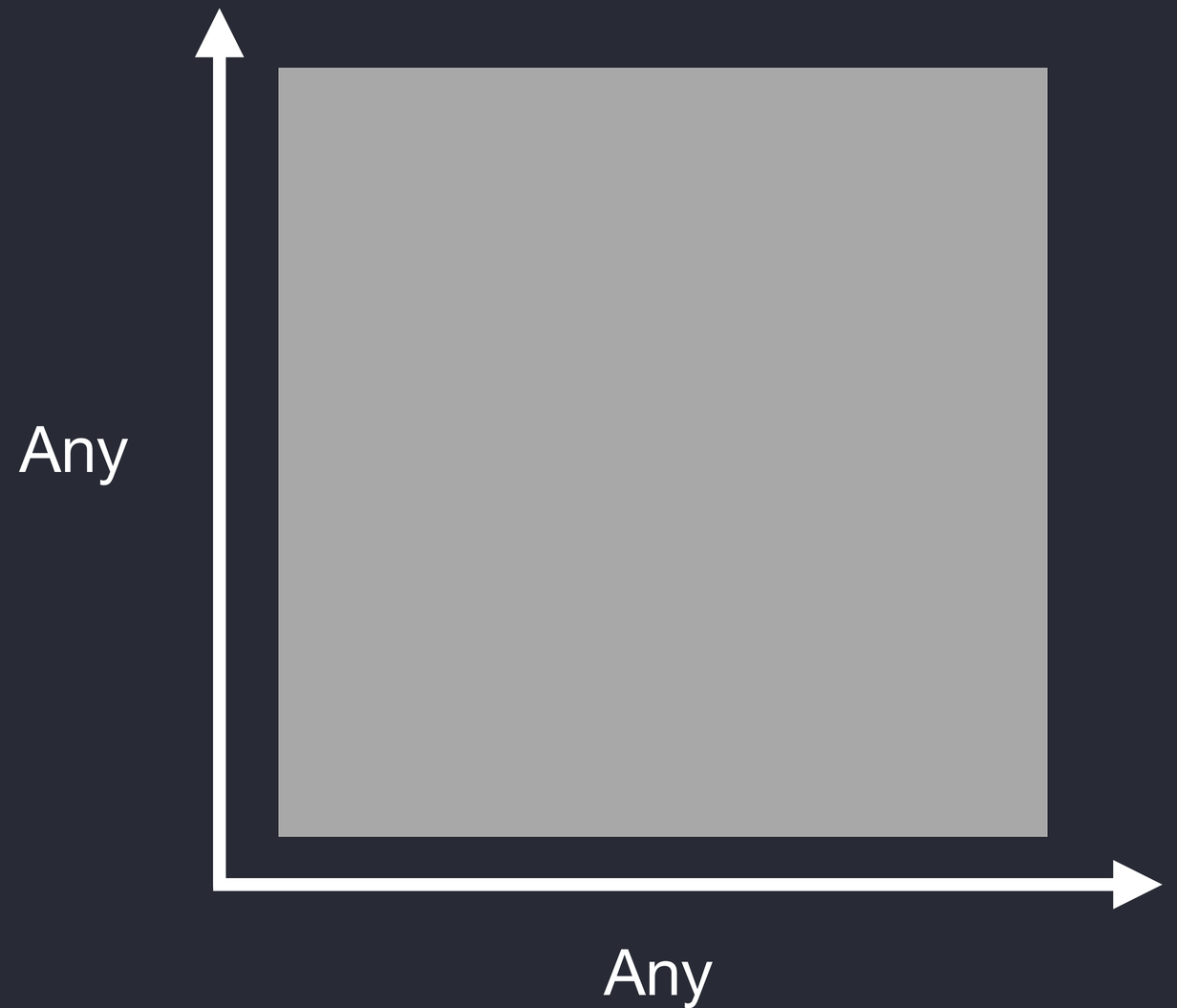
Stack views


Johnny Appleseed
Enjoys Snorkeling
Loves to Play Board Games
Learning to Build iOS Apps
Just Started Coin Collecting

