Introduction to iOS development with Swift

Lesson 4

Adrien Humilière
Trainline

adhumi+dant@gmail.com

- Optionals
- Type casting
- Guard
- Scope
- Segues and Navigation Controllers
- Tab Bar Controllers
- View Controller Life Cycle
- Simple Workflows

Optionals

nil

```
struct Book {
  let name: String
  let publicationYear: Int
}

let firstHarryPotter = Book(name: "Harry Potter and the
Sorcerer's Stone", publicationYear: 1997)
let secondHarryPotter = Book(name: "Harry Potter and the
Chamber of Secrets", publicationYear: 1998)

let books = [firstHarryPotter, secondHarryPotter]
```

Arrays

```
let unannouncedBook = Book(name: "Harry Potter 8",
                           publicationYear: ???)
```

Arrays

```
let unannouncedBook = Book(name: "Harry Potter 8",
                           publicationYear: 0)
```

Arrays

```
let unannouncedBook = Book(name: "Harry Potter 8",
                           publicationYear: 2018)
```

Arrays

```
let unannouncedBook = Book(name: "Harry Potter 8",
                           publicationYear: nil)
```

Nil is not compatible with expected argument type ‘Int’

```
struct Book {
  let name: String
  let publicationYear: Int?
}

let firstHarryPotter = Book(name: "Harry Potter and the
Sorcerer's Stone", publicationYear: 1997)
let secondHarryPotter = Book(name: "Harry Potter and the
Chamber of Secrets", publicationYear: 1998)

let books = [firstHarryPotter, secondHarryPotter]

let unannouncedBook = Book(name: "Rebels and Lions",
publicationYear: nil)
```

Specifying the type of an optional

```
var serverResponseCode: Int = 404
```

```
var serverResponseCode: Int = nil
```

‘nil’ requires a contextual type

```
var serverResponseCode: Int? = 404
```

```
var serverResponseCode: Int? = nil
```

Working with optional values

```
if publicationYear != nil {
  let actualYear = publicationYear!
  print(actualYear)
}

let unwrappedYear = publicationYear!
```

error: Execution was interrupted

Working with optional values

```
if let constantName = someOptional {
  //constantName has been safely unwrapped for use within {}
}
```

```
if let unwrappedPublicationYear = book.publicationYear {
  print("The book was published in \(unwrappedPublicationYear)")
} else {
  print("The book does not have an official publication date.")
}
```

Functions and optionals

```
let string = "123"
let possibleNumber = Int(string)
```

```
let string = "Cynthia"
let possibleNumber = Int(string)
```

Functions and optionals

```
func printFullName(firstName: String, middleName: String?,
lastName: String)
```

```
func textFromURL(url: URL) -> String?
```

Failable initializers

```
struct Toddler {
  var birthName: String
  var monthsOld: Int
}
```

Failable initializers

```
init?(birthName: String, monthsOld: Int) {
  if monthsOld < 12 || monthsOld > 36 {
    return nil
  } else {
    self.birthName = birthName
    self.monthsOld = monthsOld
  }
}
```

Failable initializers

```
let possibleToddler = Toddler(birthName: "Joanna", monthsOld: 14)
if let toddler = possibleToddler {
  print("\(toddler.birthName) is \(toddler.monthsOld) months old")
} else {
  print("The age you specified for the toddler is not between 1
and 3 yrs of age")
}
```

Optional chaining

```
class Person {
  var age: Int
  var residence: Residence?
}

class Residence {
  var address: Address?
}
```

```
class Address {
  var buildingNumber: String?
  var streetName: String?
  var apartmentNumber: String?
}
```

Optional chaining

```
if let theResidence = person.residence {
  if let theAddress = theResidence.address {
    if let theApartmentNumber = theAddress.apartmentNumber {
      print("He/she lives in apartment number
            \(theApartmentNumber). »)
    }
  }
}
```

Implicitly Unwrapped Optionals

```
class ViewController: UIViewController {
  @IBOutlet weak var label: UILabel!
}
```

Unwraps automatically

Should only be used when need to initialize an object without supplying the value and you'll be giving the object a value soon afterwards

Type Casting and Inspection

Optional chaining

```
func getClientPet() -> Animal {
  //returns the pet
}

let pet = getClientPet() //`pet` is of type `Animal`
```

Optional chaining

```
if pet is Dog {
  print("The client's pet is a dog")
} else if pet is Cat {
  print("The client's pet is a cat")
} else if pet is Bird {
  print("The client's pet is a bird")
} else {
  print("The client has a very exotic pet")
}
```

Optional chaining

```
let pets = allPets() //`pets` is of type `[Animal]`
var dogCount = 0, catCount = 0, birdCount = 0
for pet in pets {
  if pet is Dog {
    dogCount += 1
  } else if pet is Cat {
    catCount += 1
  } else if pet is Bird {
    birdCount += 1
  }
}
print("Brad looks after \(dogCount) dogs, \(catCount) cats,
and \(birdCount) birds.")
```

Type casting

```
func walk(dog: Dog) {
  print("Walking \(dog.name)")
}

func cleanLitterBox(cat: Cat) {. . .}

func cleanCage(bird: Bird) {. . .}

for pet in pets {
  if pet is Dog {
    walk(dog: pet) // Compiler error
  }
  …
}
```