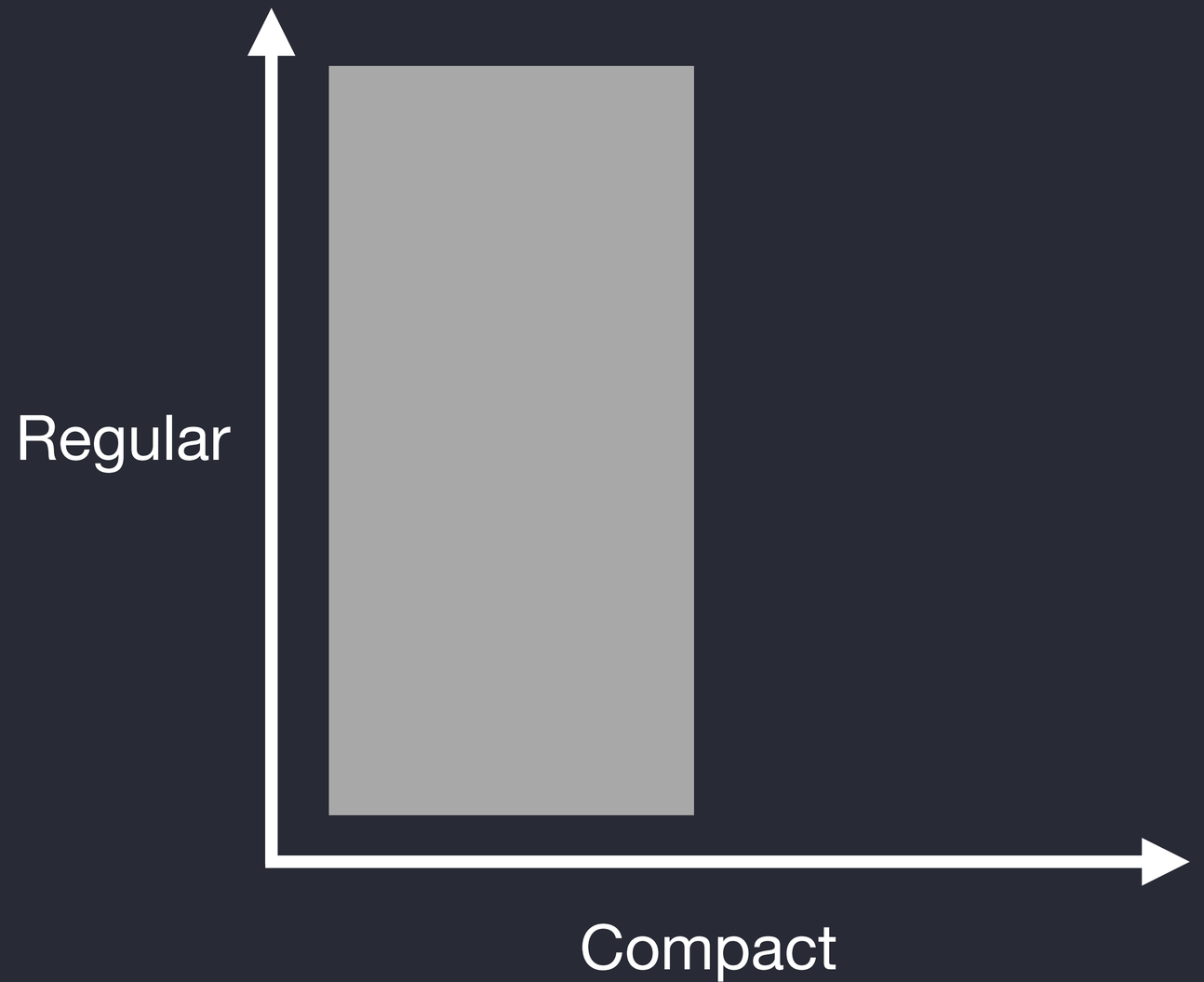
Stack views



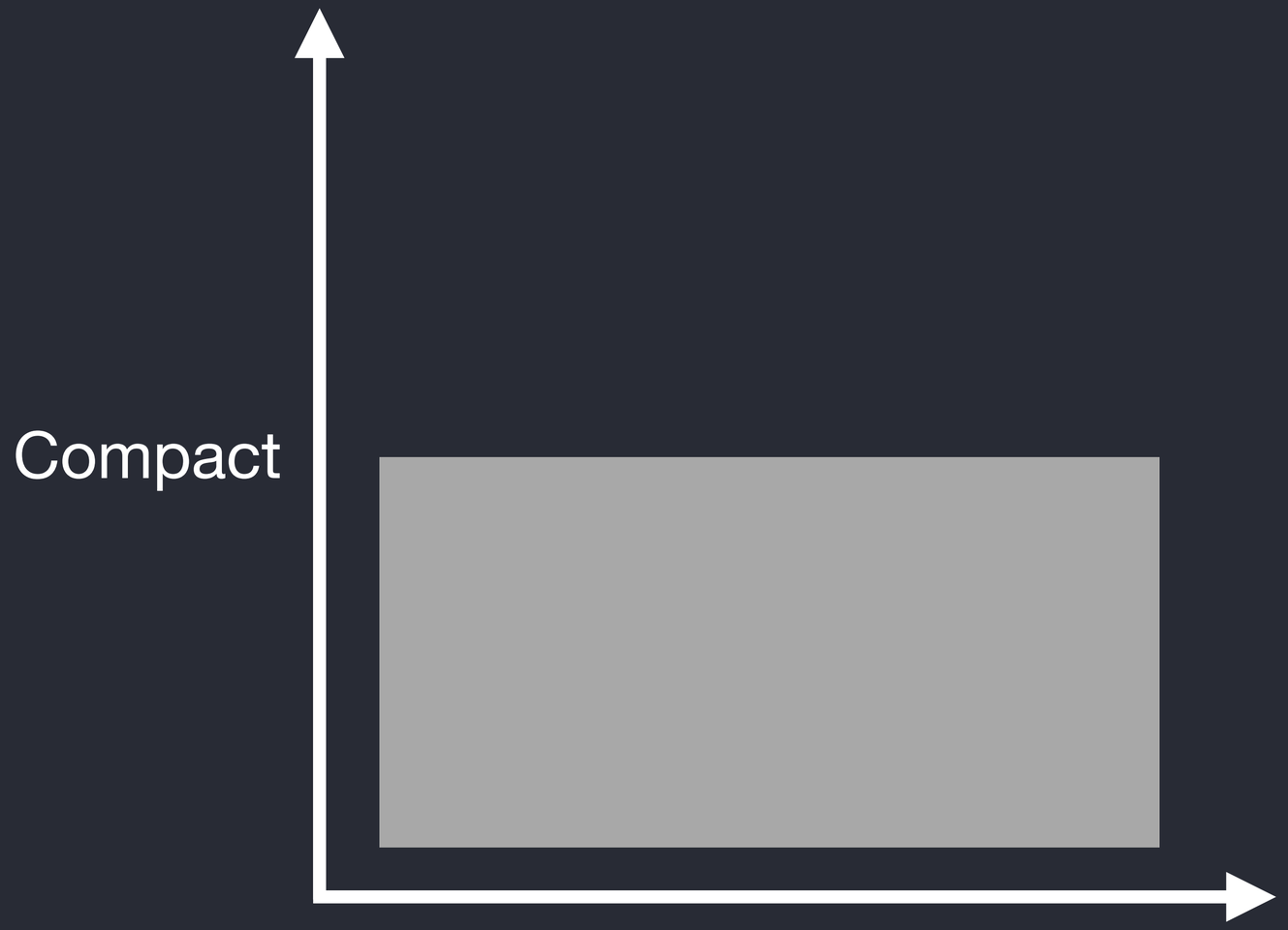
Size classes



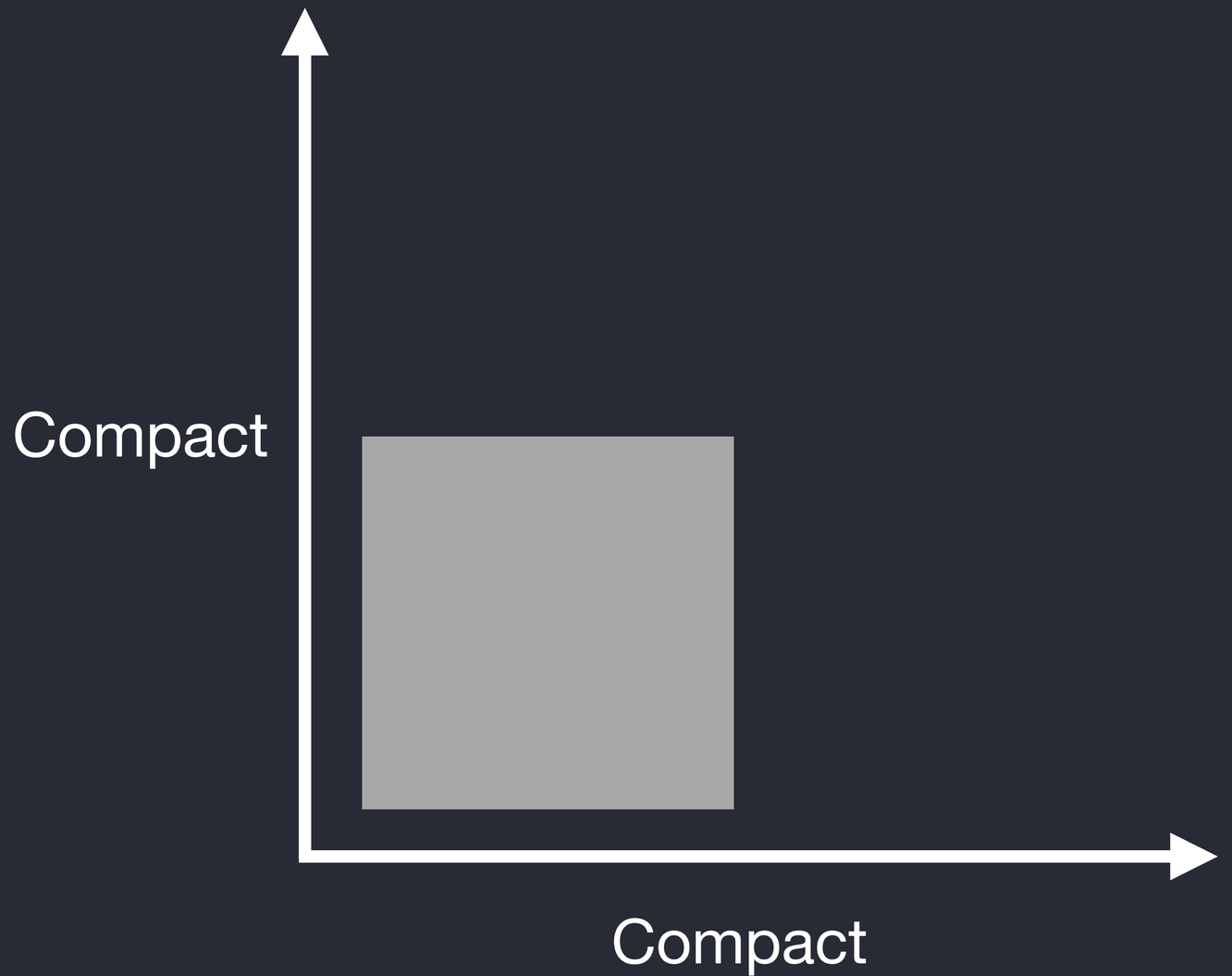
Size classes



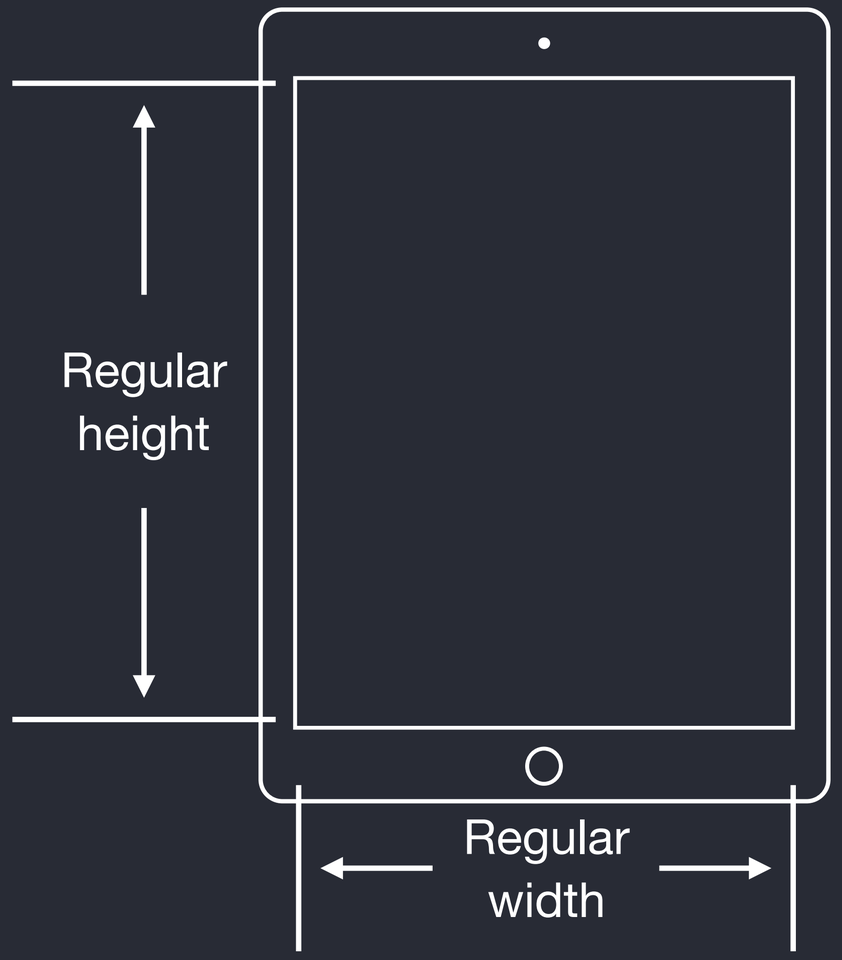
Size classes

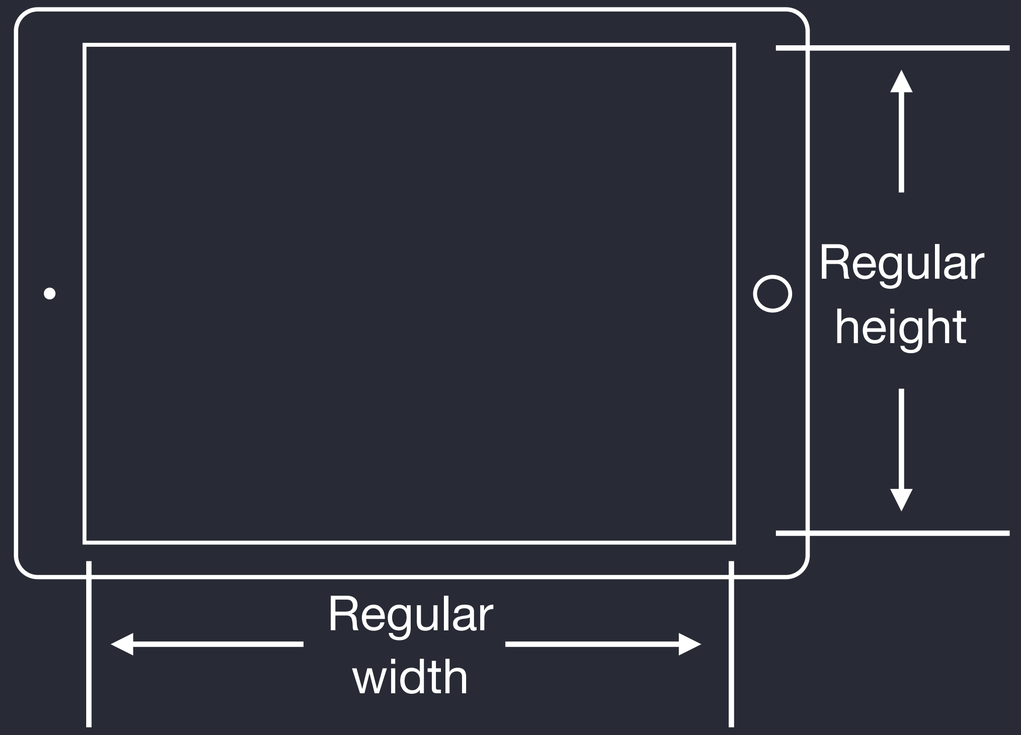


Size classes

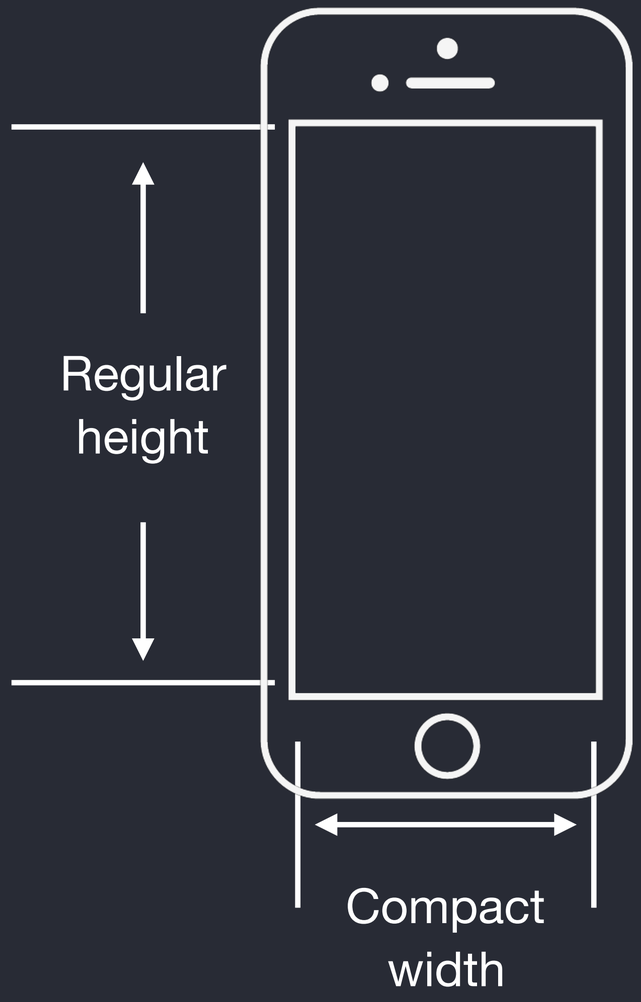


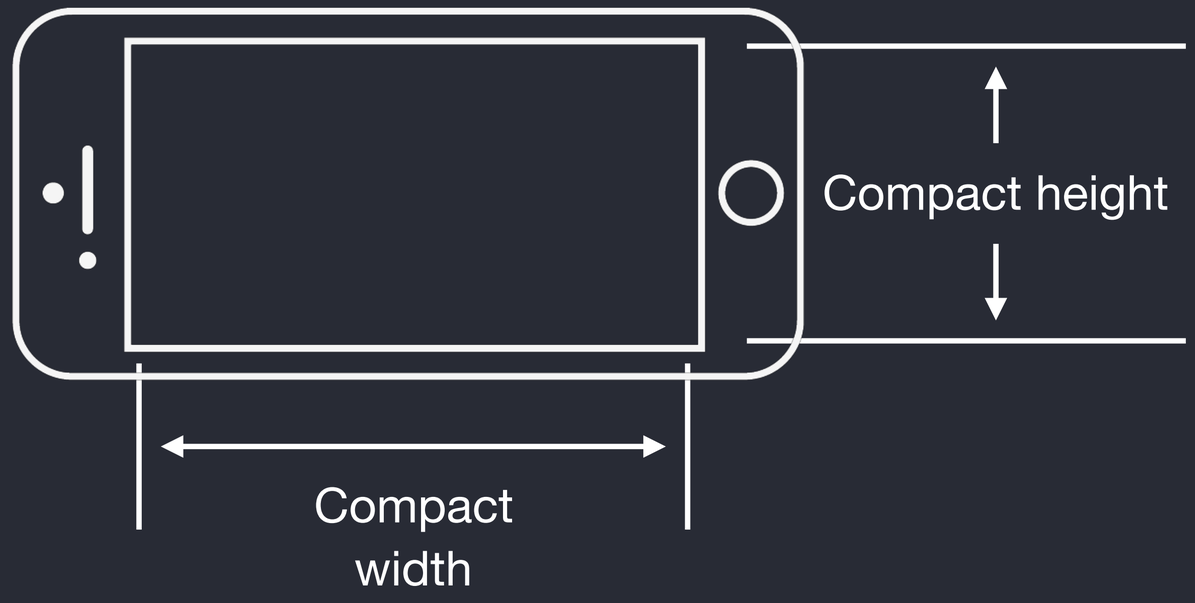
Size classes



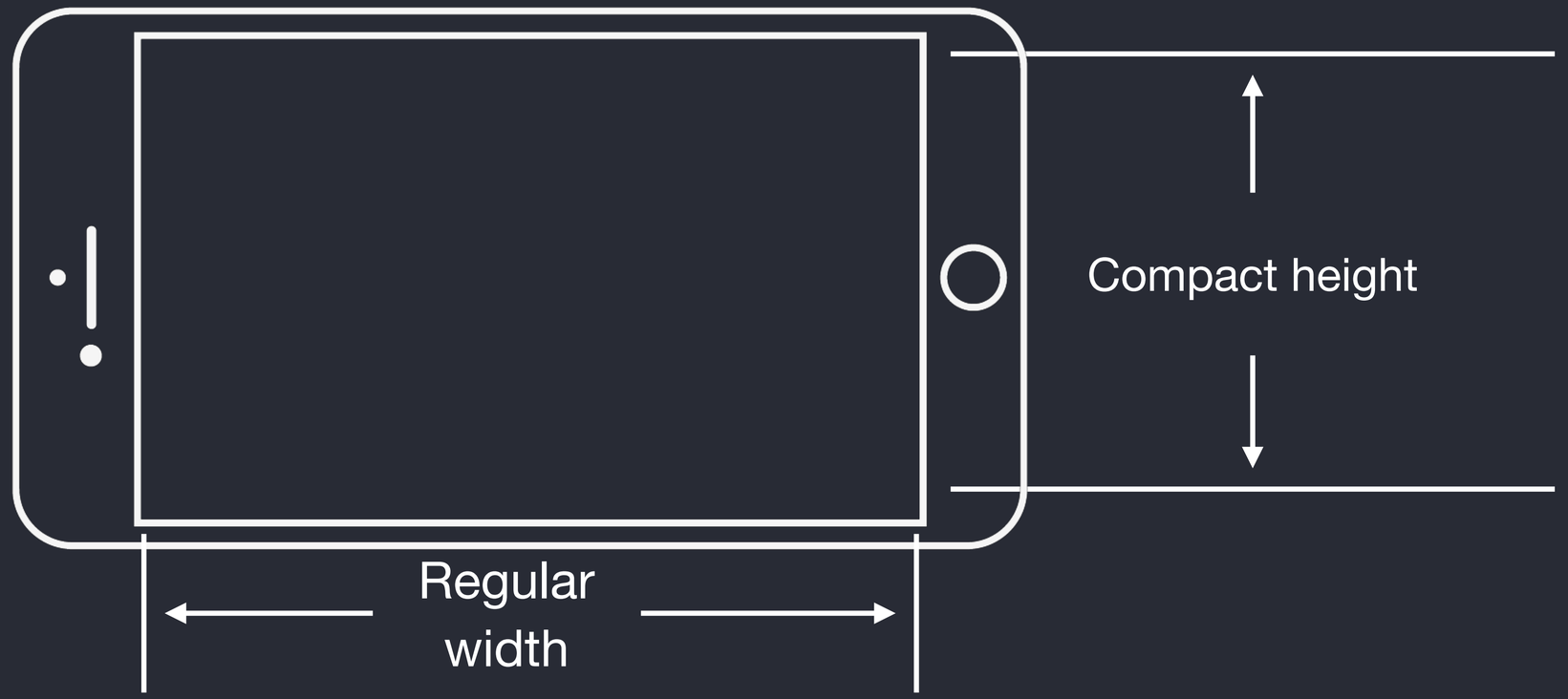


Size classes



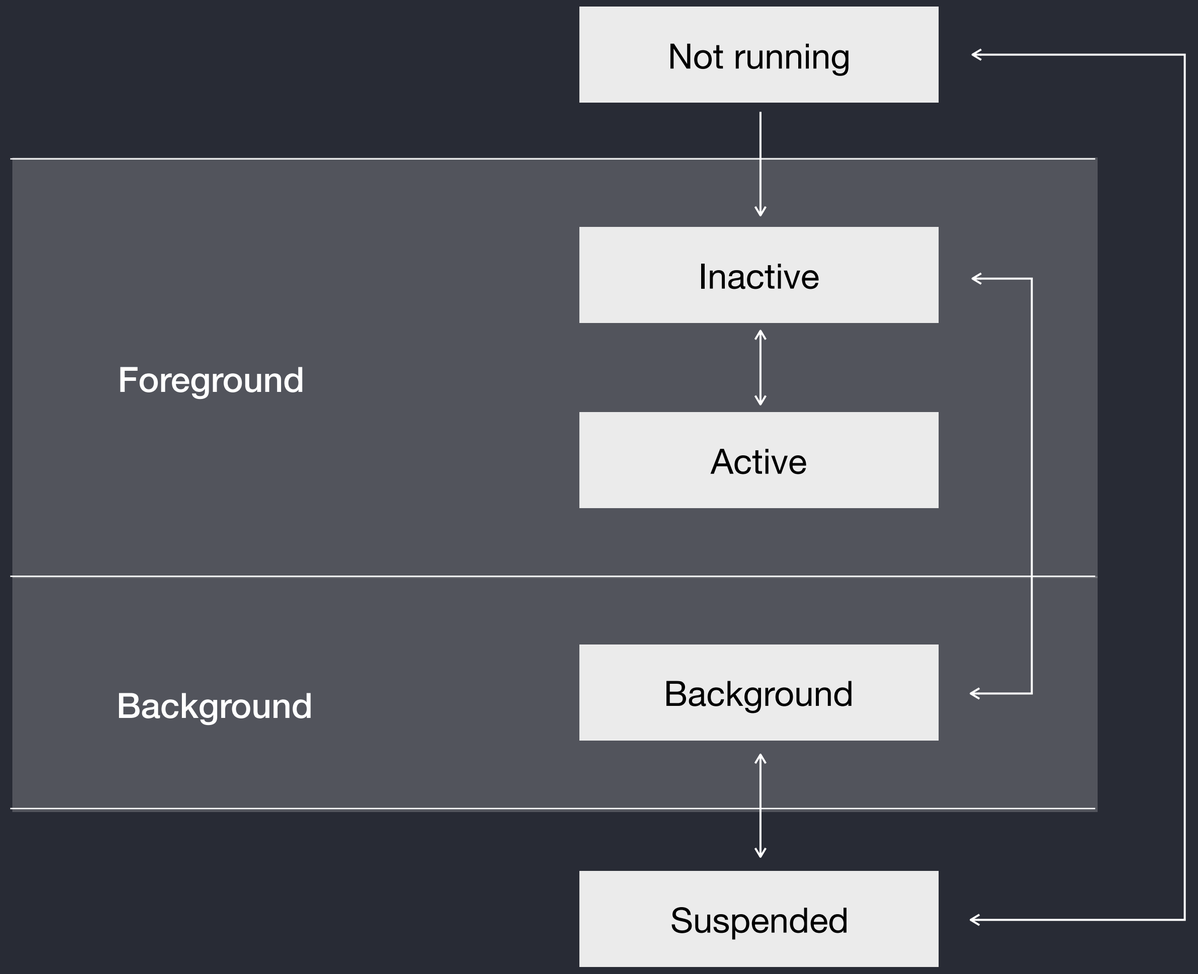


Size classes



Application lifecycle

App life cycle



UIAppDelegate

- Did Finish Launching
- Will Resign Active
- Did Enter Background
- Will Enter Foreground
- Did Become Active
- Will Terminate

UIAppDelegate

Did Finish Lauching

→ App has finished launching

```
func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
    [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
   return true
}
```

→ Override point for customization after app launch

UIAppDelegate

Will Resign Active

- App is about to move from active to inactive state

```
func applicationWillResignActive(_ application: UIApplication) {}
```

- Can occur for certain types of temporary interruptions (such as an incoming phone call or SMS message)
- Can occur when the user quits the app and it begins the transition to the background state
- Use to pause ongoing tasks, disable timers, and invalidate graphics rendering callbacks

UIAppDelegate

Did Enter Background

- App is about to move from active to inactive state

```
func applicationDidEnterBackground(_ application: UIApplication) {}
```

- Use to release shared resources, save user data, invalidate timers, and store enough application state information to restore your application to its current state in case it’s terminated later
- If your application supports background execution, this method is called instead of applicationWillTerminate: when the user quits

UIAppDelegate

Will Enter Foreground

- Called immediately before the applicationDidBecomeActive function

```
func applicationWillEnterForeground(_ application: UIApplication) {}
```

- Called as part of transition from the background to the active state
- Can be used to undo many of the changes made on entering the background

UIAppDelegate

Did Become Active

- App was launched by the user or system

```
func applicationDidBecomeActive(_ application: UIApplication) {}
```

- Restart any tasks that were paused (or not yet started) while the app was inactive
- If the app was previously in the background, optionally refresh the user interface

UIAppDelegate

Will Terminate

- App is about to be terminated

```
func applicationWillTerminate(_ application: UIApplication) {}
```

- Save data if appropriate
- See also applicationDidEnterBackground:

UIAppDelegate

Which methods should I use?

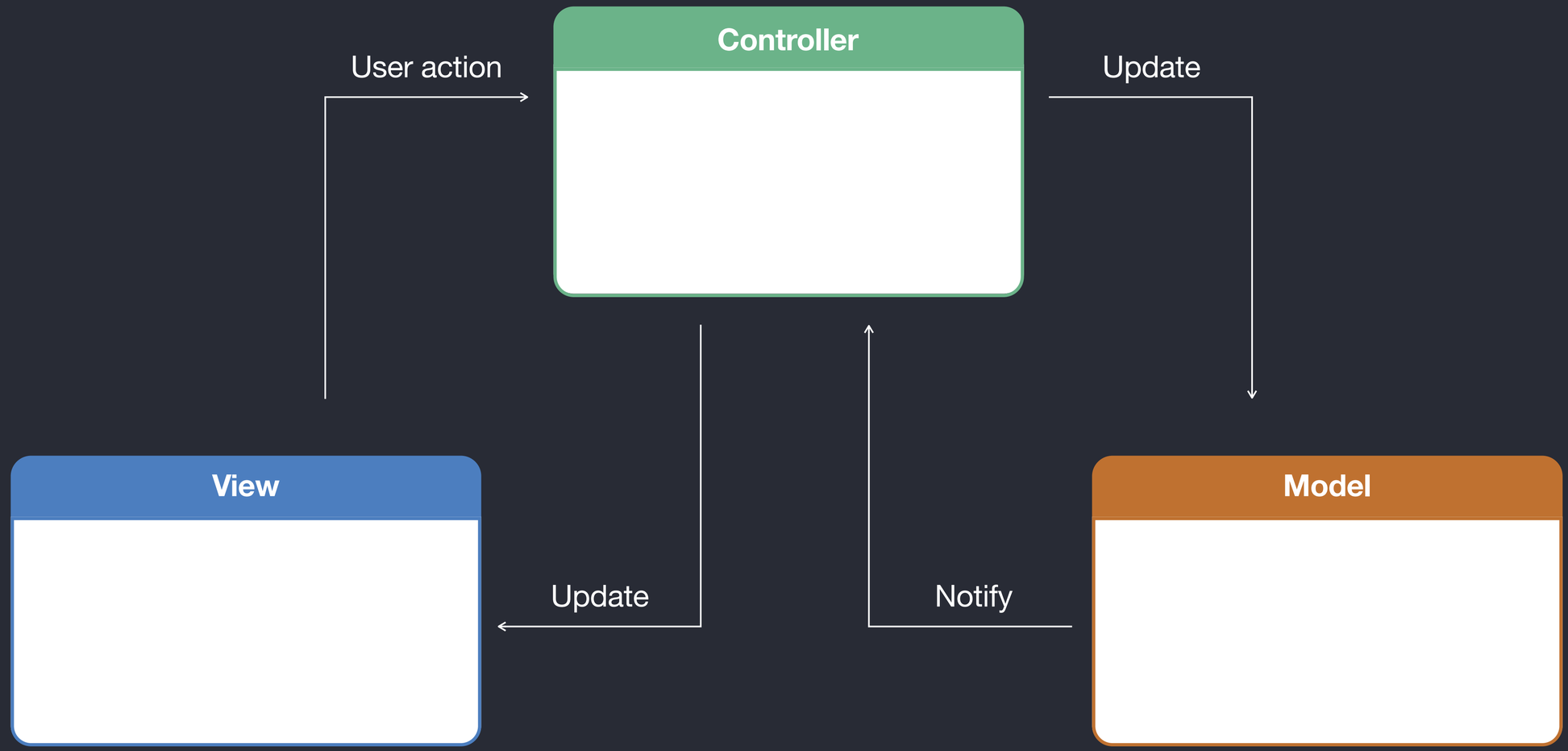
- Start with the methods that will run when launching, reopening, or closing your app

```
applicationDidFinishLaunchingWithOptions
applicationWillResignActive
applicationDidBecomeActive
```

- Take advantage of the other three delegate methods as you become more experienced

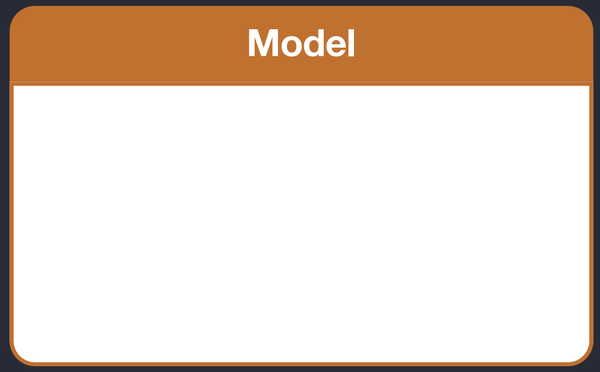
Model View Controller

Model View Controller

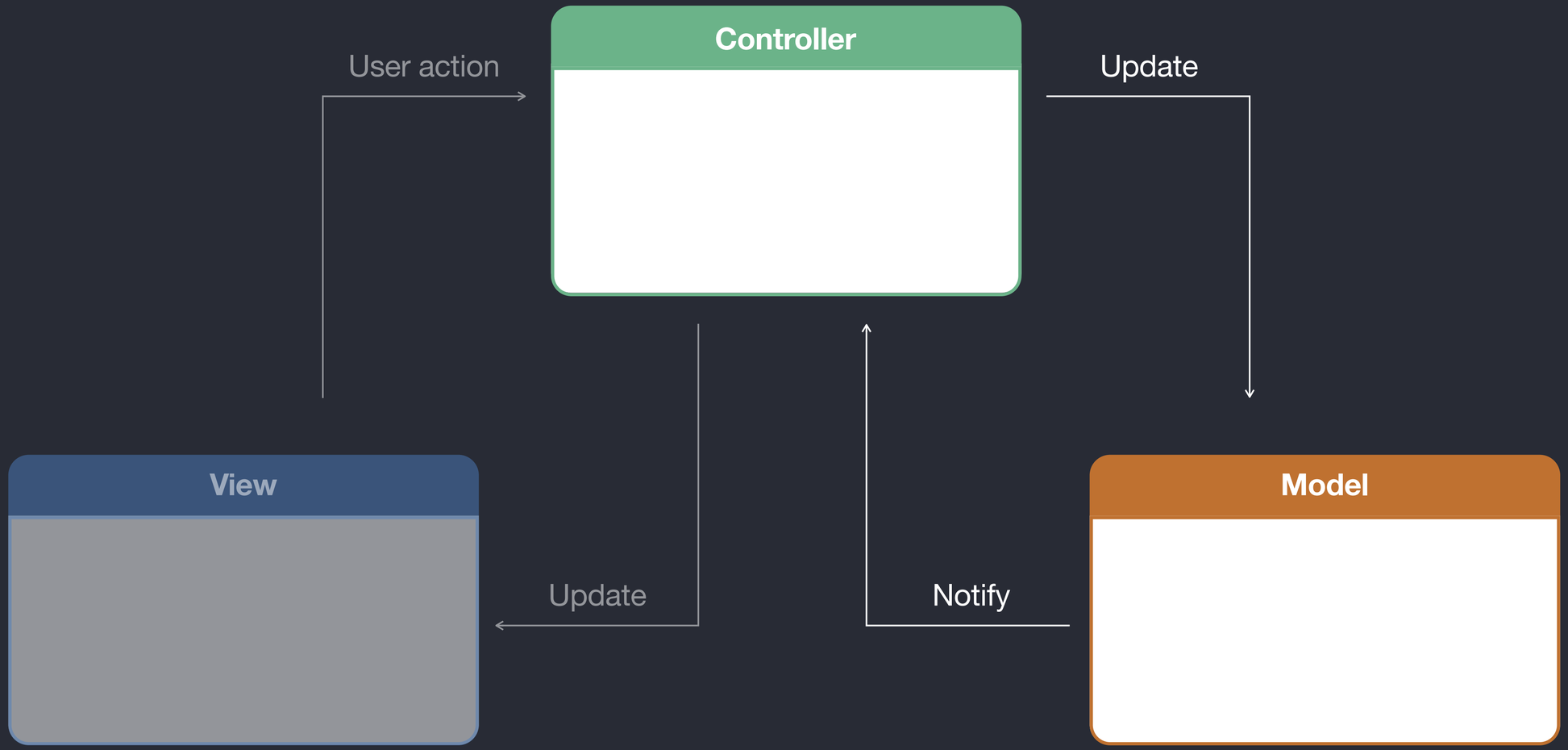


Model objects

- Groups the data needed for a specific problem domain or a type of solution to be built
- Can be related to other model objects

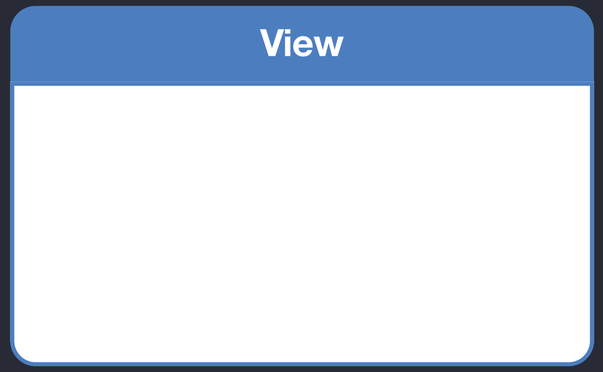


Model objects

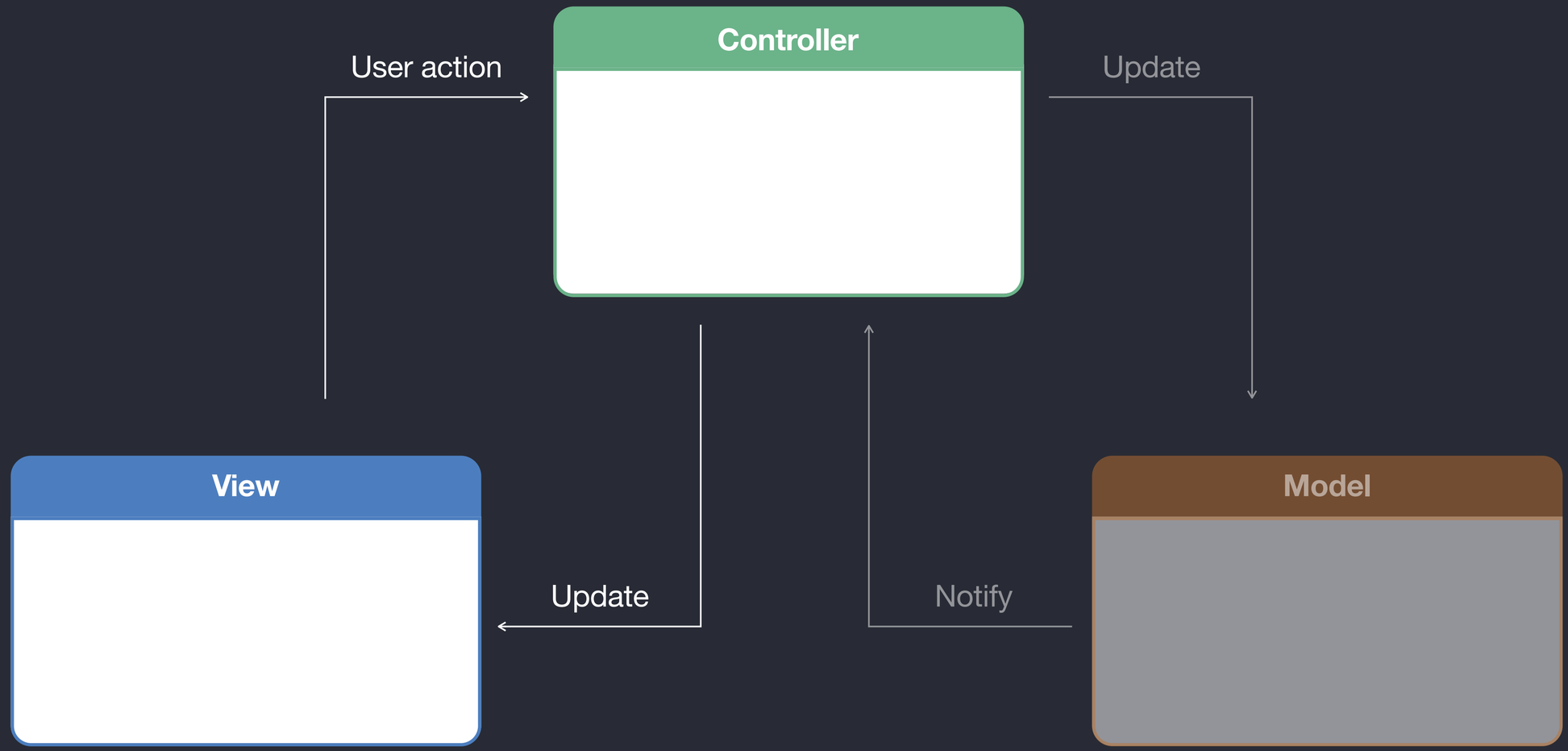


Views

- Displays data about the app’s model objects and allows user to edit the data
- Can be reused to show different instances of the model data