Type casting

```
for pet in pets {
  if let dog = pet as? Dog {
    walk(dog: dog)
  } else if let cat = pet as? Cat {
    cleanLitterBox(cat: cat)
  } else if let bird = pet as? Bird {
    cleanCage(bird: bird)
  }
}
```

Any

```
var items: [Any] = [5, "Bill", 6.7, Dog()]
```

Any

```
var items: [Any] = [5, "Bill", 6.7, Dog()]
let firstItem = items[0]

if firstItem is Int {
  print("The first element is an integer")
} else if firstItem is String {
  print("The first element is a string")
} else {
  print("The first element is neither an integer nor a string")
}
```

Any

```
var items: [Any] = [5, "Bill", 6.7, Dog()]

if let firstItem = items[0] as? Int {
  print(firstItem + 4)
}
```

Guard

```
func singHappyBirthday() {
  if birthdayIsToday {
    if invitedGuests > 0 {
      if cakeCandlesLit {
        print("Happy Birthday to you!")
      } else {
        print("The cake candle's haven't been lit.")
      }
    } else {
      print("It's just a family party.")
    }
  } else {
    print("No one has a birthday today.")
  }
}
```

```
func singHappyBirthday() {
  guard birthdayIsToday else {
    print("No one has a birthday today.")
    return
  }
  guard invitedGuests > 0 else {
    print("It's just a family party.")
    return
  }
  guard cakeCandlesLit else {
    print("The cake's candles haven't been lit.")
    return
  }
  print("Happy Birthday to you!")
}
```

guard

```
guard condition else {
  //false: execute some code
}

//true: execute some code
```

guard

```
func divide(_ number: Double, by divisor: Double) {
  if divisor != 0.0 {
    let result = number / divisor
    print(result)
  }
}
```

```
func divide(_ number: Double, by divisor: Double) {
  guard divisor != 0.0 else { return }

  let result = number / divisor
  print(result)
}
```

```
func processBook(title: String?, price: Double?, pages: Int?) {
  if let theTitle = title, let thePrice = price, let thePages =
pages {
    print("\(theTitle) costs $\(thePrice) and has \(thePages)
pages.")
  }
}
```

```
func processBook(title: String?, price: Double?, pages: Int?){
  guard let theTitle = title, let thePrice = price, let
thePages = pages else { return }
  print("\(theTitle) costs $\(thePrice) and has \(thePages)
pages.")
}
```

Constant and Variable Scope

Scope

Global scope — Defined outside of a function
Local scope — Defined within braces ({ })

```
var globalVariable = true

if globalVariable {
  let localVariable = 7
}
```

Scope

```
var age = 55

func printMyAge() {
  print("My age: \(age)")
}

print(age)
printMyAge()
```

Scope

```
func printBottleCount() {
    let bottleCount = 99
    print(bottleCount)
}

printBottleCount()
print(bottleCount)
```

Scope

```
func printTenNames() {
  var name = "Richard"
  for index in 1...10 {
    print("\(index): \(name)")
  }
  print(index)
  print(name)
}

printTenNames()
```

Variable shadowing

```
let points = 100

for index in 1...3 {
  let points = 200
  print("Loop \(index): \(points+index)")
}
print(points)
```

Variable shadowing

```
var name: String? = "Robert"

if let name = name {
 print("My name is \(name)")
}
```

Variable shadowing

```
func exclaim(name: String?) {
  if let name = name {
    print("Exclaim function was passed: \(name)")
  }
}
```

```
func exclaim(name: String?) {
  guard let name = name else { return }
  print("Exclaim function was passed: \(name)")
}
```

Shadowing and initializers

```
struct Person {
  var name: String
  var age: Int
}

let todd = Person(name: "Todd", age: 50)
print(todd.name)
print(todd.age)
```