Views



Controllers

- Acts as the messenger between views and model objects
- Types:
  - View controllers
  - Model controllers
  - Helper controllers

Model Controllers

Helps control a model object or collection of model objects

Three common reasons to create a model controller:

→ Multiple objects or scenes need access to the model data

→ Logic for adding, modifying, or deleting model data is complex

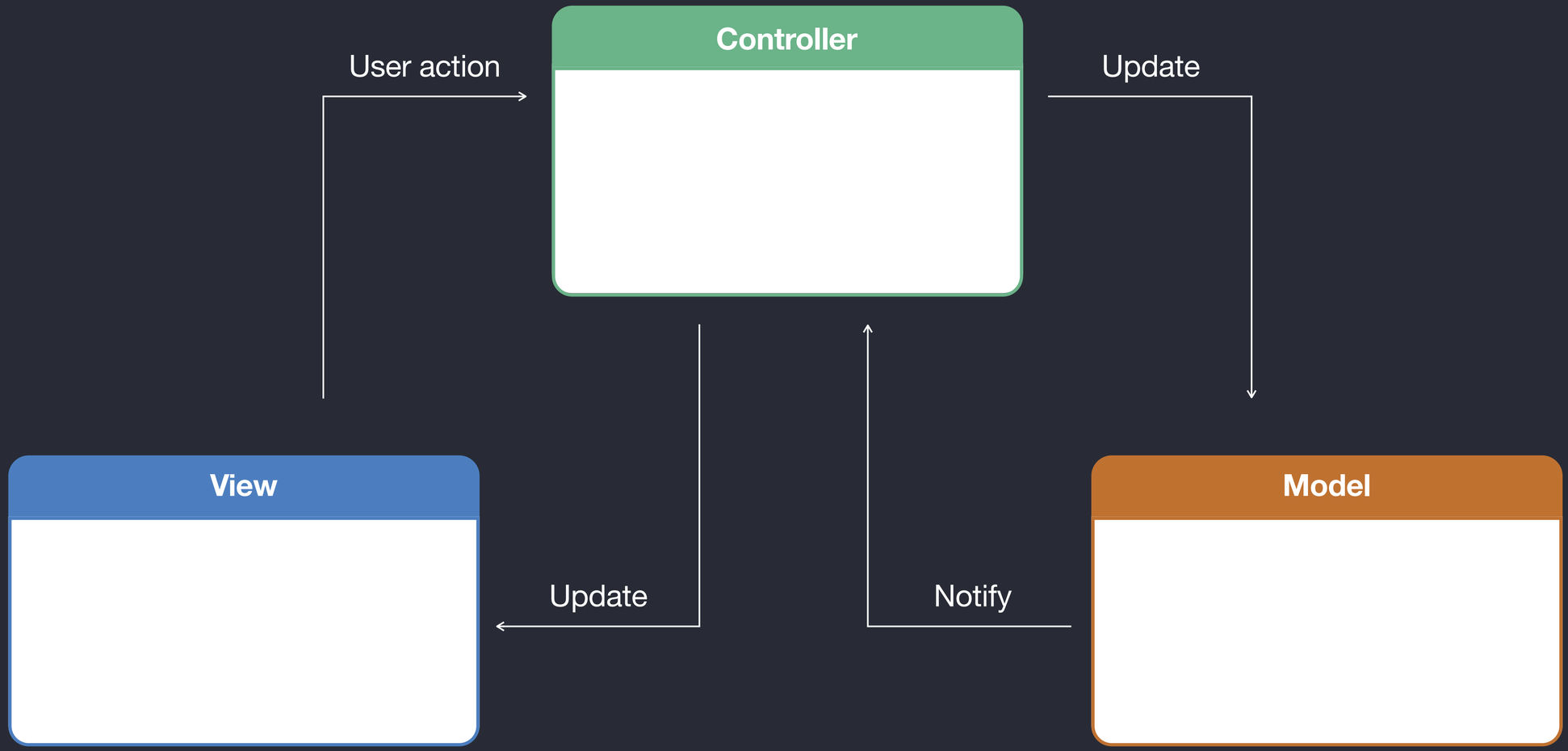
→ Keep the code in view controllers focused on managing the views

Crucial in larger projects for readability and maintainability

Helper Controllers

- Useful to consolidate related data or functionality so that it can be accessed by other objects in your app

Controllers



Example

Meal tracker example

Creating an app to track eaten meals

→ What should be in a "Meal" model object?

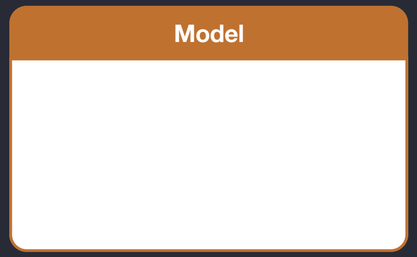
→ What views are needed to display meals?

→ How many controllers makes sense?

Meal tracker example

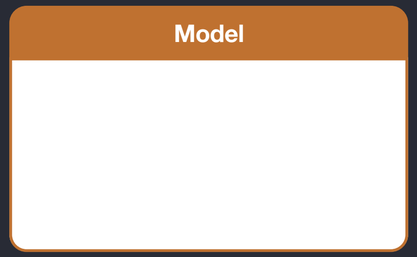
Meal:

- → Name
- → Photo
- → Notes
- → Rating
- → Timestamp



Meal tracker example

```
struct Meal {
    var name: String
    var photo: UIImage
    var notes: String
    var rating: Int
    var timestamp: Date
}
```



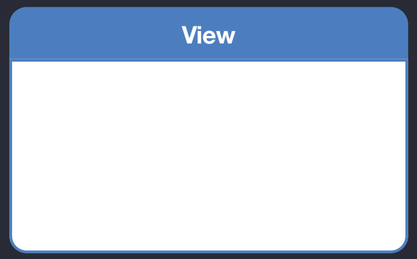
Meal tracker example

Two possible views:

→ List of all tracked meals

→ Details of each meal

Each needs a view controller class

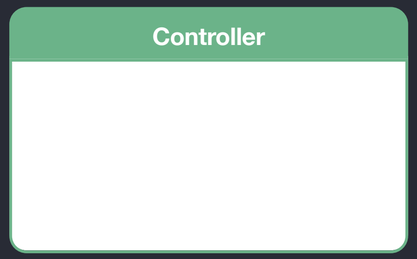


Meal tracker example

Minimum of two controllers:

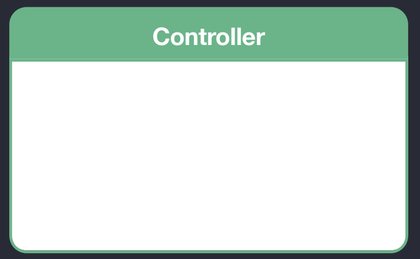
→ List view

→ Detail view



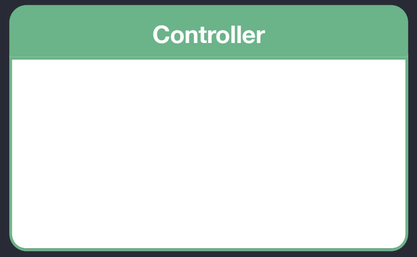
Meal tracker example

```
class MealListTableViewController: UITableViewController {

    var meals: [Meal] = []
    @IBOutlet weak var tableView: UITableView!
}
```



Meal tracker example

```
class MealListTableViewController: UITableViewController {

    var meals: [Meal] = []

    func saveMeals() {...}

    func loadMeals() {...}
}
```

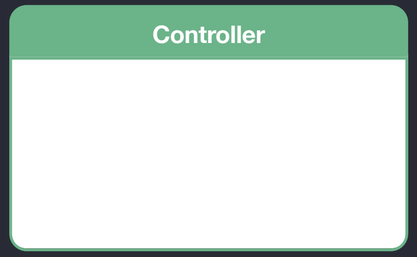


```
class MealListTableViewController: UITableViewController {

    let meals: [Meal] = []

    override func viewDidLoad() {
        // load the meals and set up the table view
    }

    // Required table view methods

    override func tableView(_ tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {...}

    override func tableView(_ tableView: UITableView,
cellForRowAt indexPath: IndexPath) -> UITableViewCell {...}
```

```
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        // Pass the selected meal to the MealDetailViewController
    }

    @IBAction func unwindToMealList(sender: UIStoryboardSegue) {
        // Capture the new or updated meal from the
        // MealDetailVC and save it to the meals property
    }

    // Persistence methods
    func saveMeals() {
        // Save the meals model data to the disk
    }

    func loadMeals() {
        // Load meals data from the disk
        // and assign it to the meals property
    }
}
```

Meal tracker example

```
class MealDetailViewController: UIViewController {...}
```



```
class MealDetailViewController: UIViewController,
UIImagePickerControllerDelegate {

    @IBOutlet weak var nameTextField: UITextField!
    @IBOutlet weak var photoImageView: UIImageView!
    @IBOutlet weak var ratingControl: RatingControl!
    @IBOutlet weak var saveButton: UIBarButtonItem!

    var meal: Meal?

    override func viewDidLoad() {
        if let meal = meal {
            update(meal)
        }
    }

    func update(_ meal: Meal) {
        // Update all outlets to reflect the data about the meal
    }
```

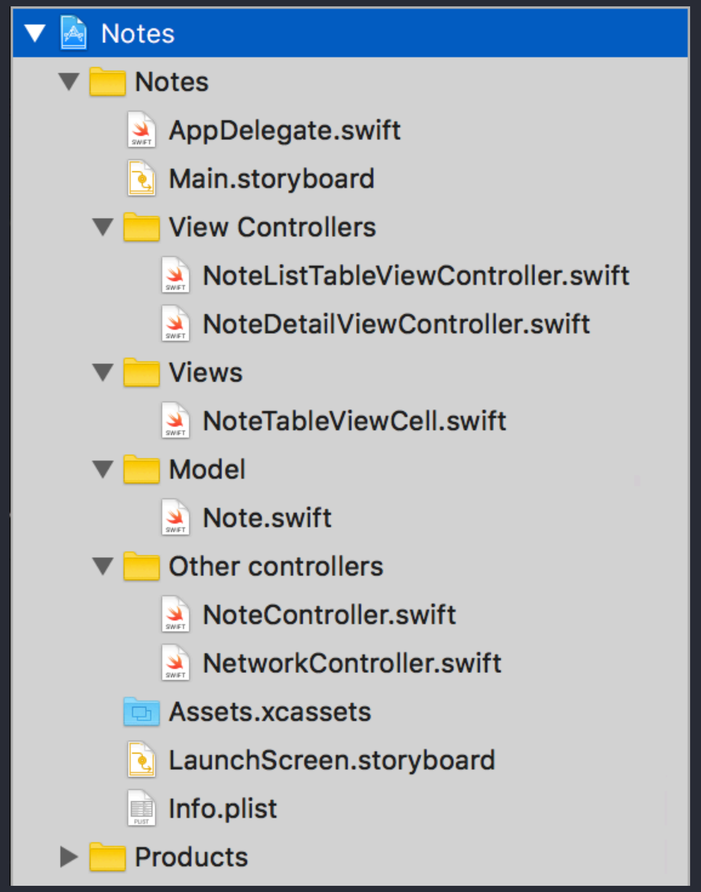
```
    // Navigation methods
    override func prepare(for segue: UIStoryboardSegue,
sender: Any?) {
        // Update the meal property that will be accessed
        // by the MealListTableViewController to update
        // the list of meals
    }

    @IBAction func cancel(_ sender: UIBarButtonItem) {
        // Dismiss the view without saving the meal
    }
```

Reminder

- → Model-View-Controller is a useful pattern
- → More than one way to implement it
- → Everyone has their own style
- → Yours will evolve as you gain experience

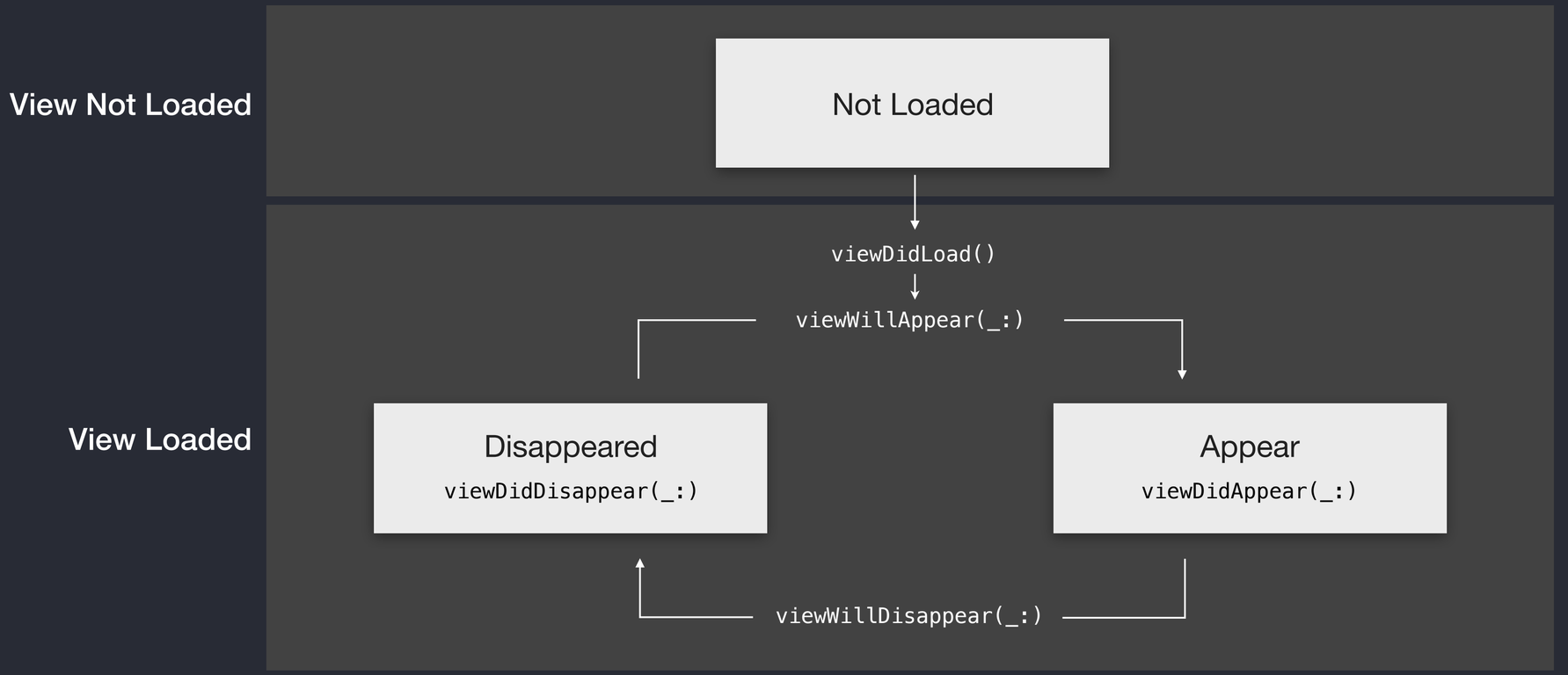
Project organization

- Use clear, descriptive filenames
- Create separate files for each of your type definitions
- Write your code as if complete strangers are going to read it
- Group files to help organize your code