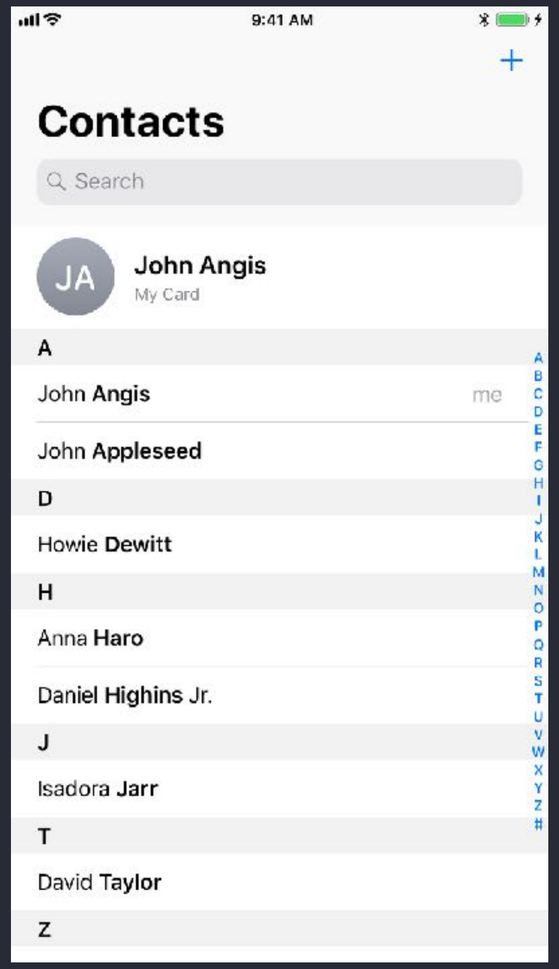
Shadowing and initializers

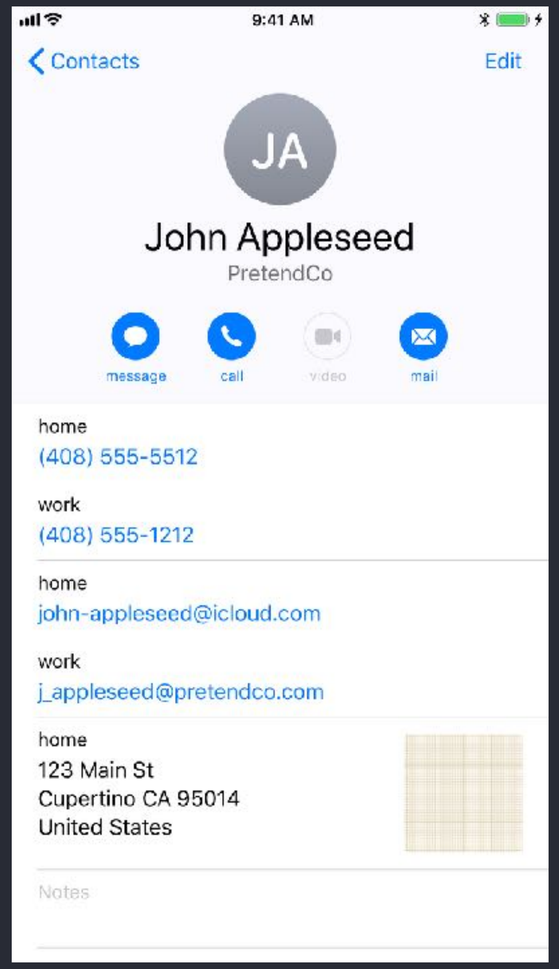
```
struct Person {
  var name: String
  var age: Int

  init(name: String, age: Int) {
    self.name = name
    self.age = age
  }
}
```

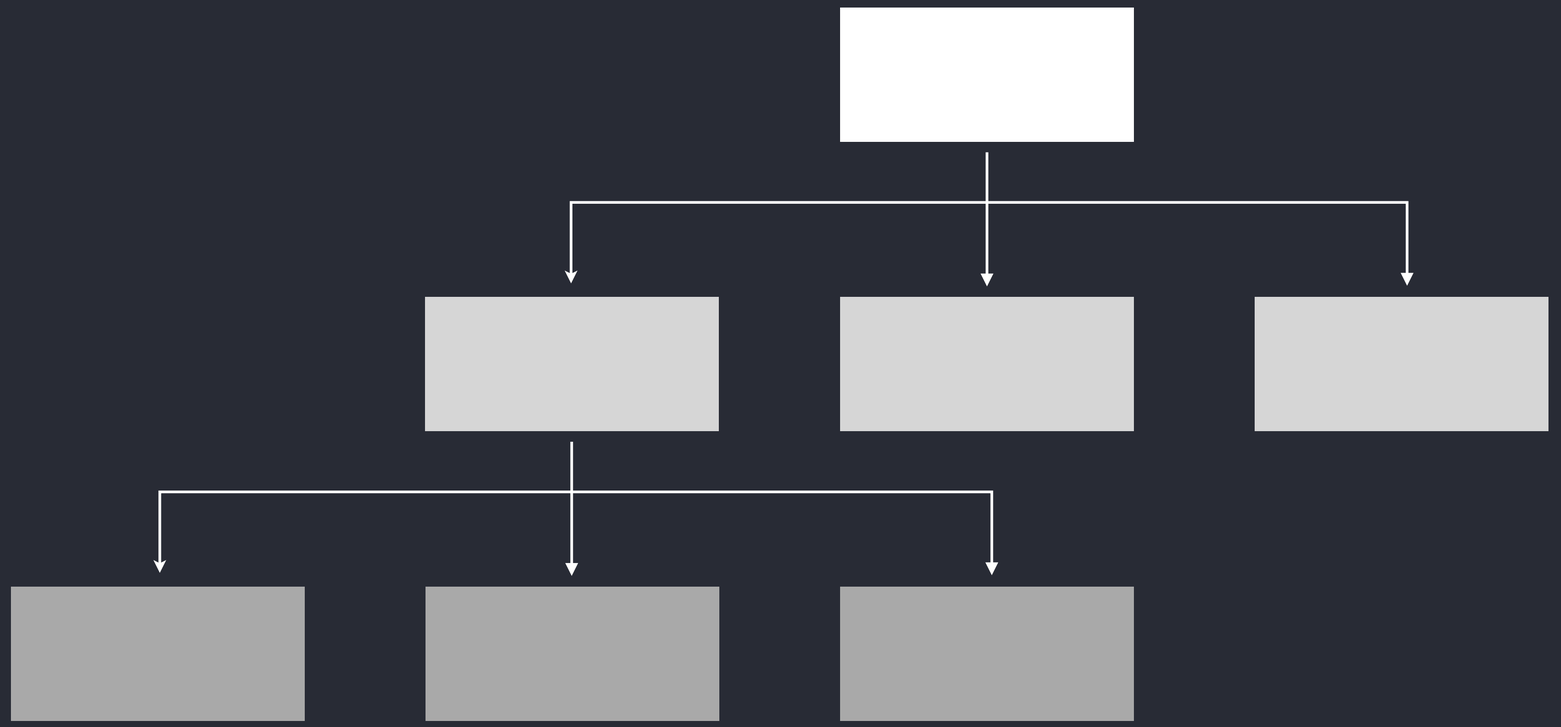
Segues and Navigation Controllers

Segues and navigation controllers