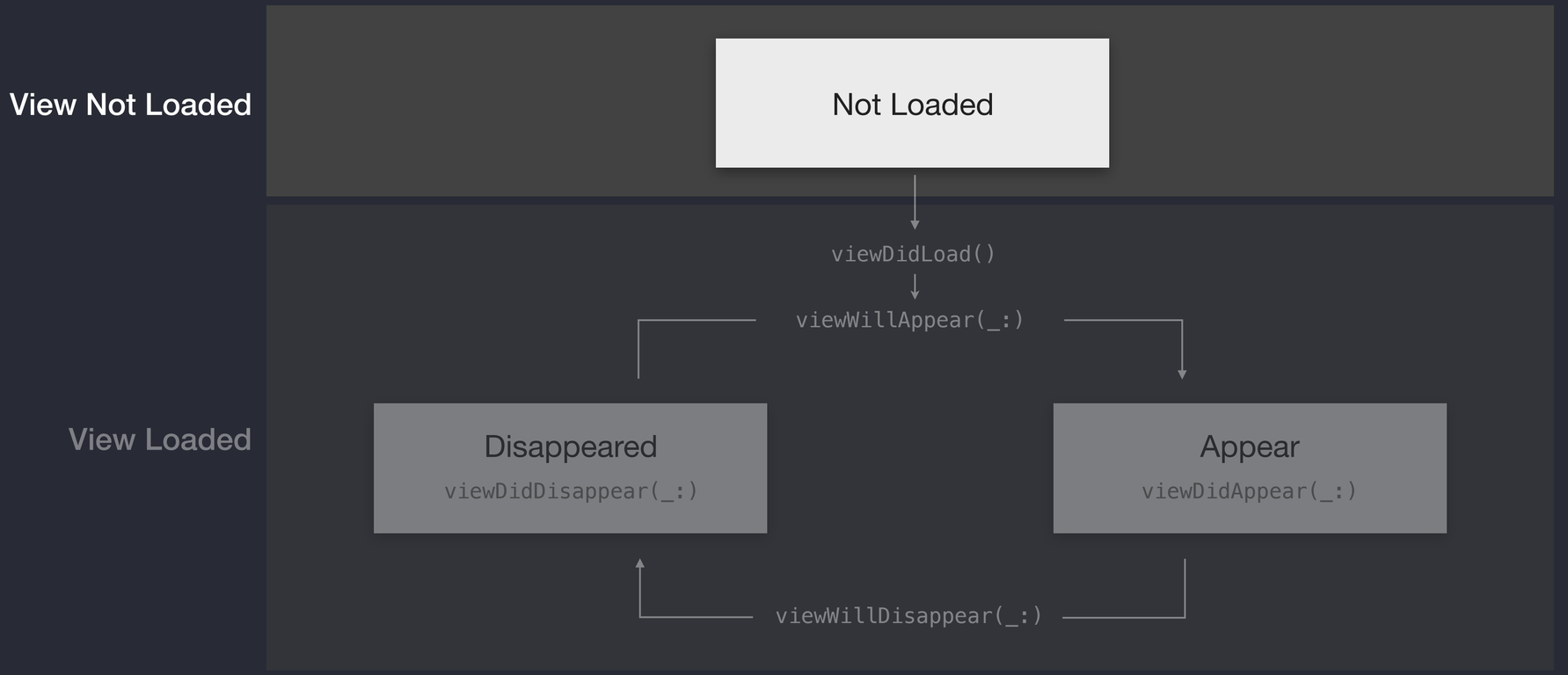


View Controller Life Cycle

View controller life cycle



View controller life cycle

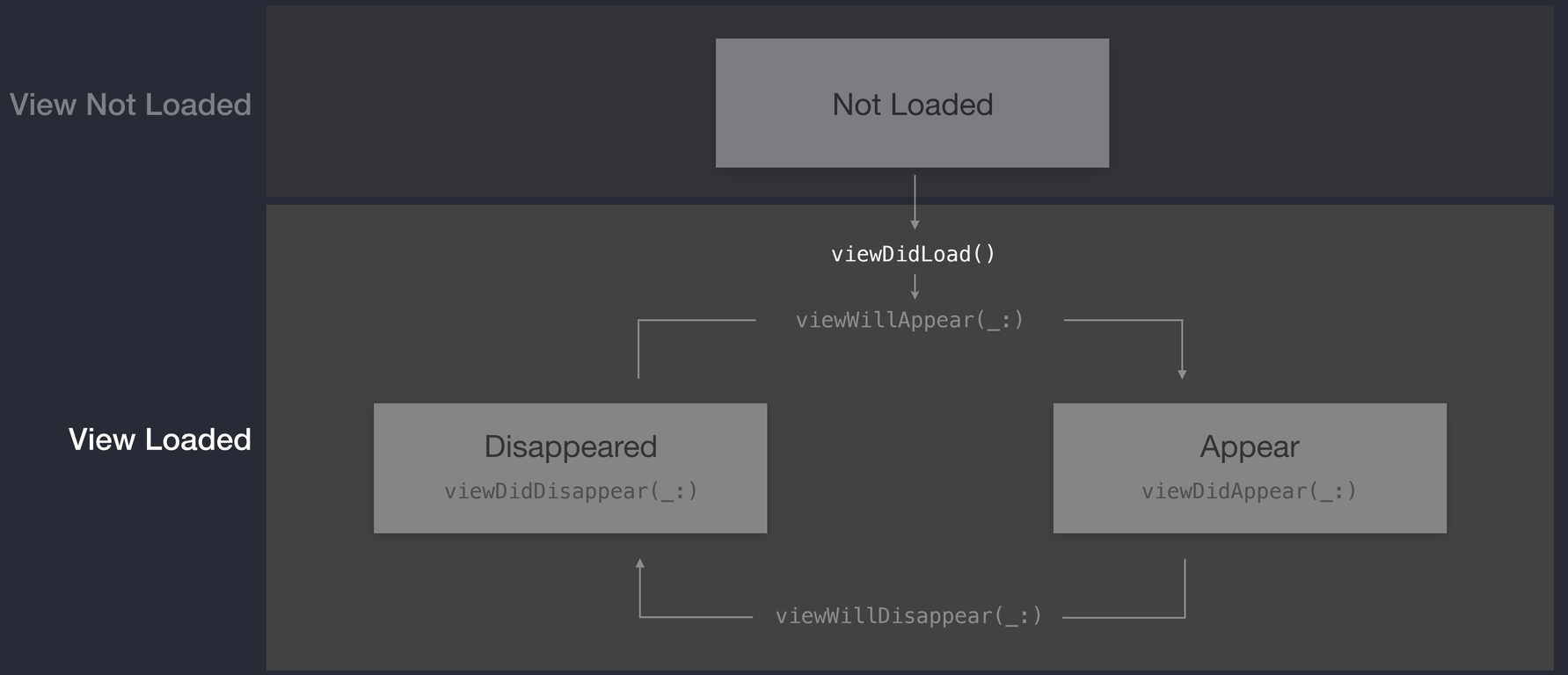


View event management

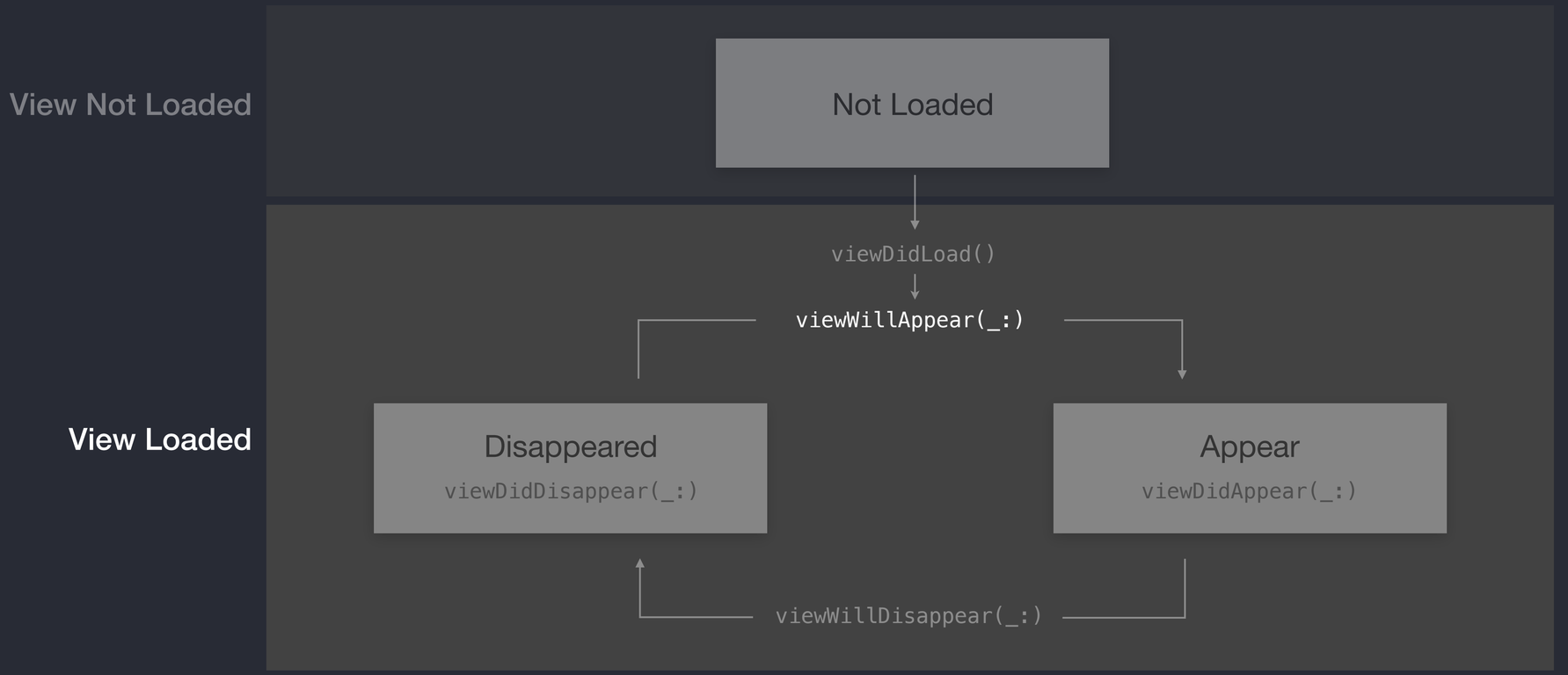
```
viewWillAppear(_:)
viewDidAppear(_:)
viewWillDisappear(_:)
viewDidDisappear(_:)
```

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    // Add your code here
}
```

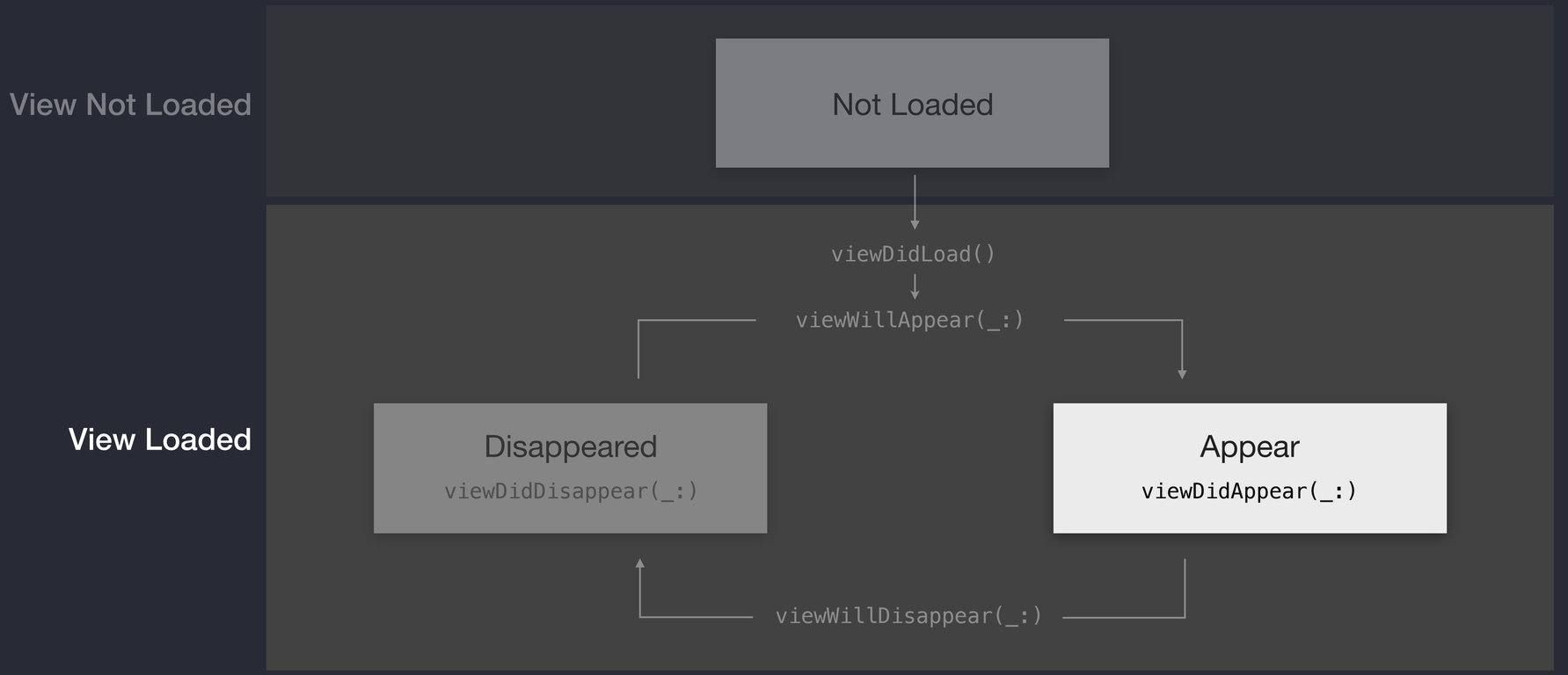
View event management



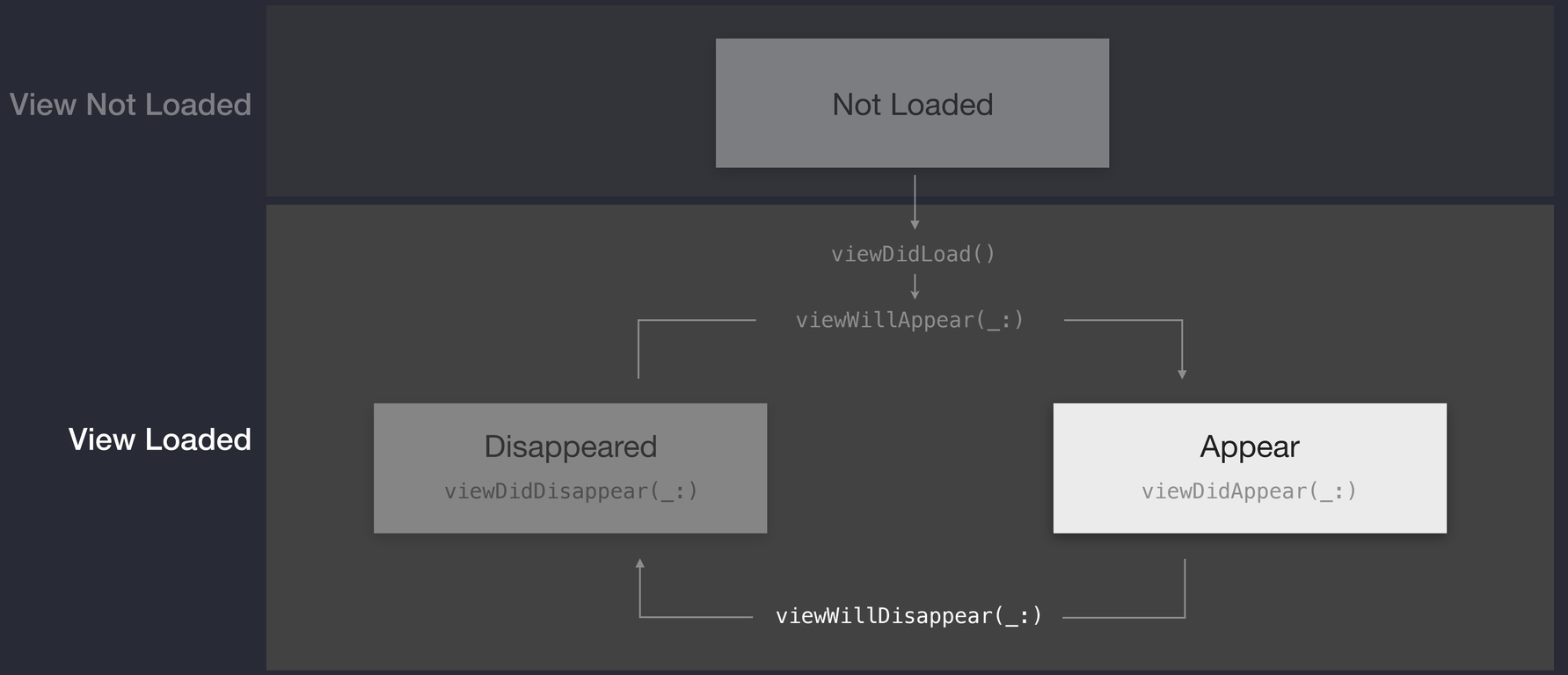
View event management



View event management

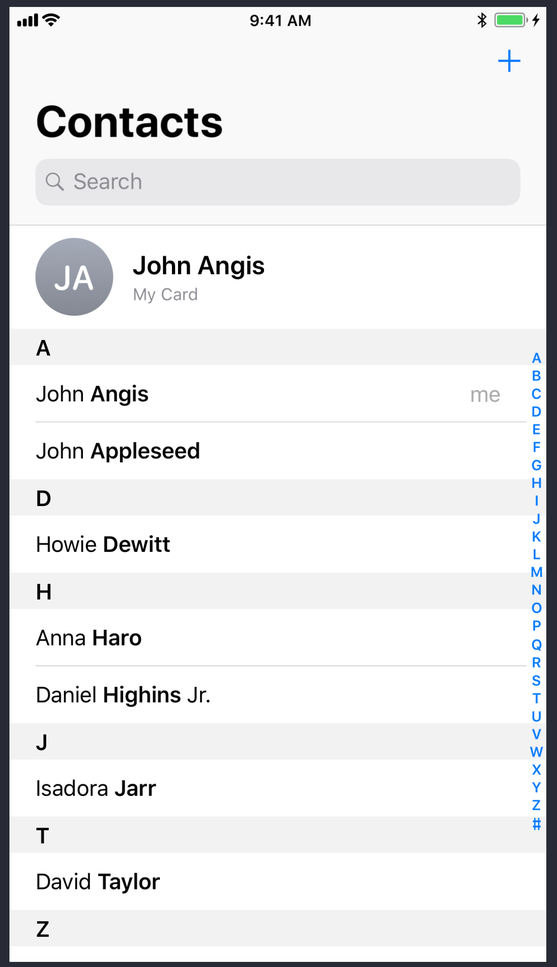


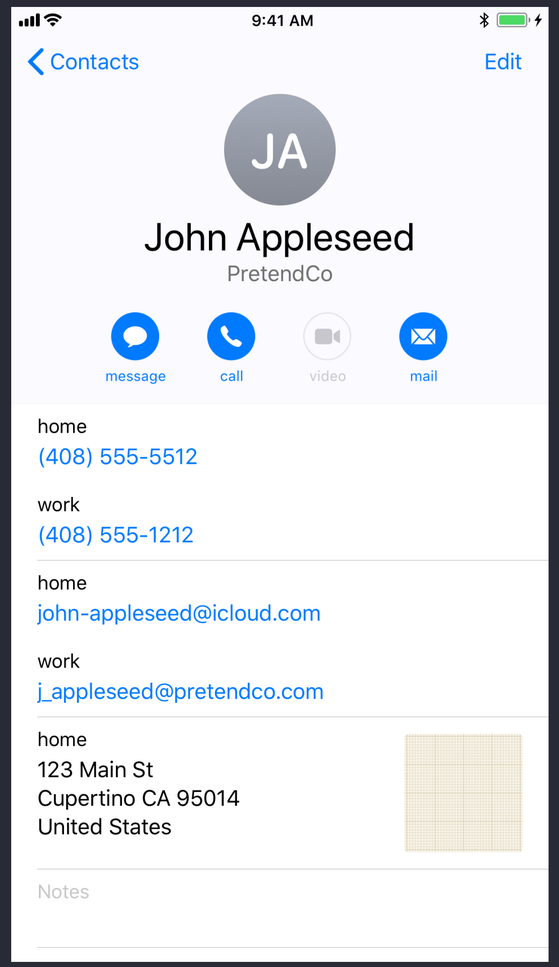
View event management



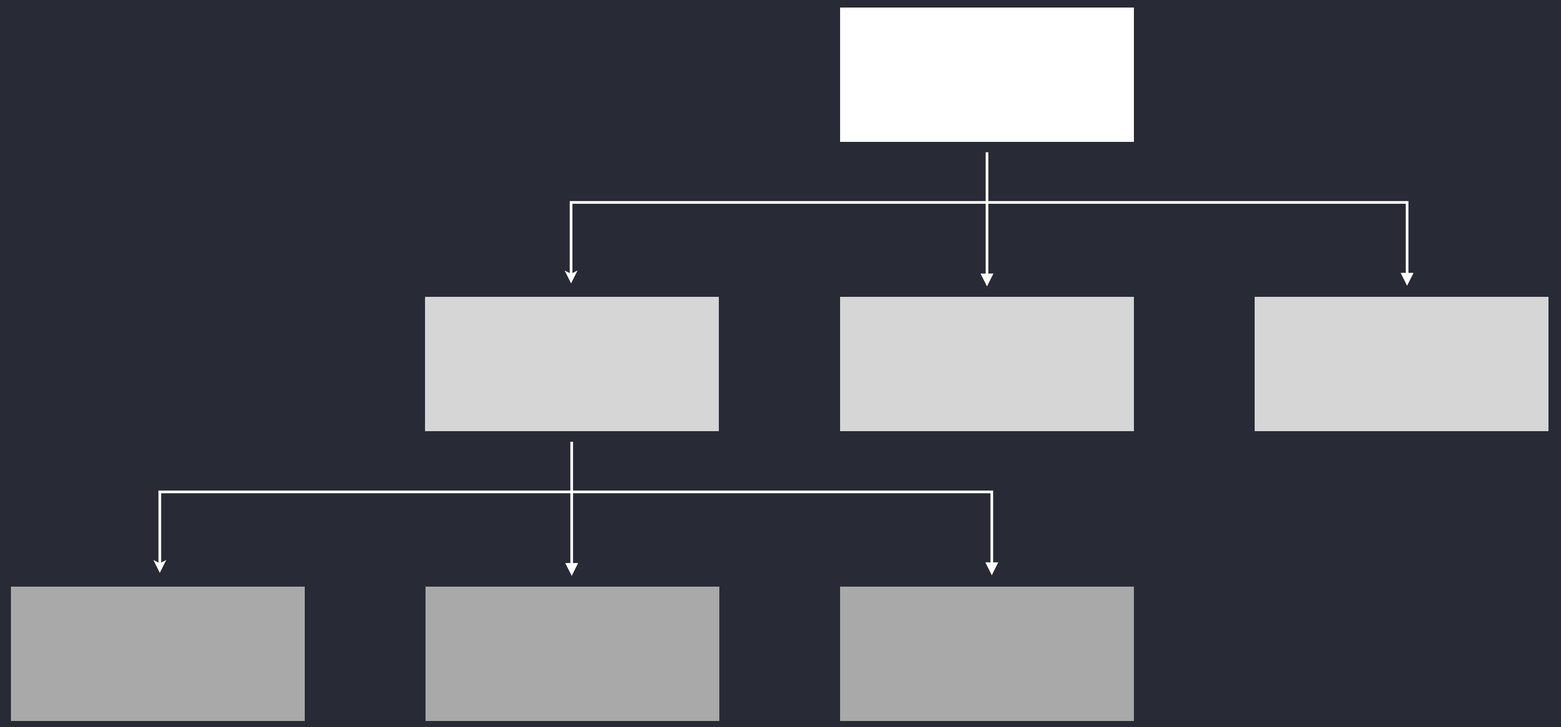
Segues and Navigation Controllers

Segues and navigation controllers





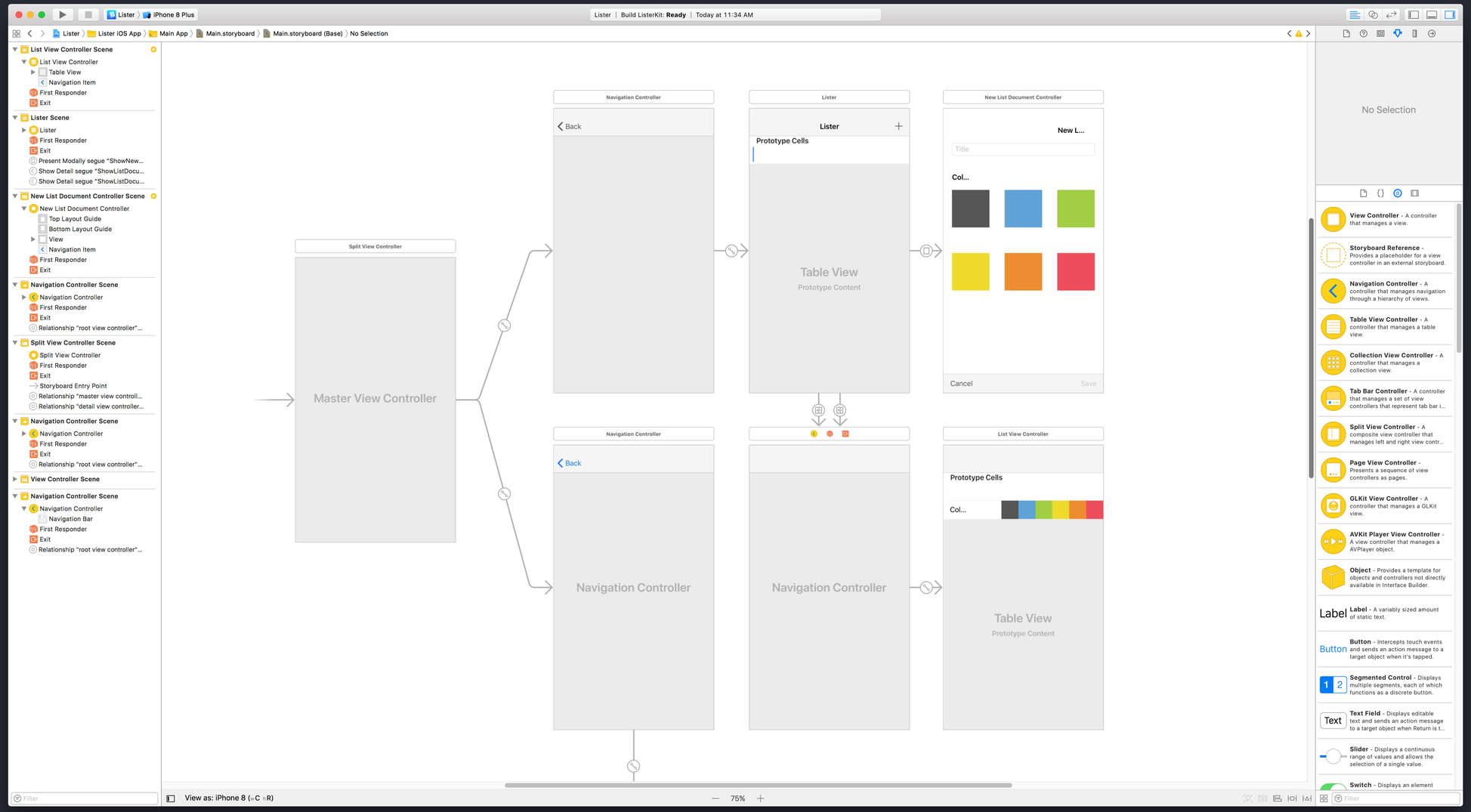
Navigation hierarchy



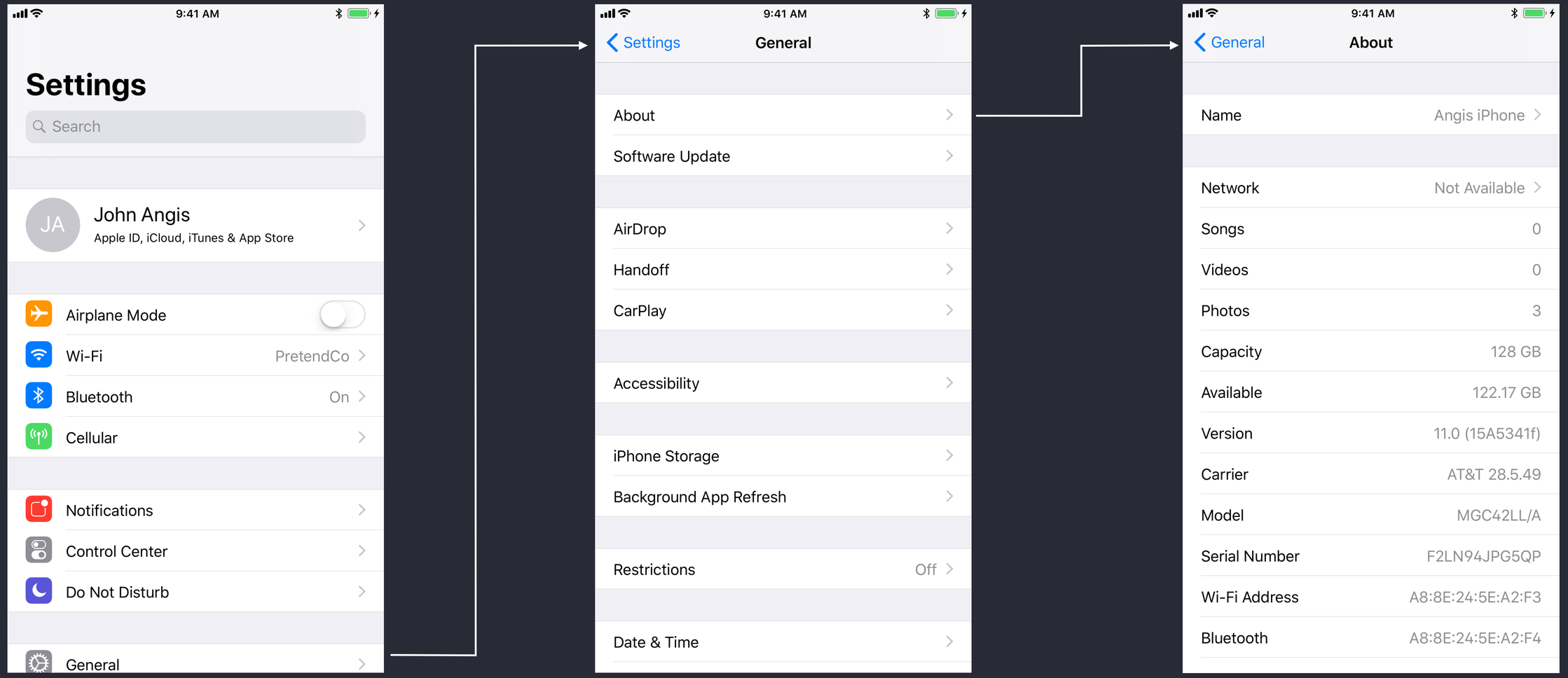
Segues (UIStoryboardSegue)

- A UIStoryboardSegue object performs the visual transition between two view controllers
- It is also used to prepare for the transition from one view controller to another
- Segue objects contain information about the view controllers that are involved in a transition
- When a segue is triggered, before the visual transition occurs, the storyboard runtime can call certain methods in the current view controller (useful if you need to pass information forward)

Segues (UIStoryboardSegue)



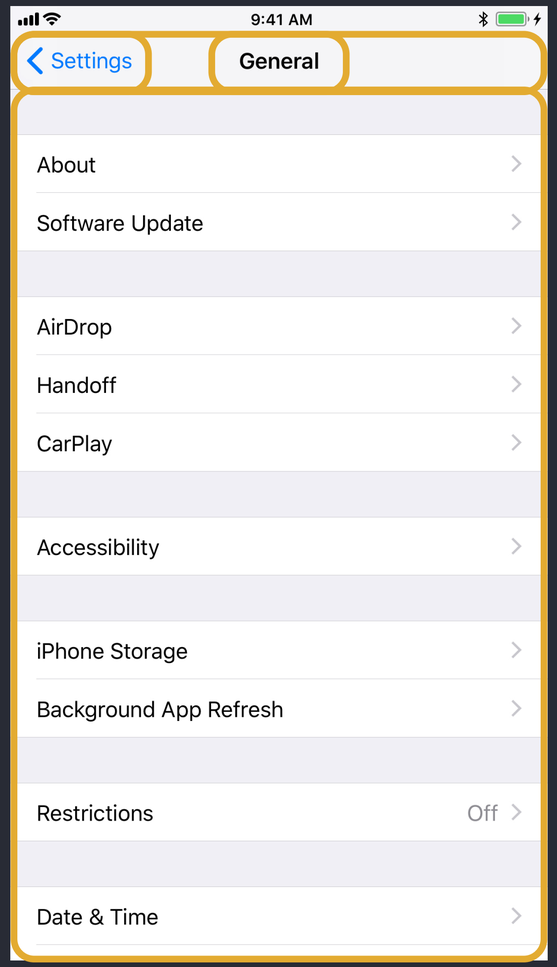
Navigation controller – UINavigationController



Navigation controller – UINavigationController

The top view controller's title

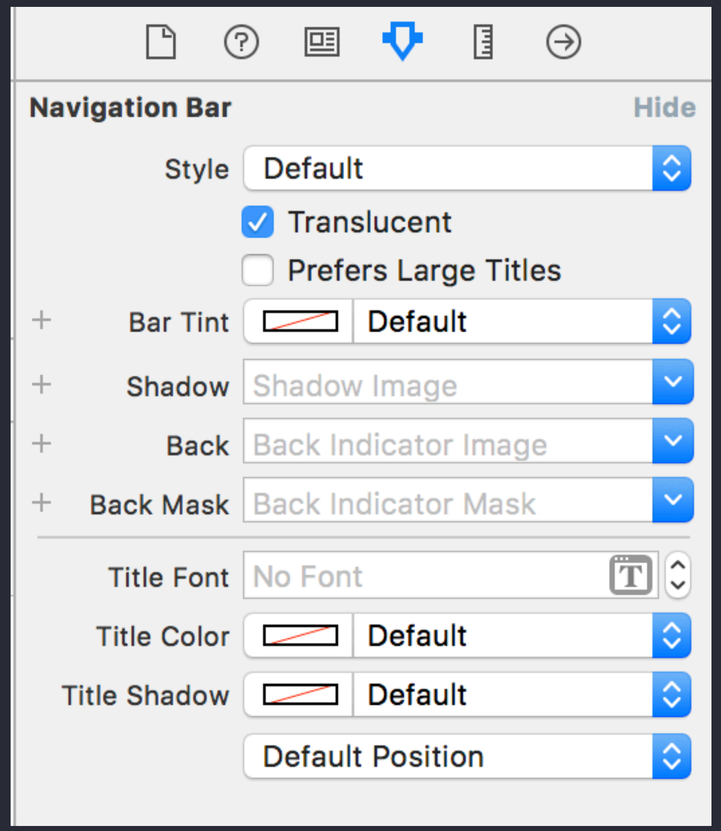
Back button

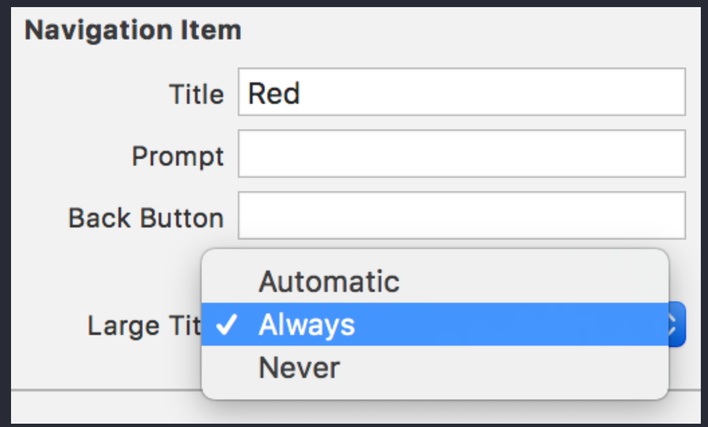


Navigation bar

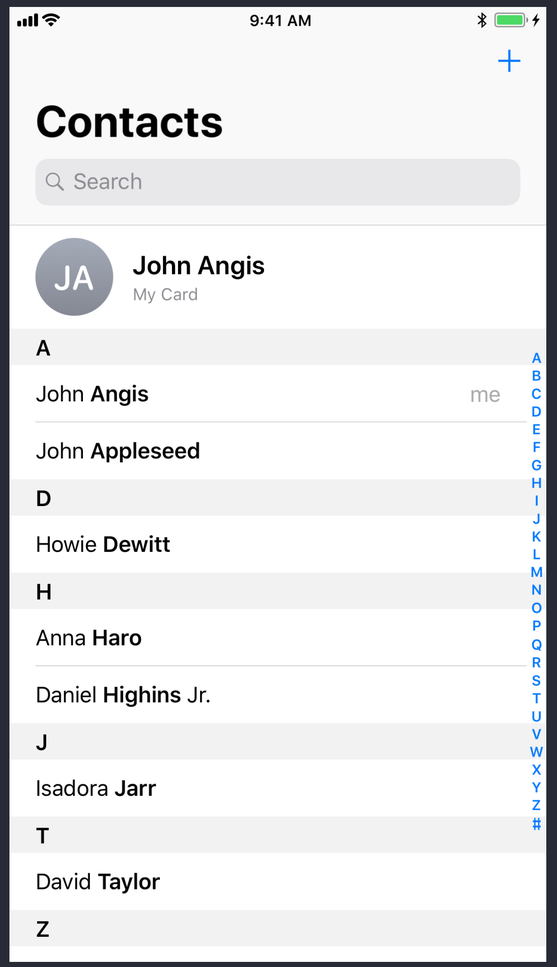
The top view controller's view

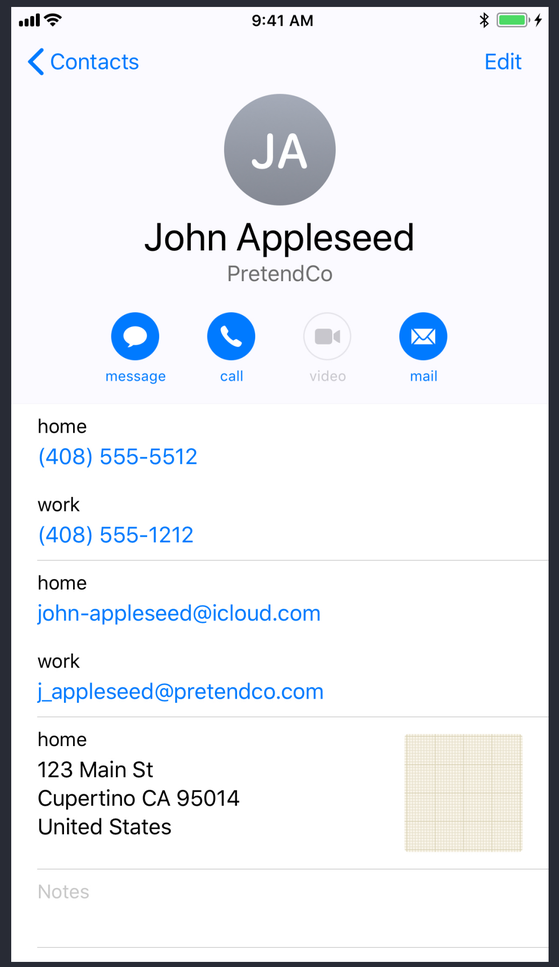
Navigation controller – UINavigationController





Pass information





Pass information

```
func prepare(for segue: UIStoryboardSegue, sender: Any?)
```

Segue properties

`identifier`
`destination`

```
override func prepare(for segue: UIStoryboardSegue, sender:
Any?) {
    segue.destination.navigationItem.title = textField.text
}
```

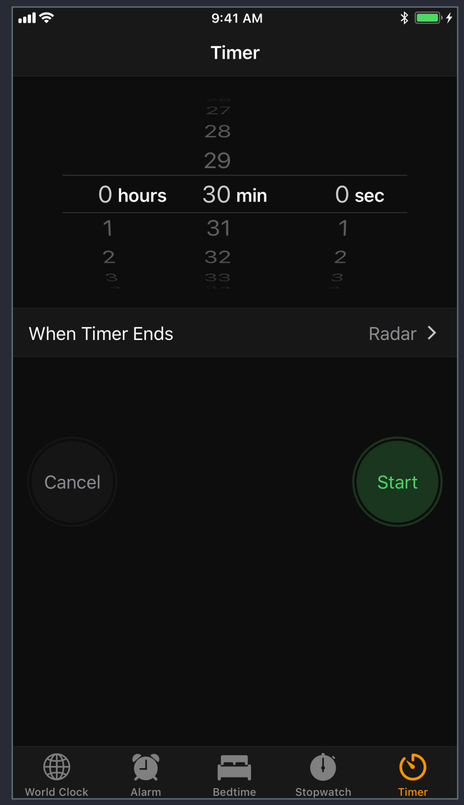
Programmatic segue

```
performSegue(withIdentifier: "ShowDetail", sender: nil)
```

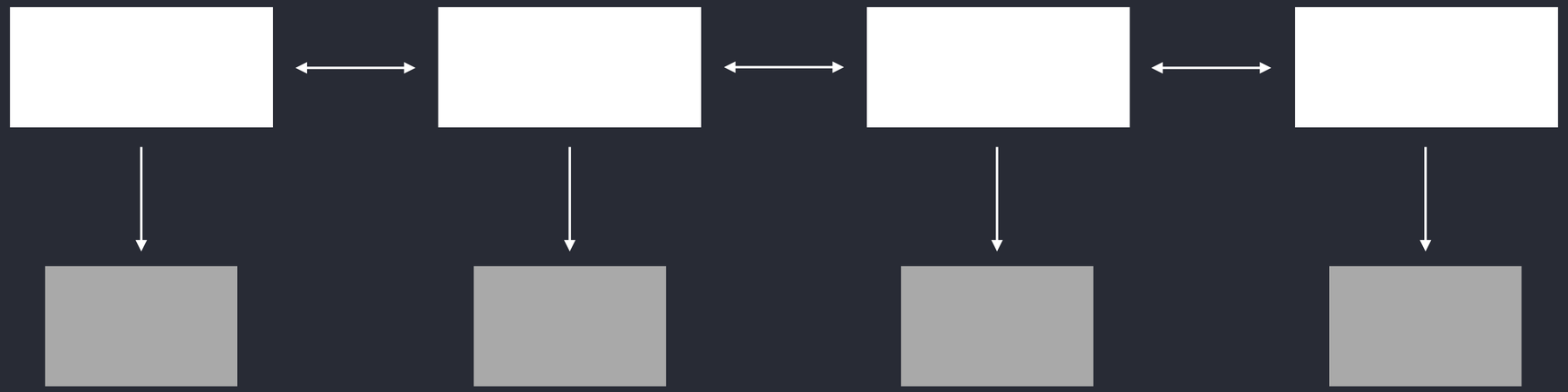
Tab Bar Controllers

UITabBarController

- A specialized view controller that manages a radio-style selection interface
- A tab bar is displayed at the bottom of the view



Navigation hierarchy

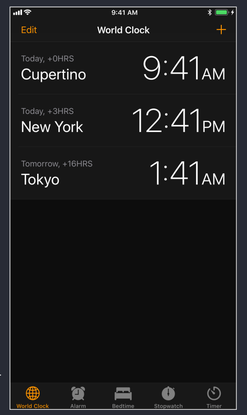


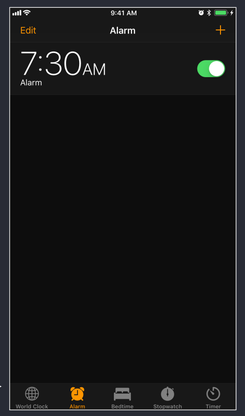
UITabBarController

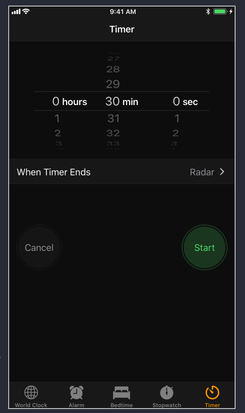
Tab Bar Controller

`viewControllers`

- World Clock Controller
- Alarm Controller
- Bedtime Controller
- Stopwatch Controller
- Timer Controller







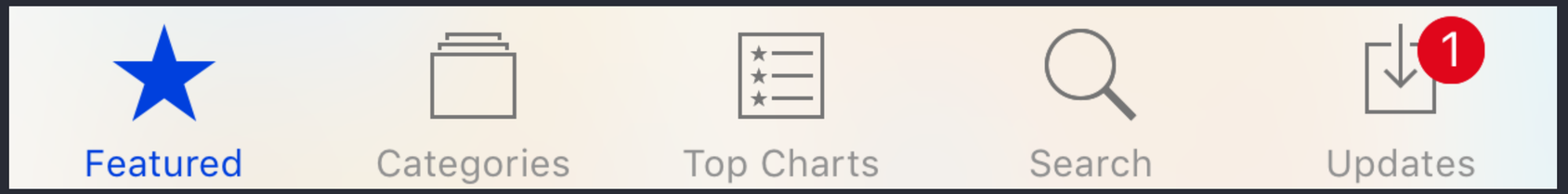
Add a tab bar controller

- Using a storyboard
- Drag in a UITabBarController from the object library

Add a tab bar controller

- Drag a new view controller object onto the canvas
- To create a segue, control-drag from the UITabBarController to the view controller
- Select "view controllers" under Relationship Segue

UITabBarItem




Search


Contacts


More


History


Recent


Most Recent


Featured


Most Viewed


Top Rated


Bookmarks


Favorites


Downloads

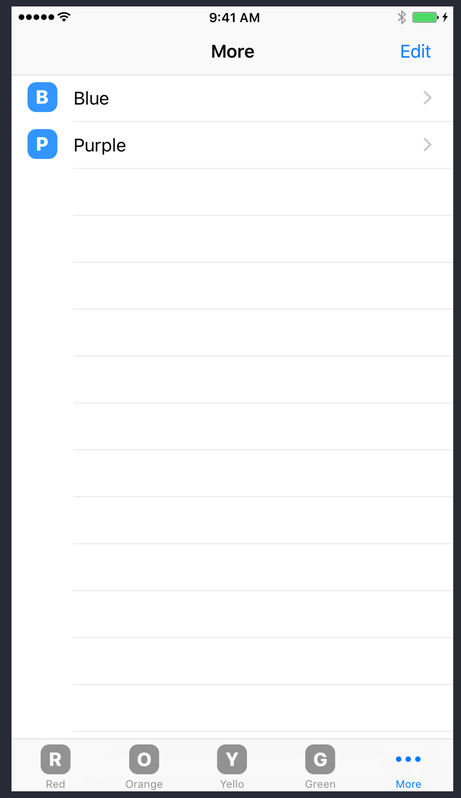
Programmatic customization