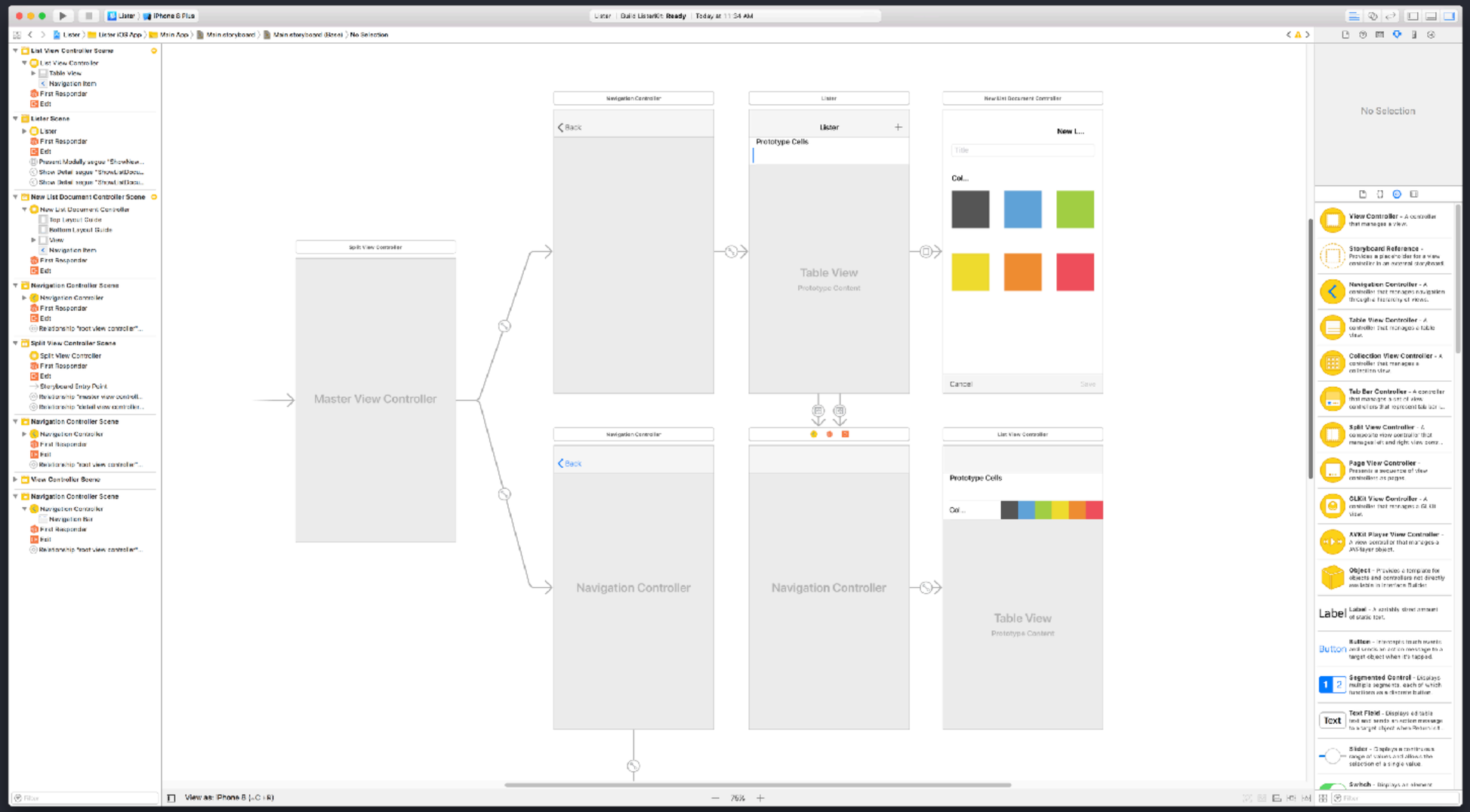
Navigation hierarchy



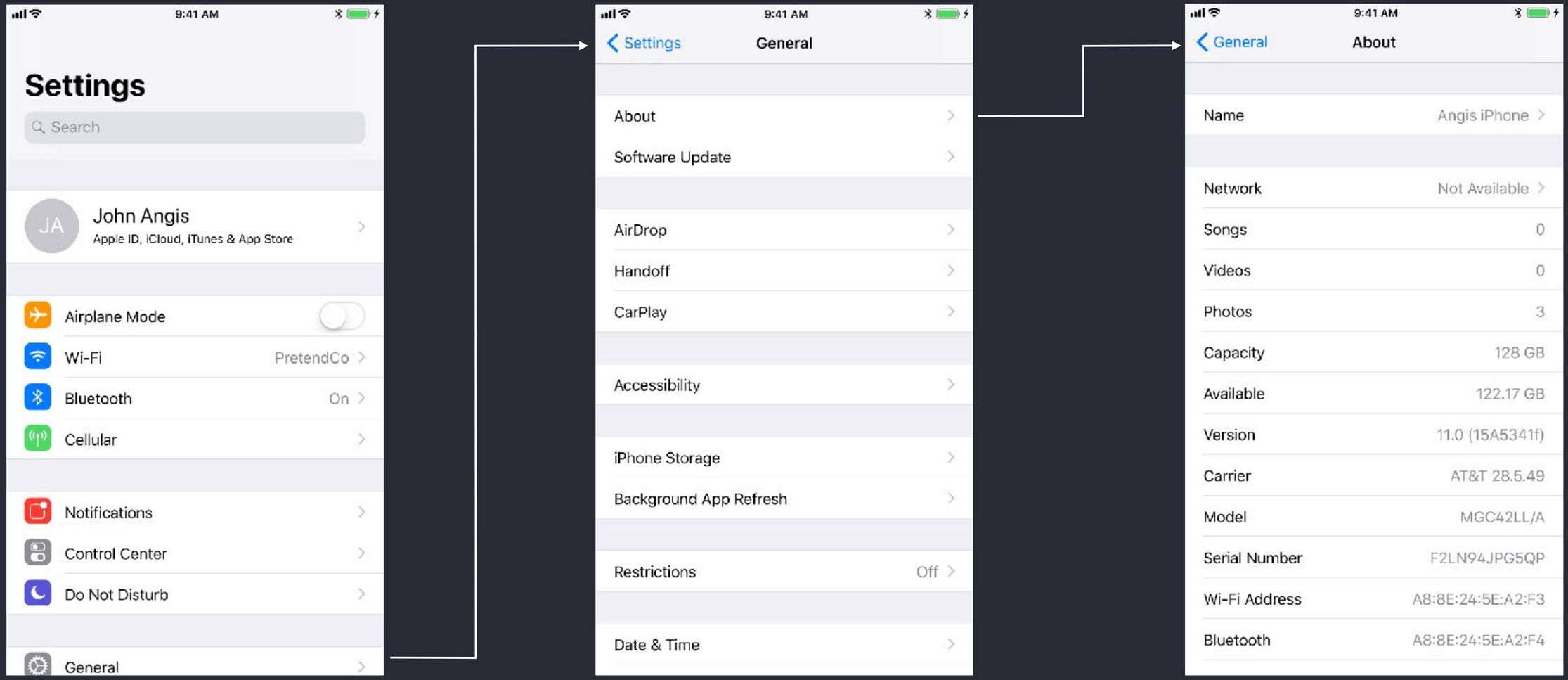
Segues (UIStoryboardSegue)