```
tabBarItem.badgeValue = "!"
```


```
tabBarItem.badgeValue = nil
```

Even more tab items

- More view controller:
  - Appears when needed
  - Can’t be customized
- If possible, plan app to avoid More



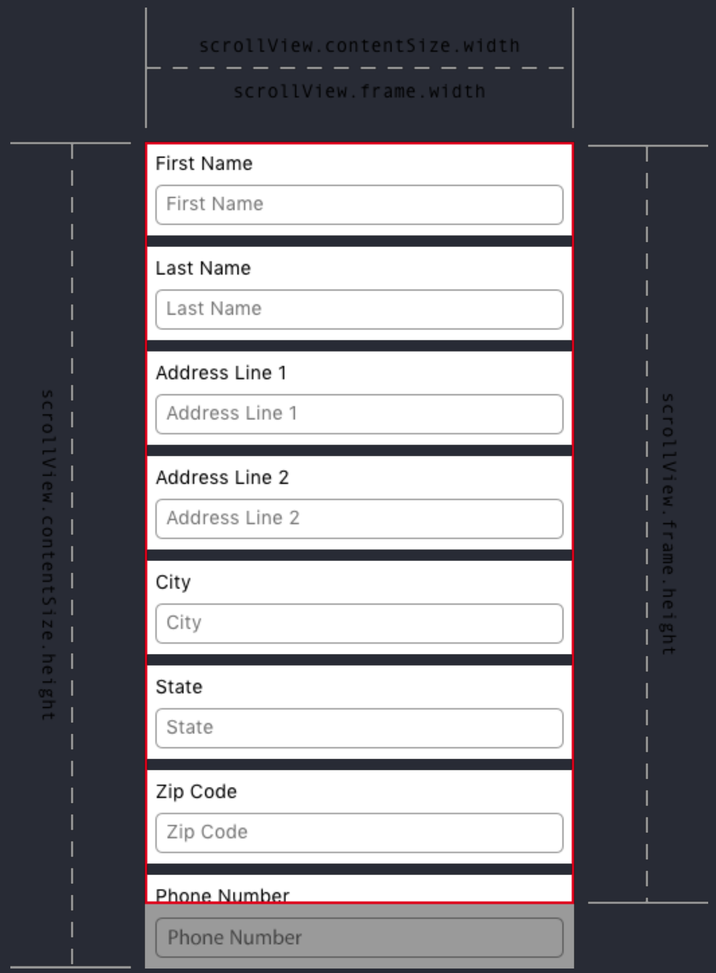
Scroll views

Scroll views

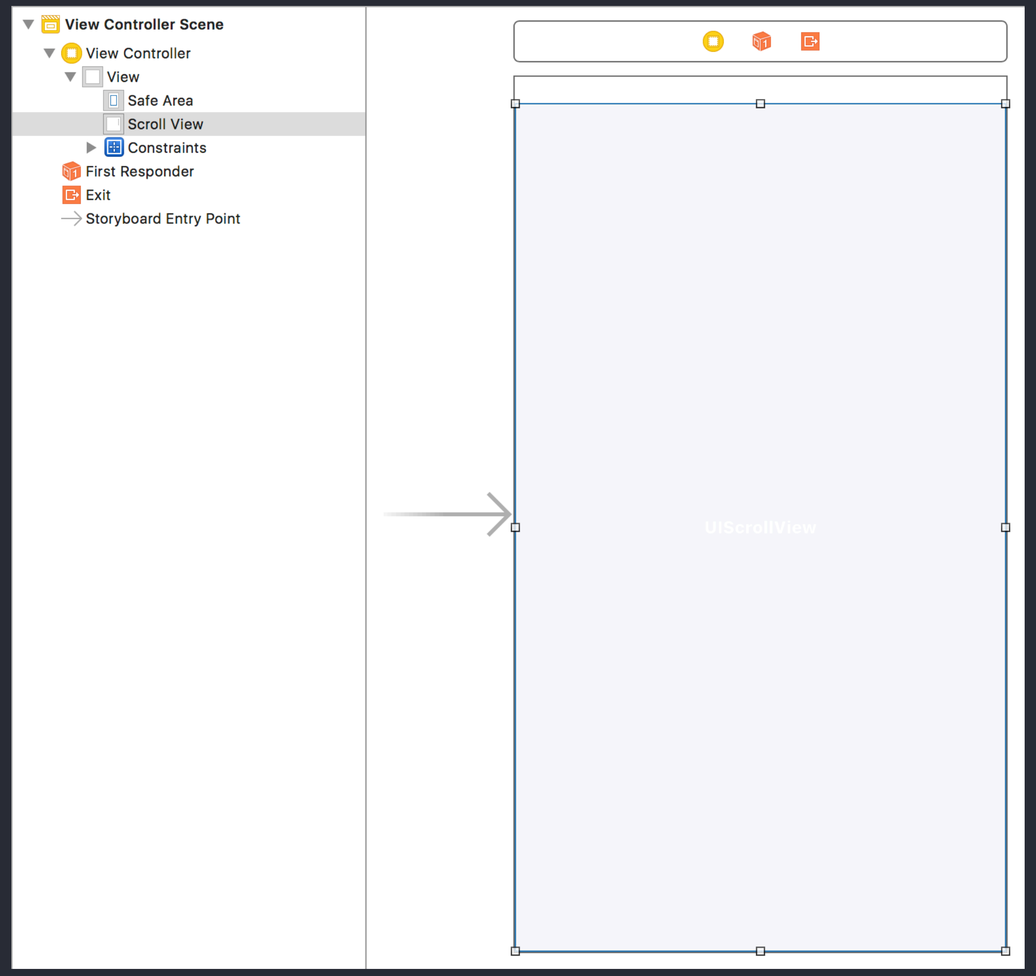
UIScrollView

- For displaying more content than can fit on the screen
- Users scroll within the content by making swiping gestures
- Content can optionally be zoomed with a pinch gesture
- UIScrollView needs to know the size of the content

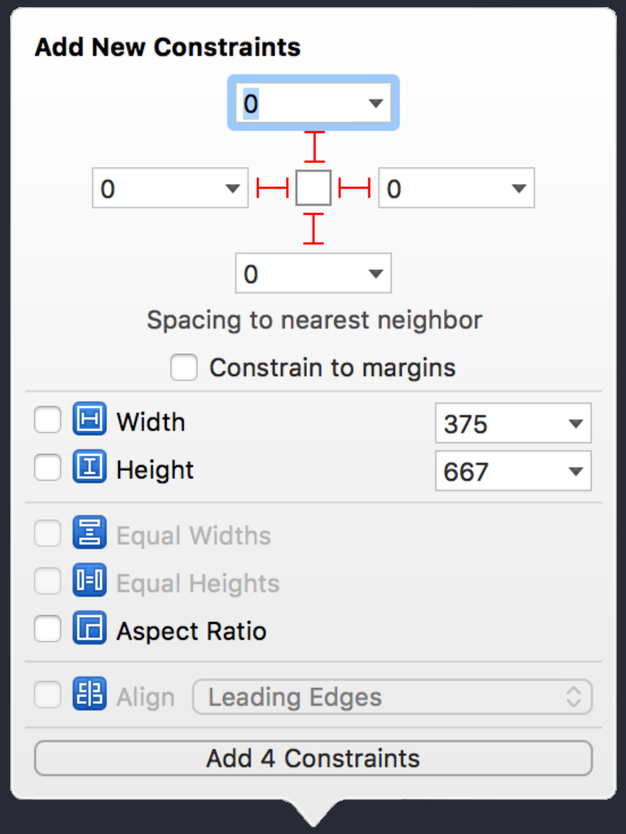
UIScrollView



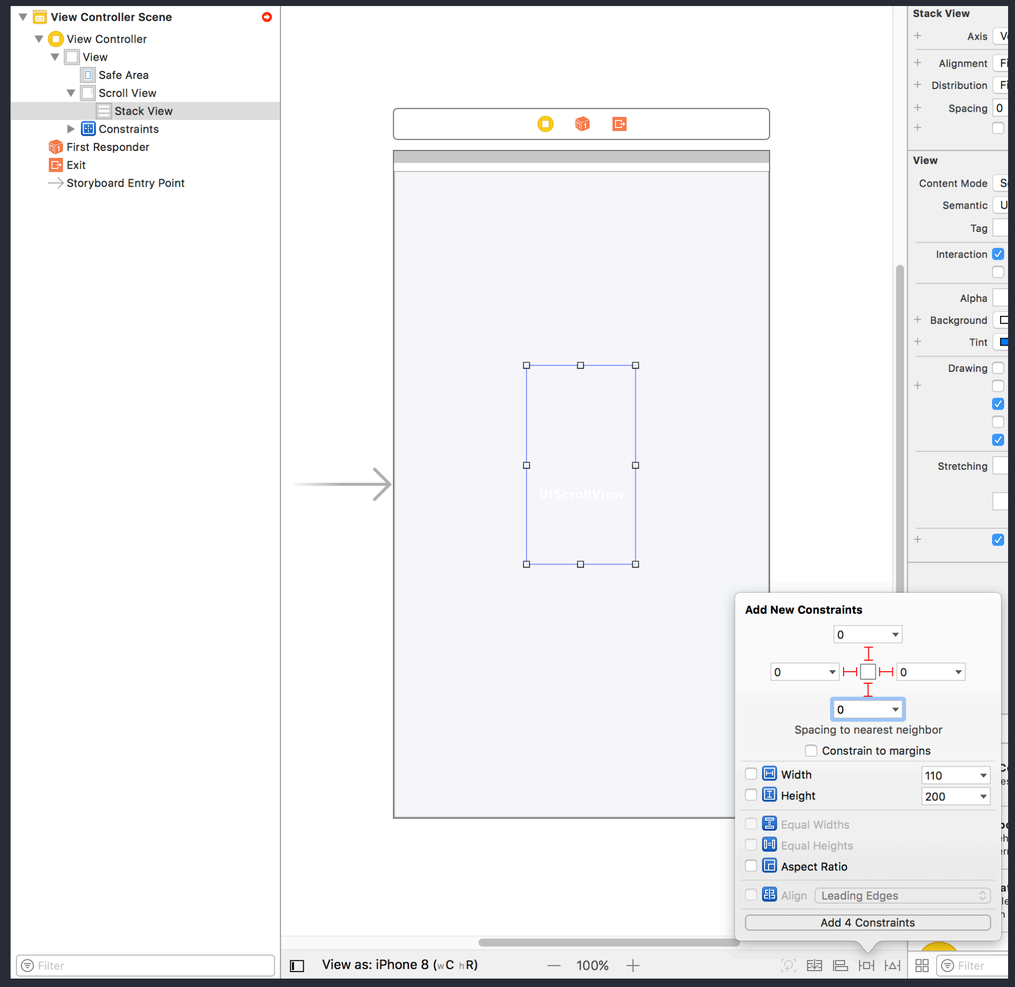
Scroll views in Interface Builder



Scroll views in Interface Builder



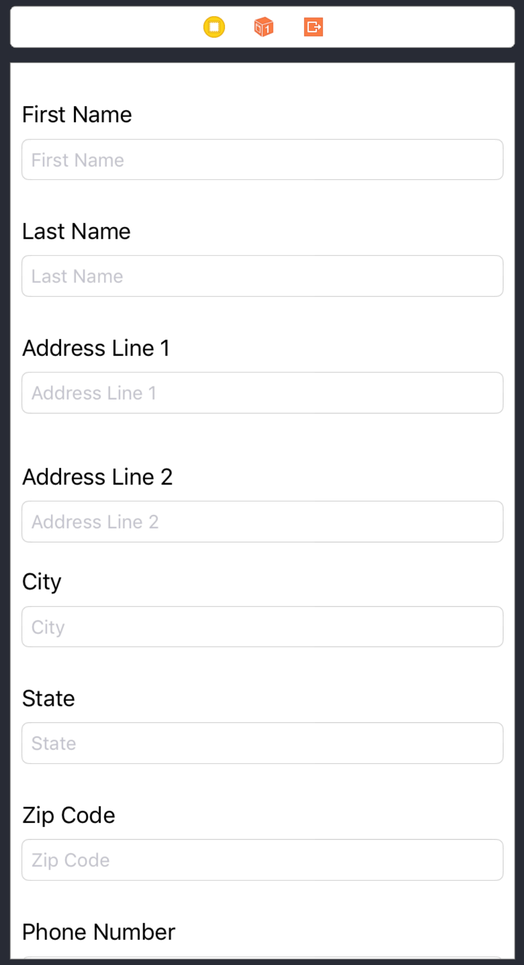
Scroll views in Interface Builder



Scroll views in Interface Builder

```
imageView.centerXAnchor.constraints(equalTo: scrollView.contentLayoutGuide.centerXAnchor)
imageView.centerYAnchor.constraints(equalTo: scrollView.contentLayoutGuide.centerYAnchor)
```

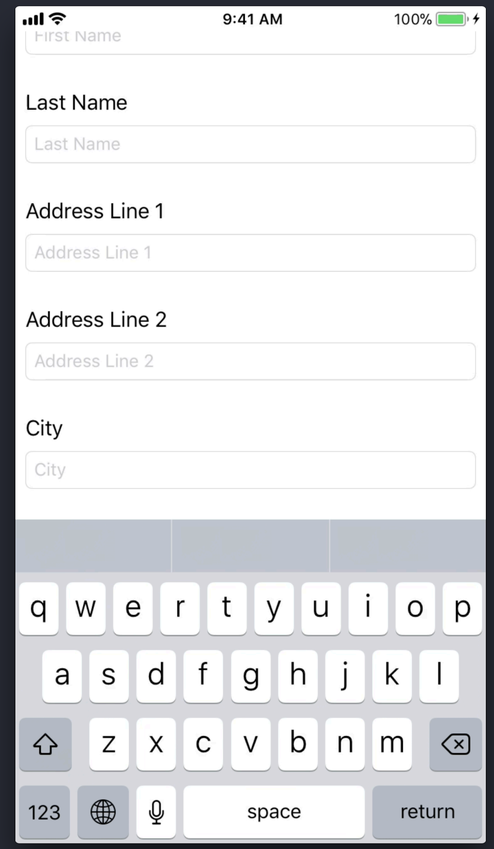
Scroll views in Interface Builder



Keyboard issues

- Sent a notification when the keyboard has been shown or will be hidden
- Register for keyboard notifications

```
func registerForKeyboardNotifications() {
  NotificationCenter.default.addObserver(self,
selector: #selector(keyboardWasShown(_:)),
name: .UIKeyboardDidShow, object: nil)
  NotificationCenter.default.addObserver(self,
selector: #selector(keyboardWillBeHidden(_:)),
name: .UIKeyboardWillHide, object: nil)
}
```

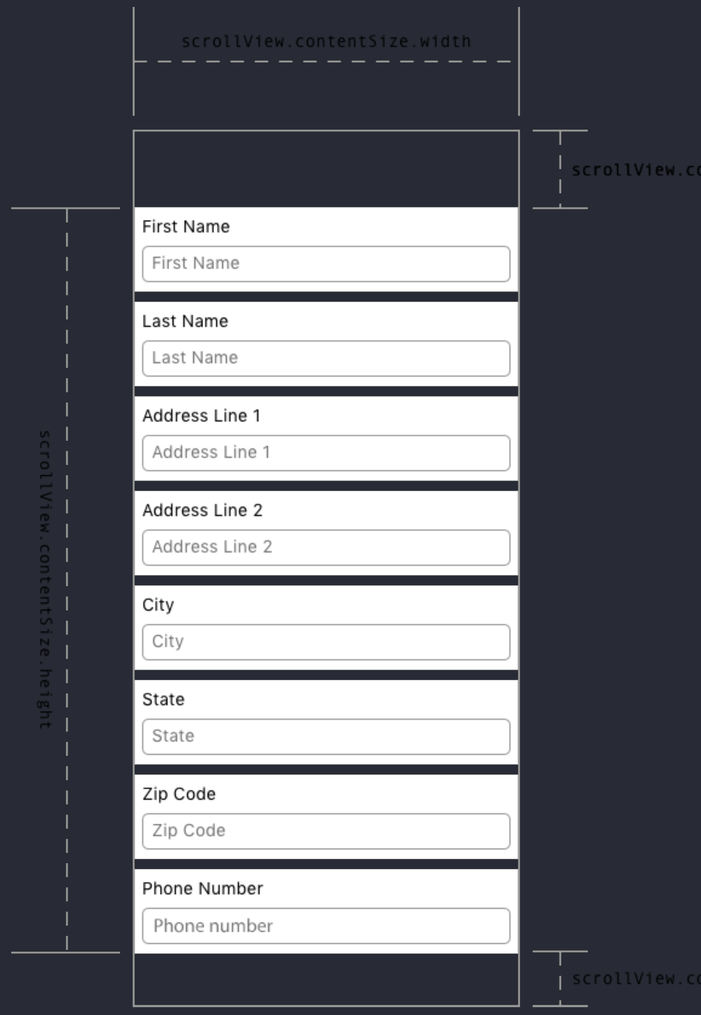


```
func keyboardWasShown(_ notificiation: NSNotification) {
    guard let info = notificiation.userInfo,
        let keyboardFrameValue =
info[UIKeyboardFrameBeginUserInfoKey] as? NSValue else { return }

    let keyboardFrame = keyboardFrameValue.cgRectValue
    let keyboardSize = keyboardFrame.size



    let contentInsets = UIEdgeInsetsMake(0.0, 0.0,
keyboardSize.height, 0.0)
    scrollView.contentInset = contentInsets
    scrollView.scrollIndicatorInsets = contentInsets
}

func keyboardWillBeHidden(_ notification: NSNotification) {
    let contentInsets = UIEdgeInsets.zero
    scrollView.contentInset = contentInsets
    scrollView.scrollIndicatorInsets = contentInsets
}
```

Content insets

- Allows you to pad the content at the top and bottom of the scroll view
- Useful if you have toolbars floating above your scroll view

```
scrollview.contentInset.top
                       .bottom
                       .left
                       .right
```



Scroll indicator

```
let contentInsets = UIEdgeInsetsMake(0.0, 0.0,
     keyboardSize.height, 0.0)
scrollView.contentInset = contentInsets
scrollView.scrollIndicatorInsets = contentInsets
```

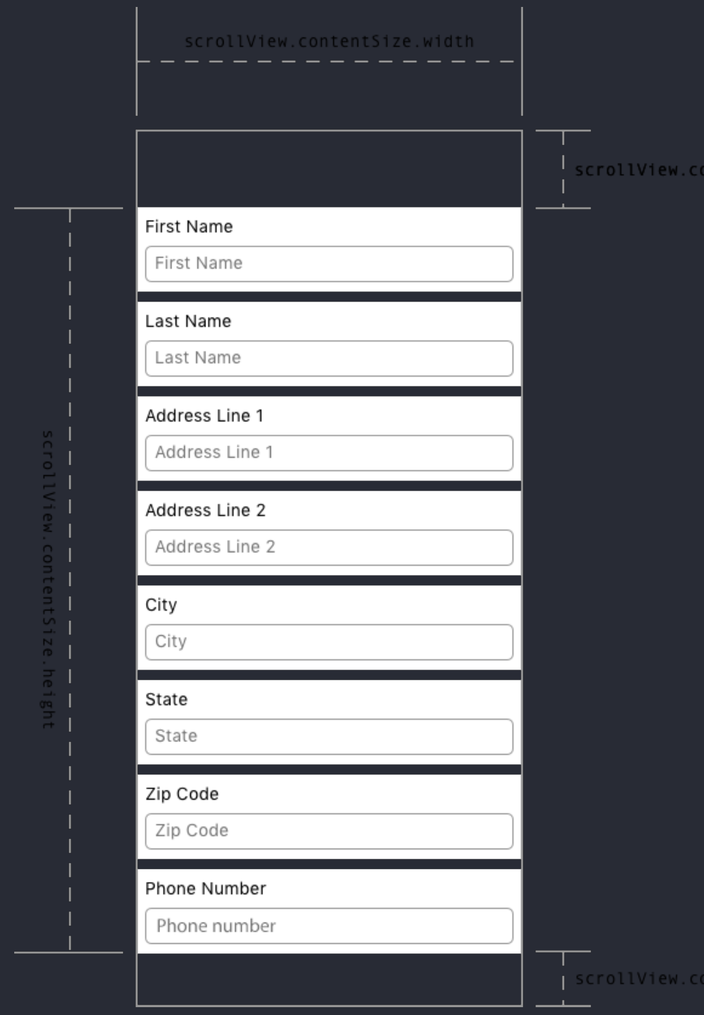


Table views

Table views

An instance of the UITableView class

A subclass of UIScrollView

→ Displays a list of items

→ Displays one or possibly thousands of data objects

→ Presents vertical scrolling and single-column, multiple rows

→ Provides customizable options

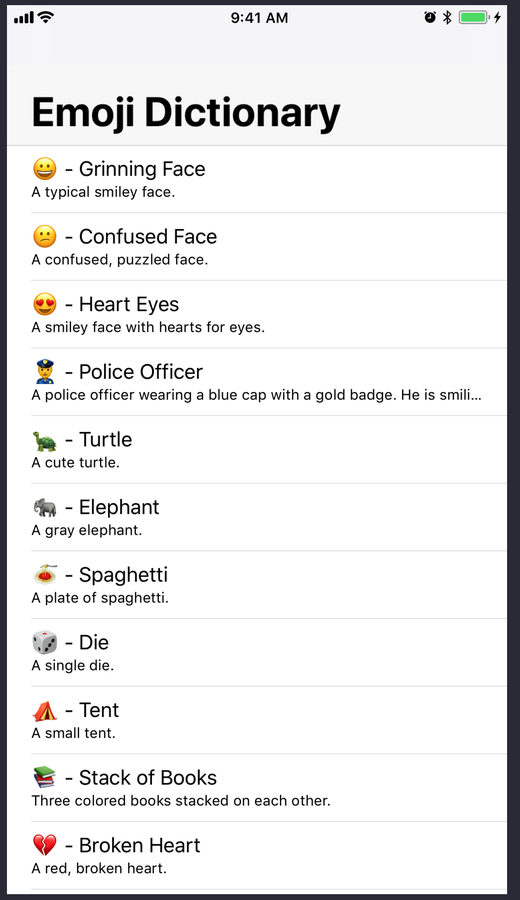
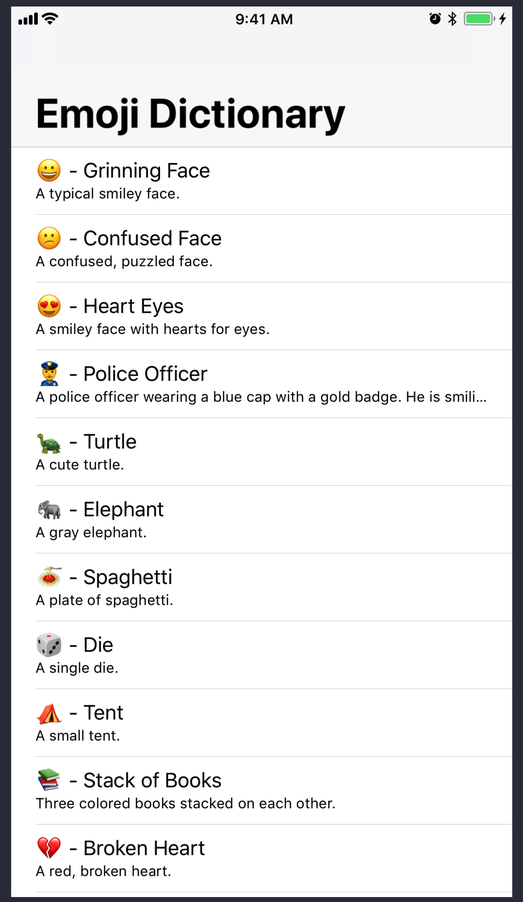
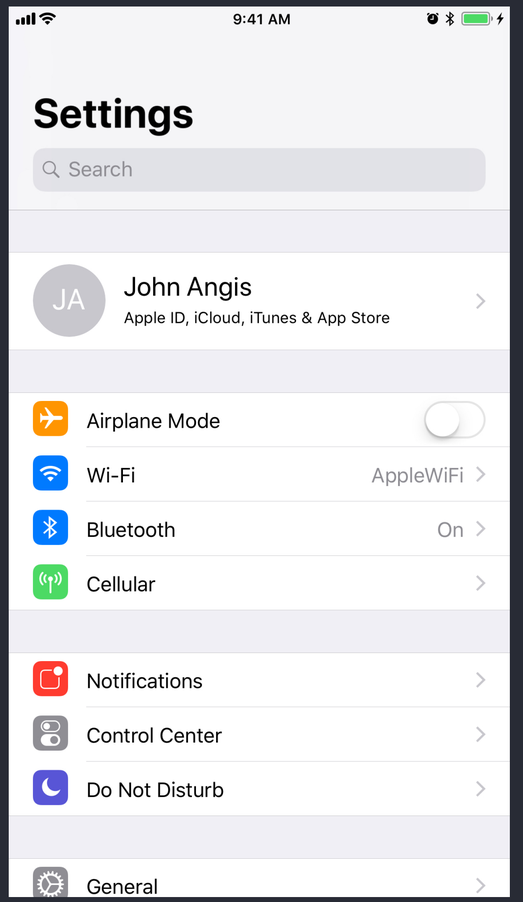


Table views

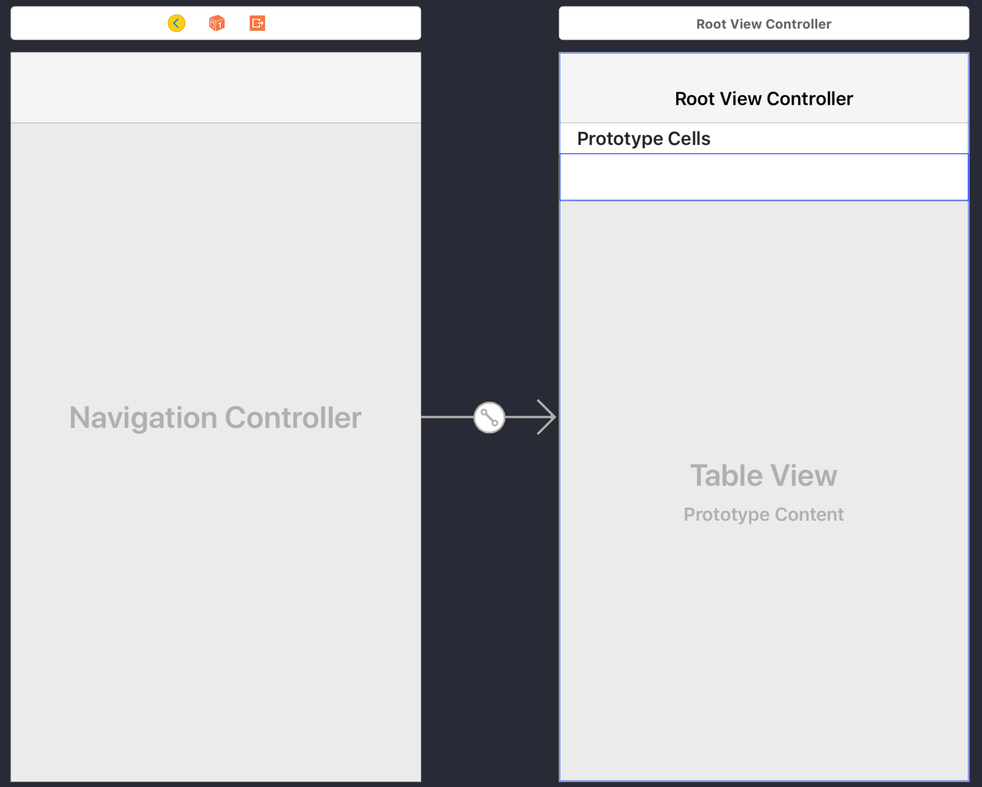

9:41 AM
Emoji Dictionary
😀 - Grinning Face
A typical smiley face.
😕 - Confused Face
A confused, puzzled face.
😍 - Heart Eyes
A smiley face with hearts for eyes.
👮 - Police Officer
A police officer wearing a blue cap with a gold badge. He is smili...
🐢 - Turtle
A cute turtle.
🐘 - Elephant
A gray elephant.
🍝 - Spaghetti
A plate of spaghetti.
🎲 - Die
A single die.
⛺ - Tent
A small tent.
📚 - Stack of Books
Three colored books stacked on each other.
💔 - Broken Heart
A red, broken heart.


9:41 AM
Settings
Search
JA
John Angis
Apple ID, iCloud, iTunes & App Store
Airplane Mode
Wi-Fi
AppleWiFi
Bluetooth
On
Cellular
Notifications
Control Center
Do Not Disturb
General

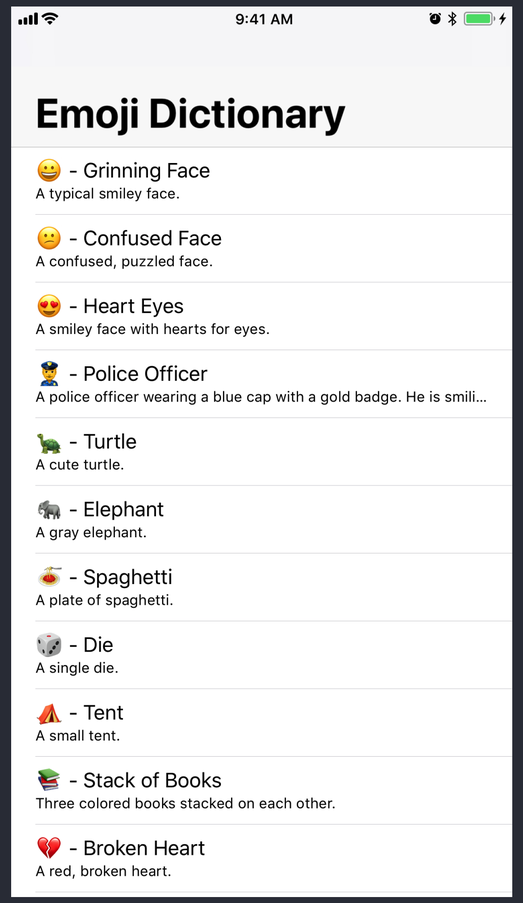
Anatomy of a table view

Two possible approaches to add table views:

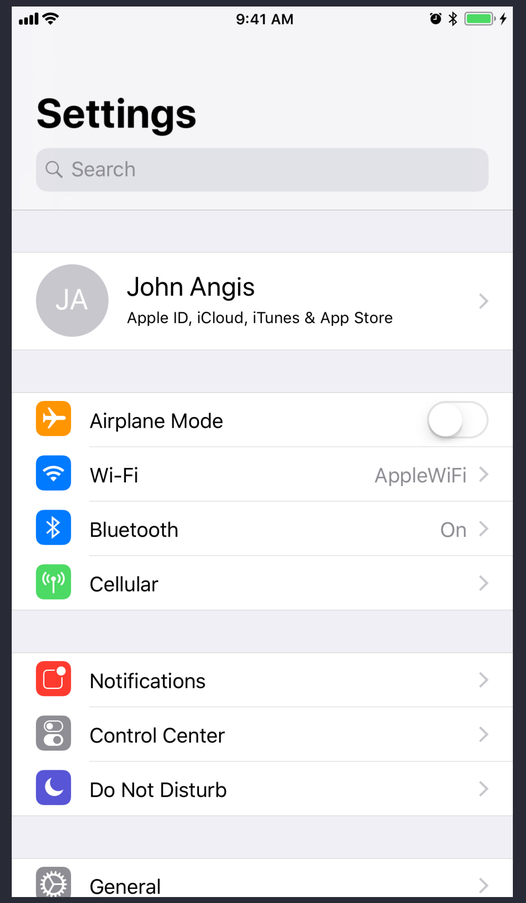
- Add a table view instance directly to a view controller's view
- Add a table view controller to your storyboard



Anatomy of a table view



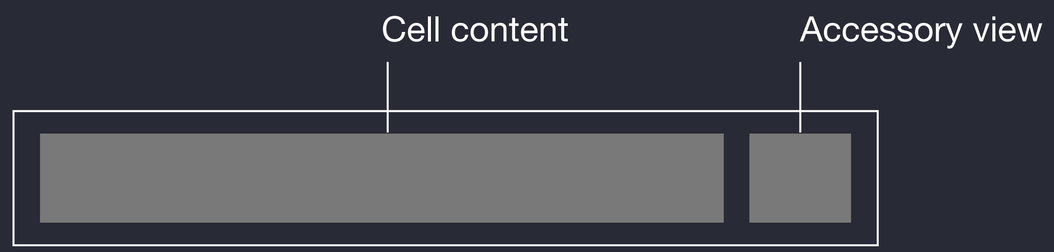
Plain



Grouped

Anatomy of a table view - cells

Every row is represented with a table view cell



In editing mode, the cell content shrinks

Anatomy of a table view - cells

`UITableViewCell` class defines three properties for cell content

Cell property	Description
textLabel	`UILabel` for the title
detailTextLabel	`UILabel` for the subtitle
imageView	`UIImageView` for an image

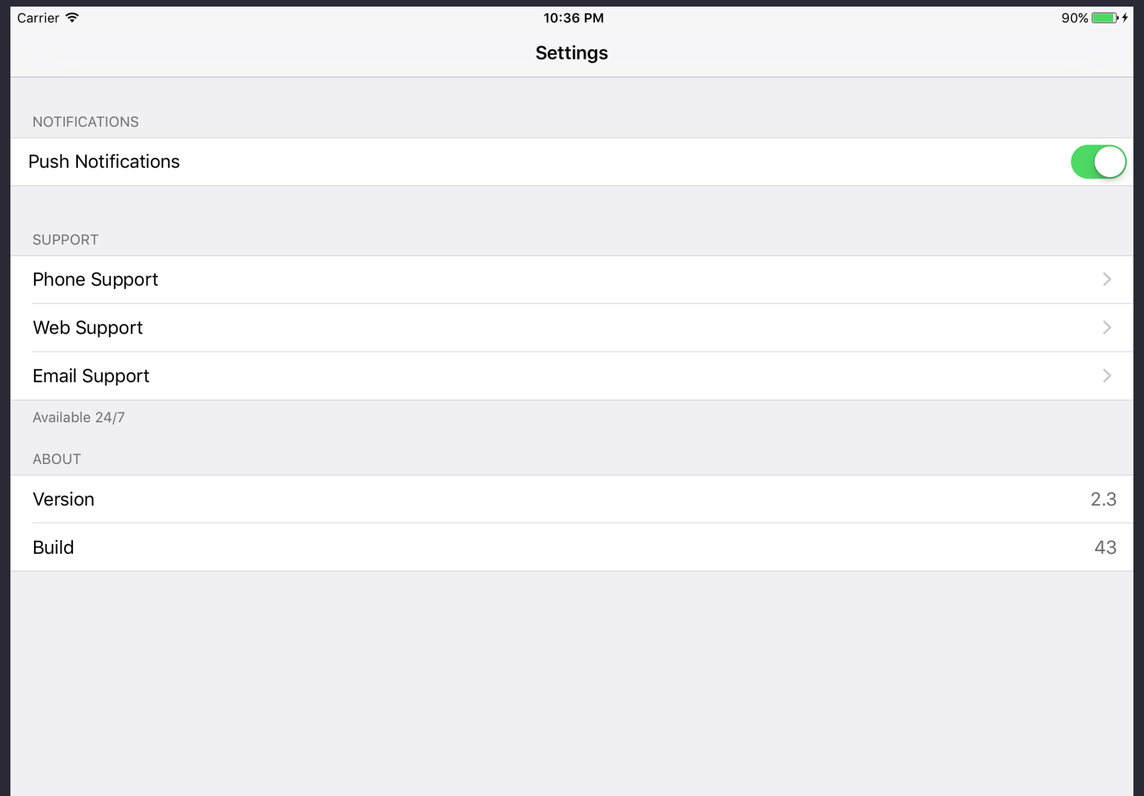
Anatomy of a table view - cells

Storyboard name	Programmatic enum name	Displays
Basic	.default	textLabel, imageView
Subtitle	.subtitle	textlabel , detailTextLabel, imageView
Right detail	.value1	textlabel , detailTextLabel, imageView
Left detail	.value2	textLabel , detailTextLabel

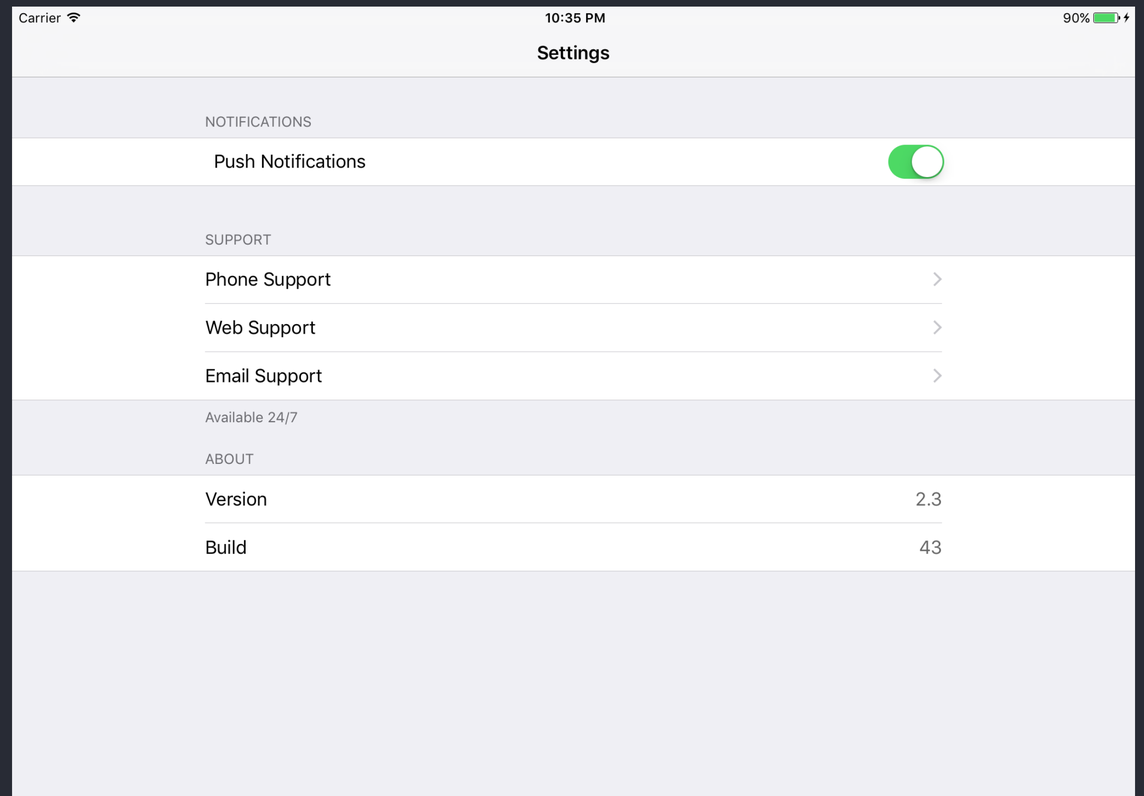
Anatomy of a table view - cells

→ Set `tableView.cellLayoutMarginsFollowReadableWidth` to true

Default



Adjusted



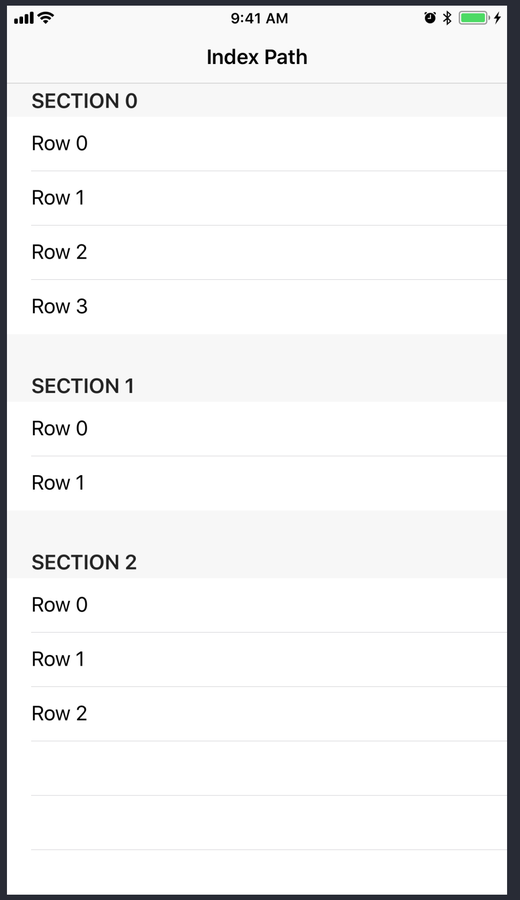
Index paths

Points to a specific row in a specific section

Accessible through the row and section properties

- `indexPath.row`
- `indexPath.section`

Values are zero-based



Arrays and table views

- → Collection of similar data
- → Typically backed by a collection of model objects

```
var emojis: [Emoji] =
[Emoji(symbol: Character("😀"), name: "Grinning Face", description: "A
typical smiley face.", usage: "happiness"),
 Emoji(symbol: Character("😕"), name: "Confused Face", description: "A
confused, puzzled face.", usage: "unsure what to think; displeasure"),
 Emoji(symbol: Character("😍"), name: "Heart Eyes", description: "A
smiley face with hearts for eyes.", usage: "love of something;
attractive")]
```

Arrays and table views

Cell dequeuing

- Table views only load visible cells
- Saves memory
- Allows for a smooth flow when scrolling

```
let cell: UITableViewCell =
tableView.dequeueReusableCell(withIdentifier: "Cell", for:
indexPath)
```

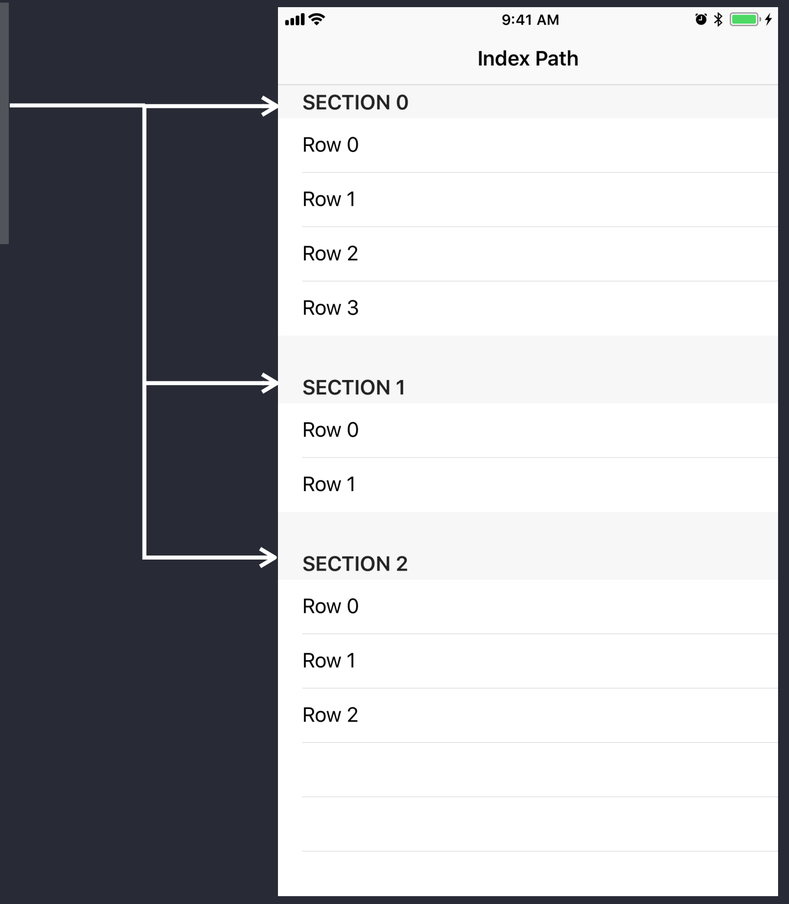
Table view protocols

- `UITableViewDataSource`
  Provides data for populating sections and rows
- `UITableViewDelegate`
  Customizes appearance and behavior

UITableViewDataSource

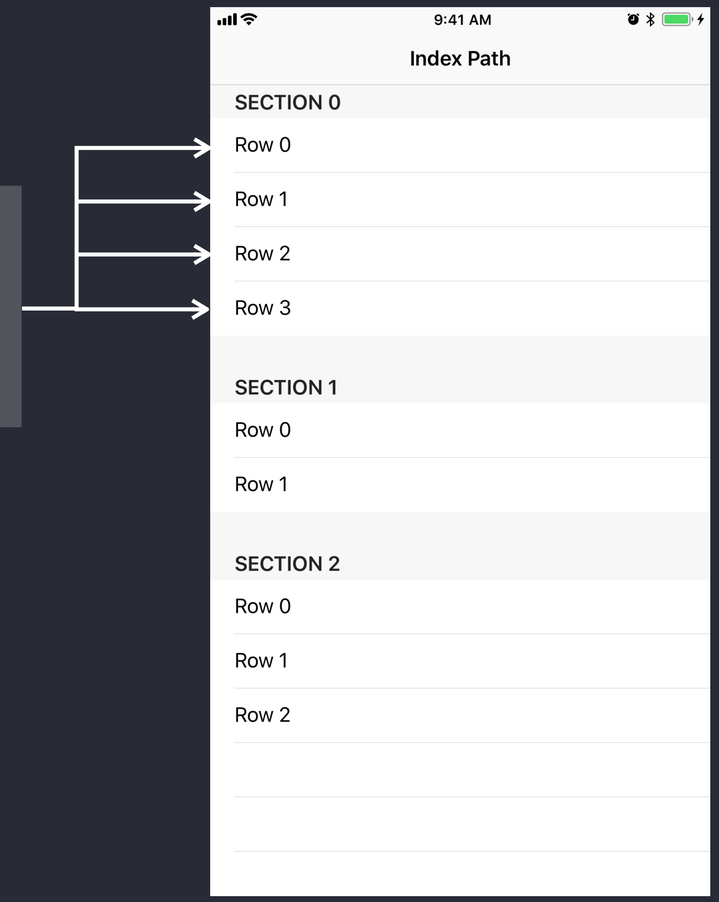
```
optional func numberOfSections(in
  tableView: UITableView) -> Int
```

→ If function isn’t provided, the table view assumes one section



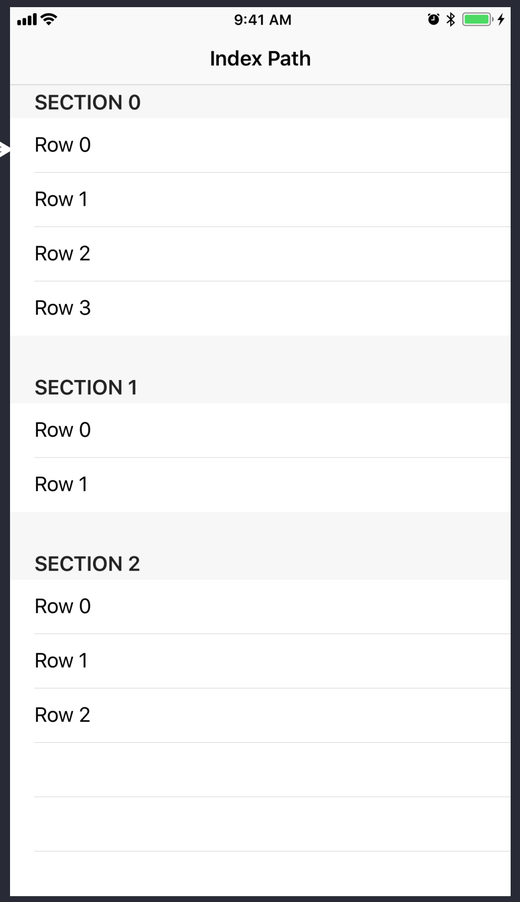
UITableViewDataSource