- 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

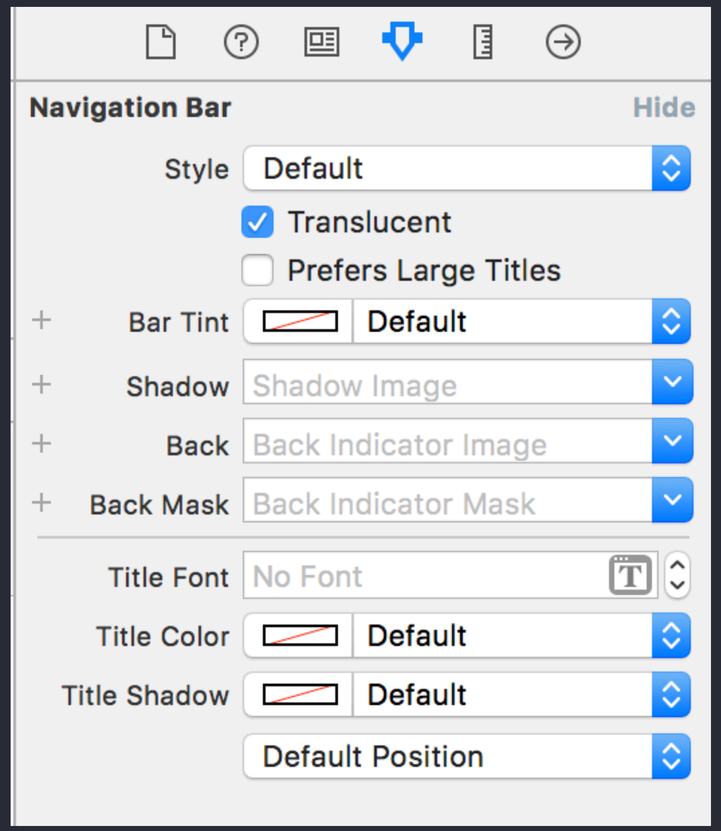
The top view controller's title

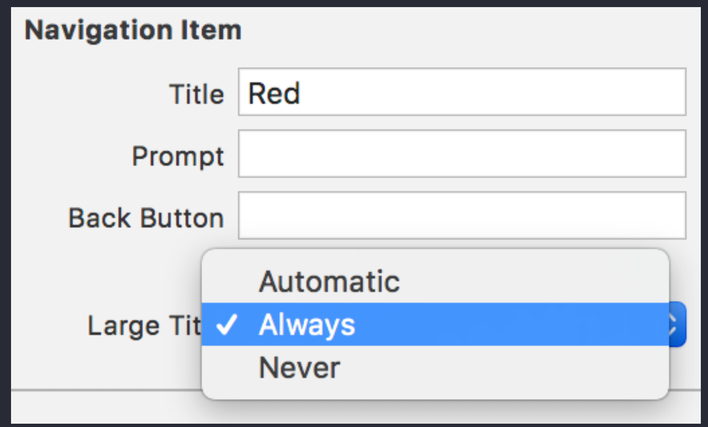
Back button

Navigation bar

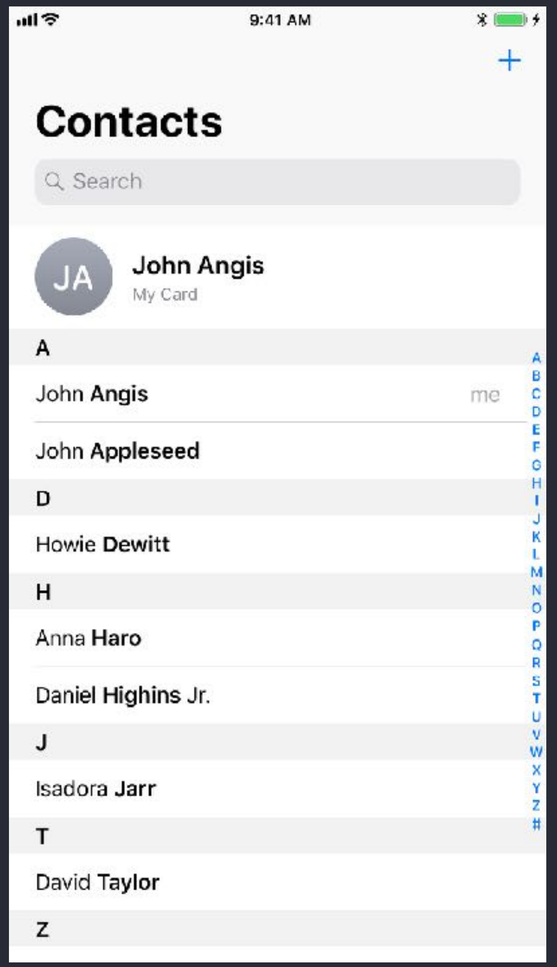
The top view controller's view

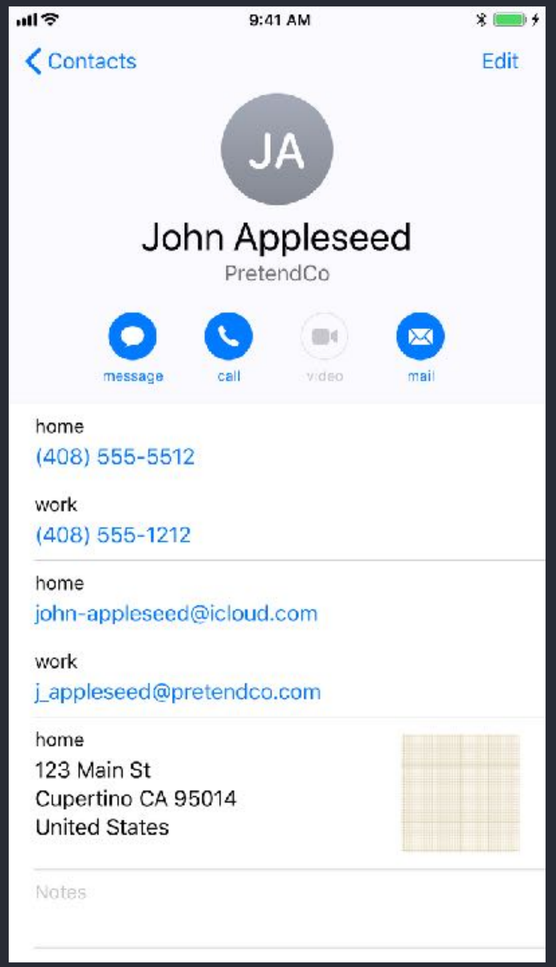
Navigation controller





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
}