```
func tableView(_ tableView: UITableView,
  numberOfRowsInSection section: Int) -> Int
```



UITableViewDataSource

```
func tableView(_ tableView: UITableView,
  cellForRowAt indexPath: IndexPath) ->
  UITableViewCell
```



UITableViewDelegate

→ Responding to accessory view interaction

```
tableView(_:accessoryButtonTappedForRowWith:)
```

→ Responding to user interaction

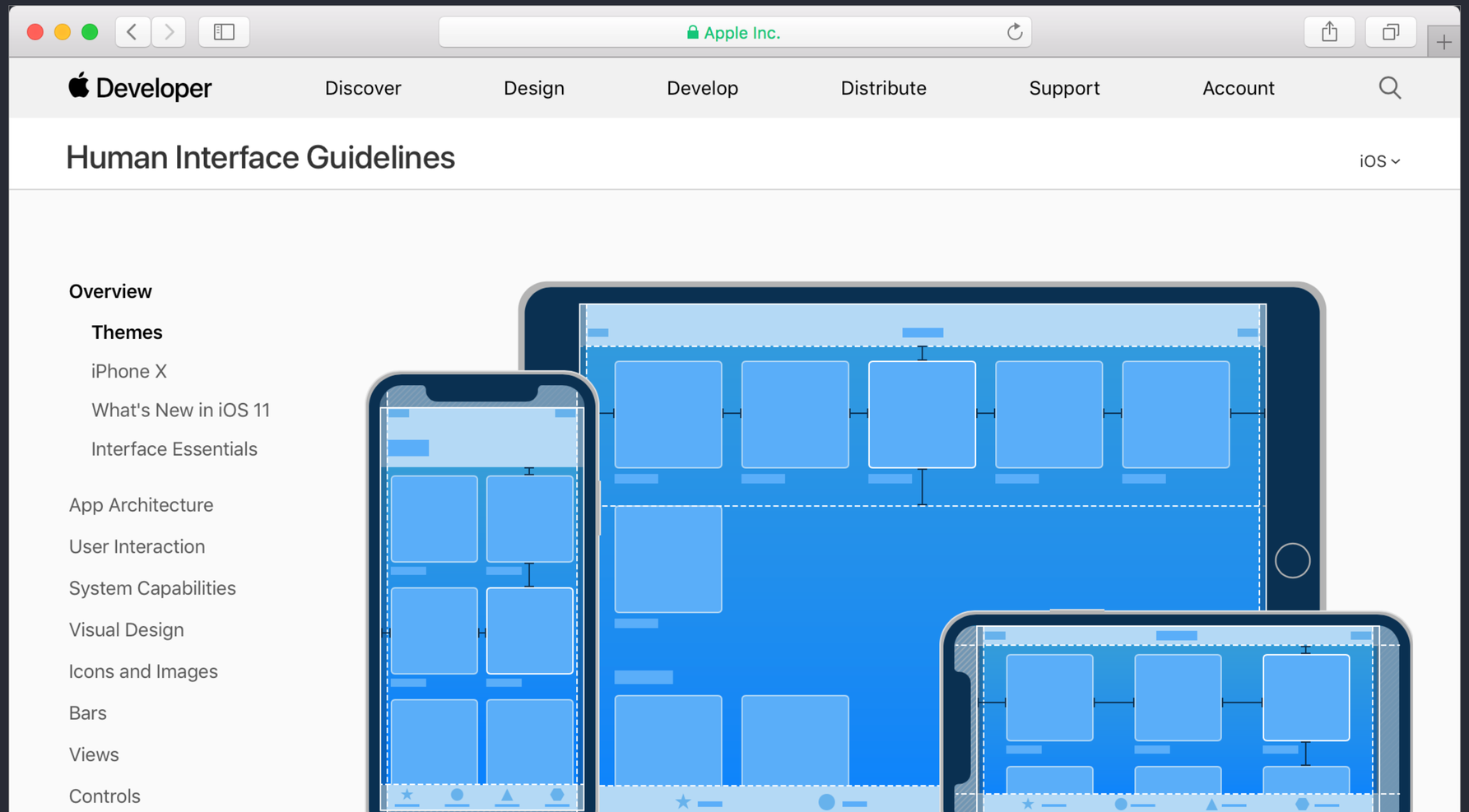
```
tableView(_:didSelectRowAt:)
```

Reload data

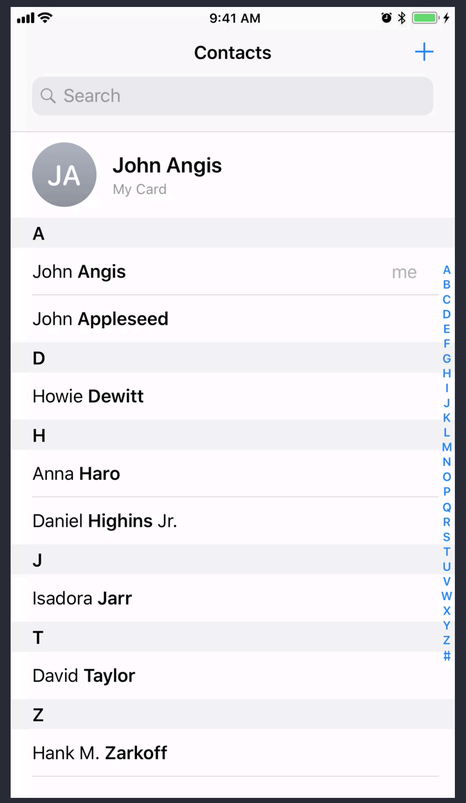
```
reloadData()
```

Building Simple Workflows

Human interface guidelines



Modal versus push

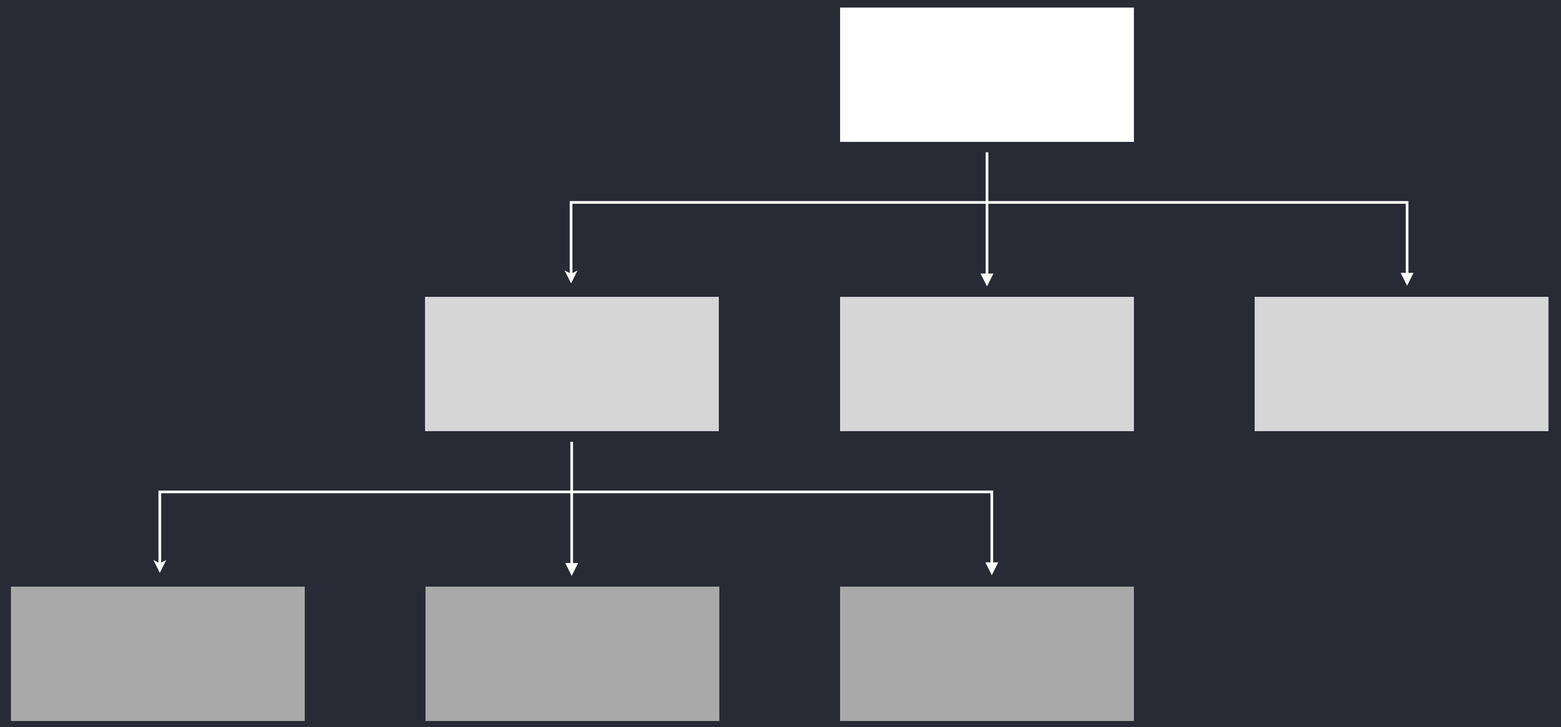


Navigation hierarchy

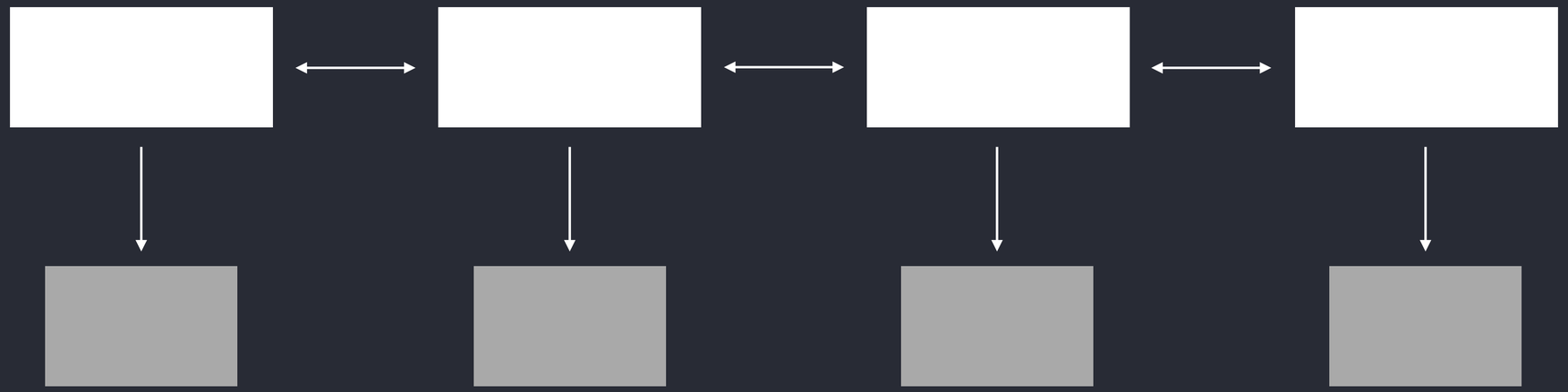
Three main types:

- → Hierarchical
- → Flat
- → Content-driven or experience-driven

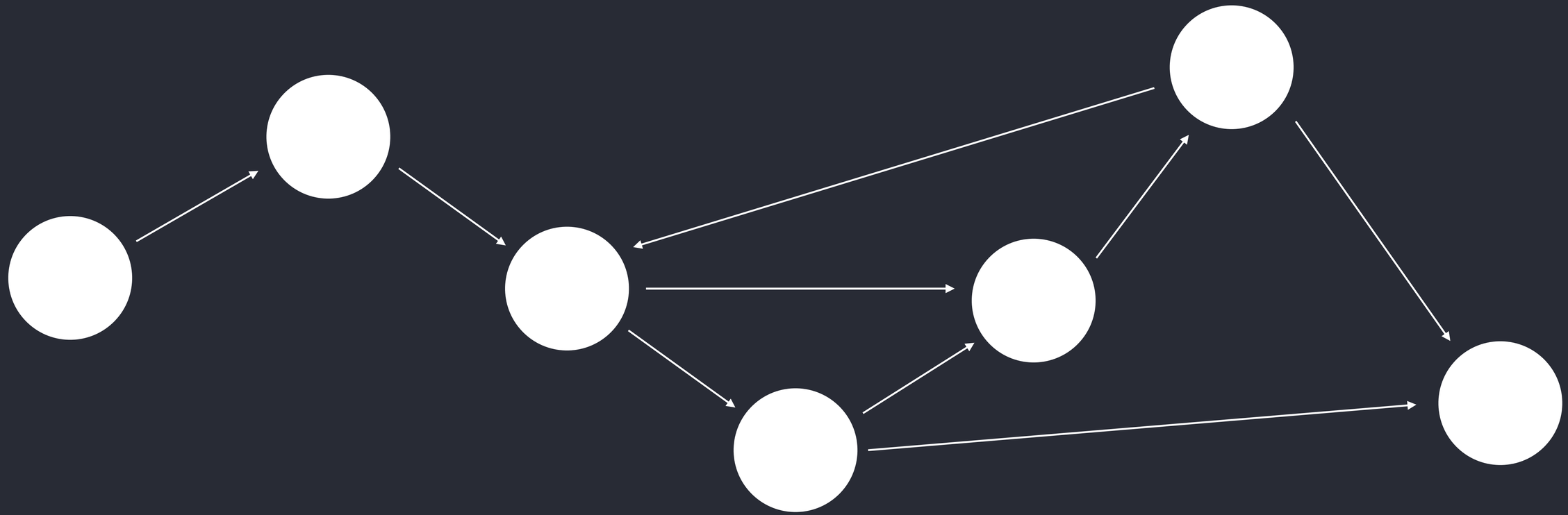
Hierarchical



Flat



Content-driven



Navigation design guidelines

Design an information structure that makes access to content fast and easy

Use standard navigation components

Use a navigation bar to traverse a hierarchy of data

Use a tab bar to present peer categories of content or functionality

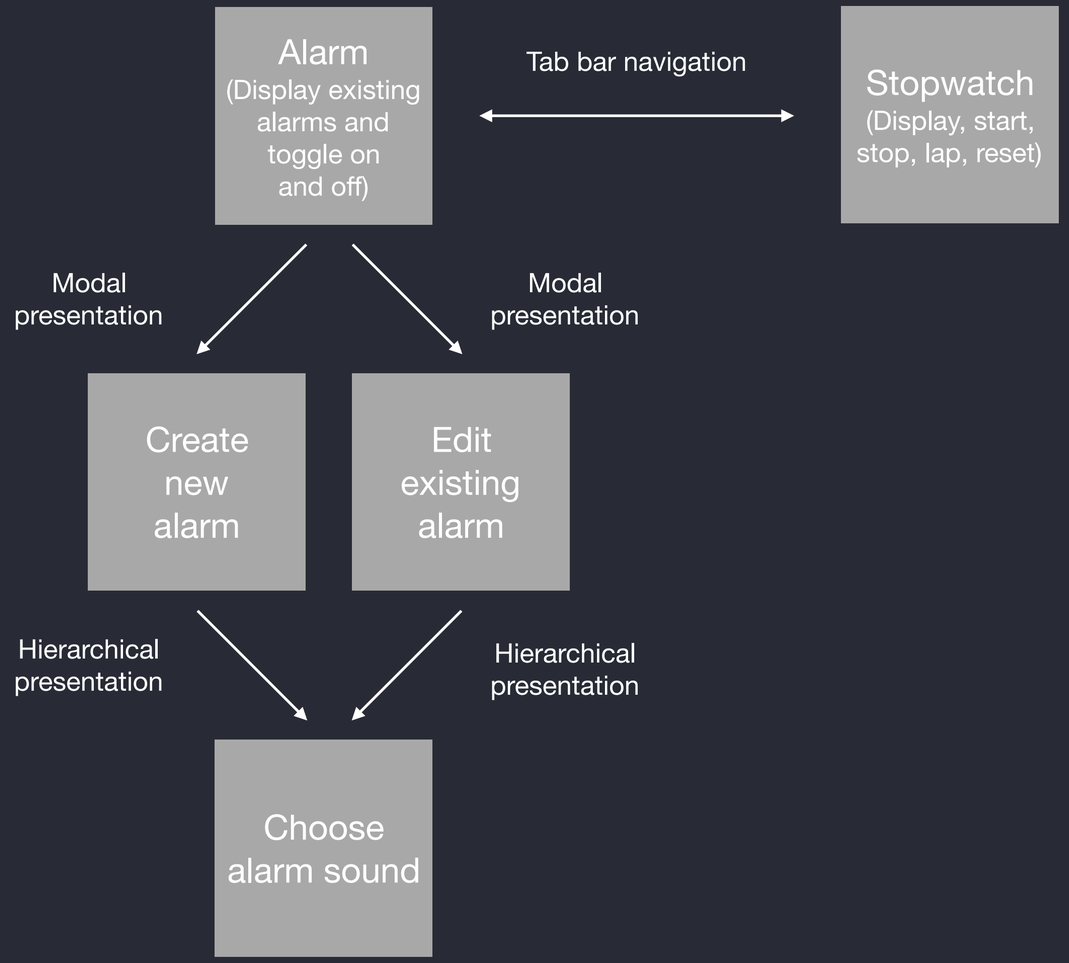
Example workflow

Alarm and stopwatch app

Features

- → Display alarms
- → Toggle alarms on and off
- → Create alarms
- → Change sound of alarms
- → Basic stopwatch functionality (display, start, lap, stop, reset)

Example workflow



Saving data

Encoding and decoding with Codable

```
class Note: Codable {…}
```

- → Use an Encoder object to encode
- → Use a Decoder object to decode

Encoding and decoding with Codable

```
struct Note: Codable {
    let title: String
    let text: String
    let timestamp: Date
}

let newNote = Note(title: "Dry cleaning", text: "Pick up suit
                   from dry cleaners", timestamp: Date())

let propertyListEncoder = PropertyListEncoder()
if let encodedNotes = try? propertyListEncoder.encode(newNote) {
   …
}
```

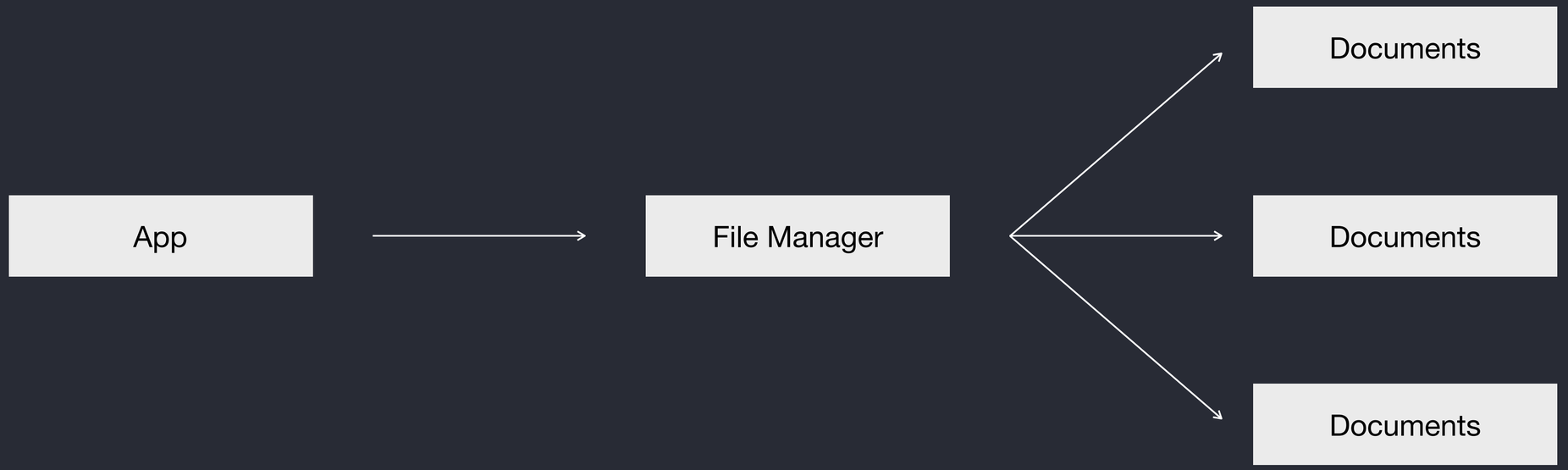
Encoding and decoding with Codable

```
let propertyListDecoder = PropertyListDecoder()
if let decodedNote = try? propertyListDecoder.decode(Note.self,
from: encodedNote) {
    …
}
```

App Sandbox

App Sandbox

Writing data to a file



Encoding and decoding with Codable

```
let documentsDirectory =
FileManager.default.urls(for: .documentDirectory,
                         in: .userDomainMask).first!
let archiveURL =
documentsDirectory.appendingPathComponent("appData")
                  .appendingPathExtension("plist")
```

Encoding and decoding with Codable

```
let propertyListEncoder = PropertyListEncoder()
let encodedData = try? propertyListEncoder.encode(data)

try? encodedData?.write(to: archiveURL,
                        options: .noFileProtection)
```

Encoding and decoding with Codable

```
let propertyListDecoder = PropertyListDecoder()
if let retrievedData = try? Data(contentsOf: archiveURL),
    let decodedNote = try? propertyListDecoder.decode(Note.self,
from: retrievedNoteData) {
  …
}
```

Your model objects should implement the Codable protocol.
Reading and writing should happen in the model controller.

Archive in the correct app delegate life-cycle events. For example:

→ When the app enters the background

→ When the app is terminated

adhumi.fr/teaching

Credentials
m2sar / sarM2