```

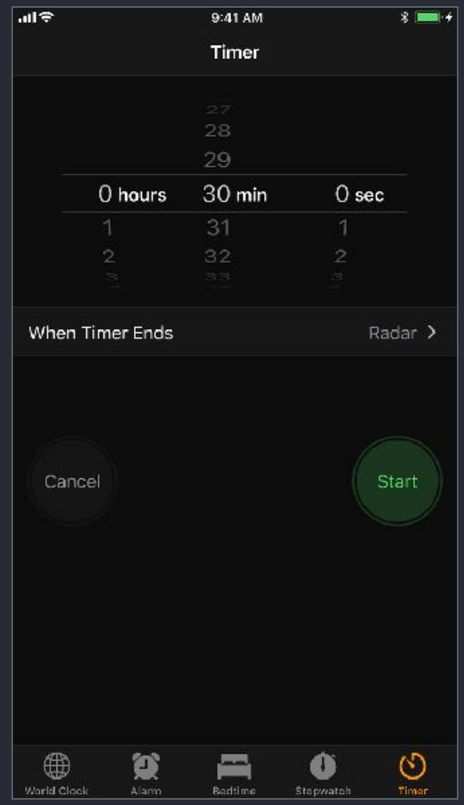
Programmatic segue

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

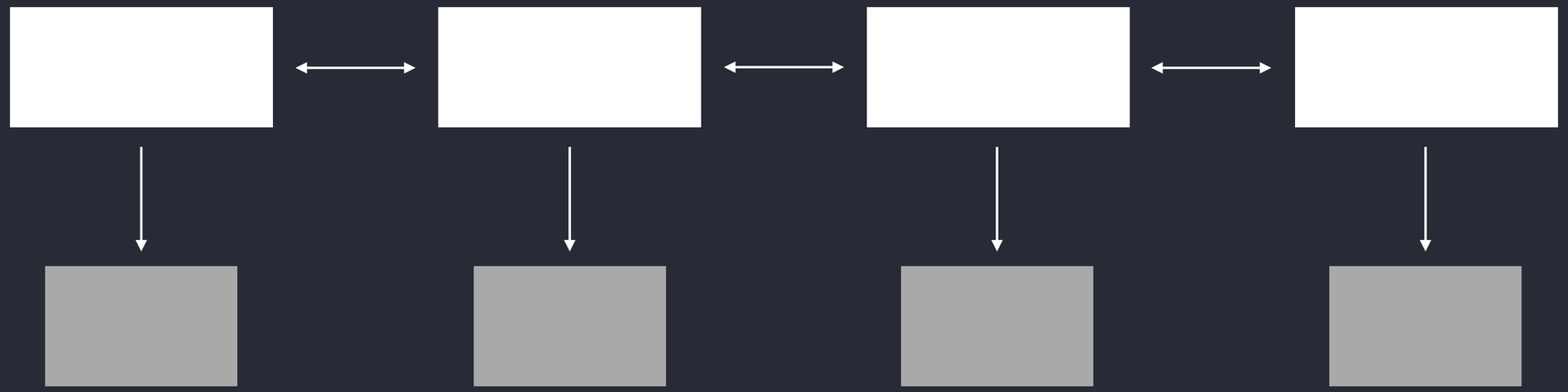
Tab Bar Controllers

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

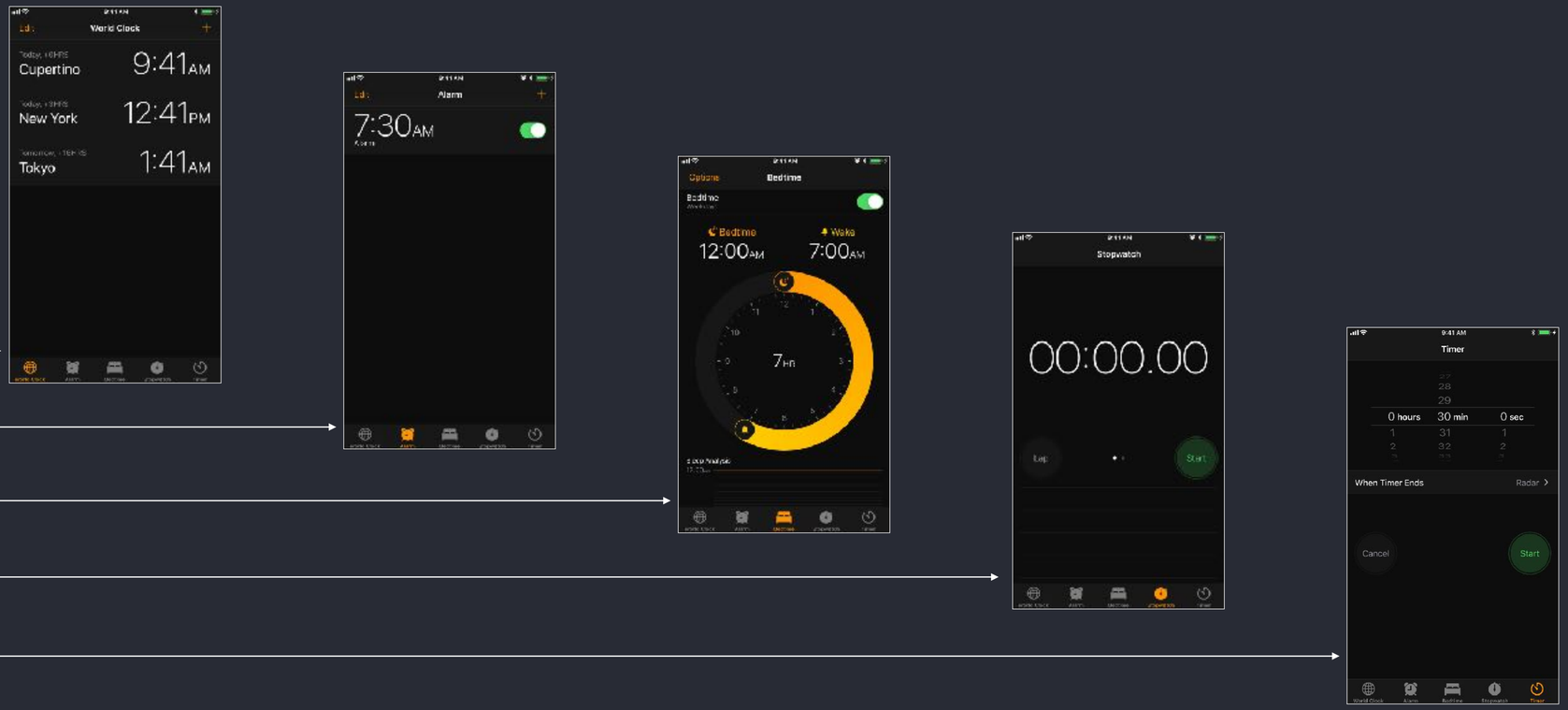


UITabBarController

Tab Bar Controller

`viewControllers`

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



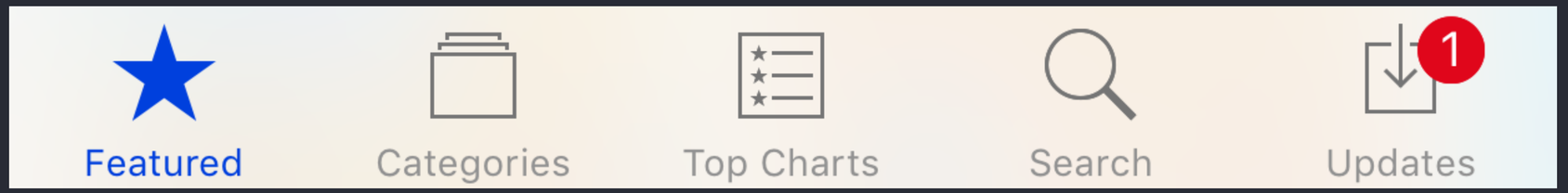
Add a tab bar controller

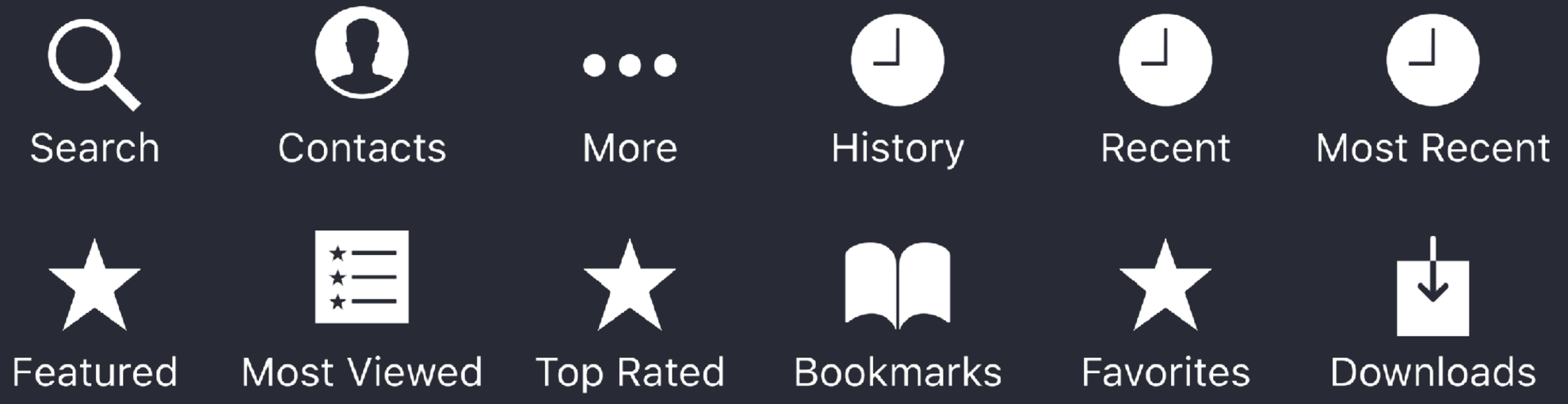
- Using a storyboard
- Drag in a UITabBarController from the object library

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





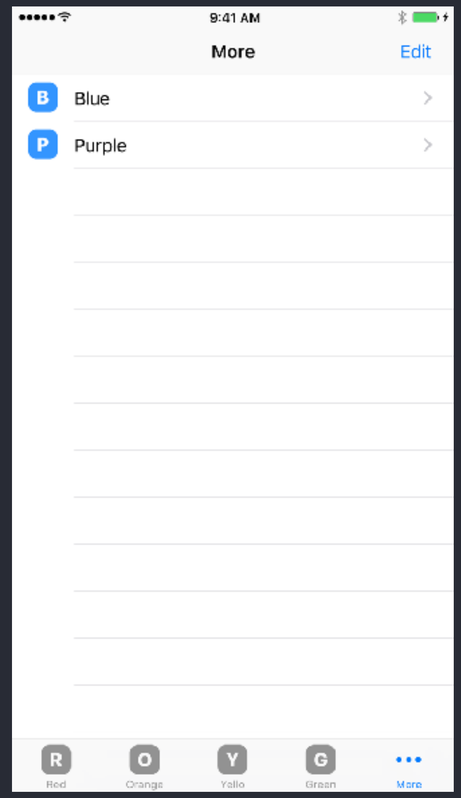
Programmatic customization

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


```
tabBarItem.badgeValue = nil
```

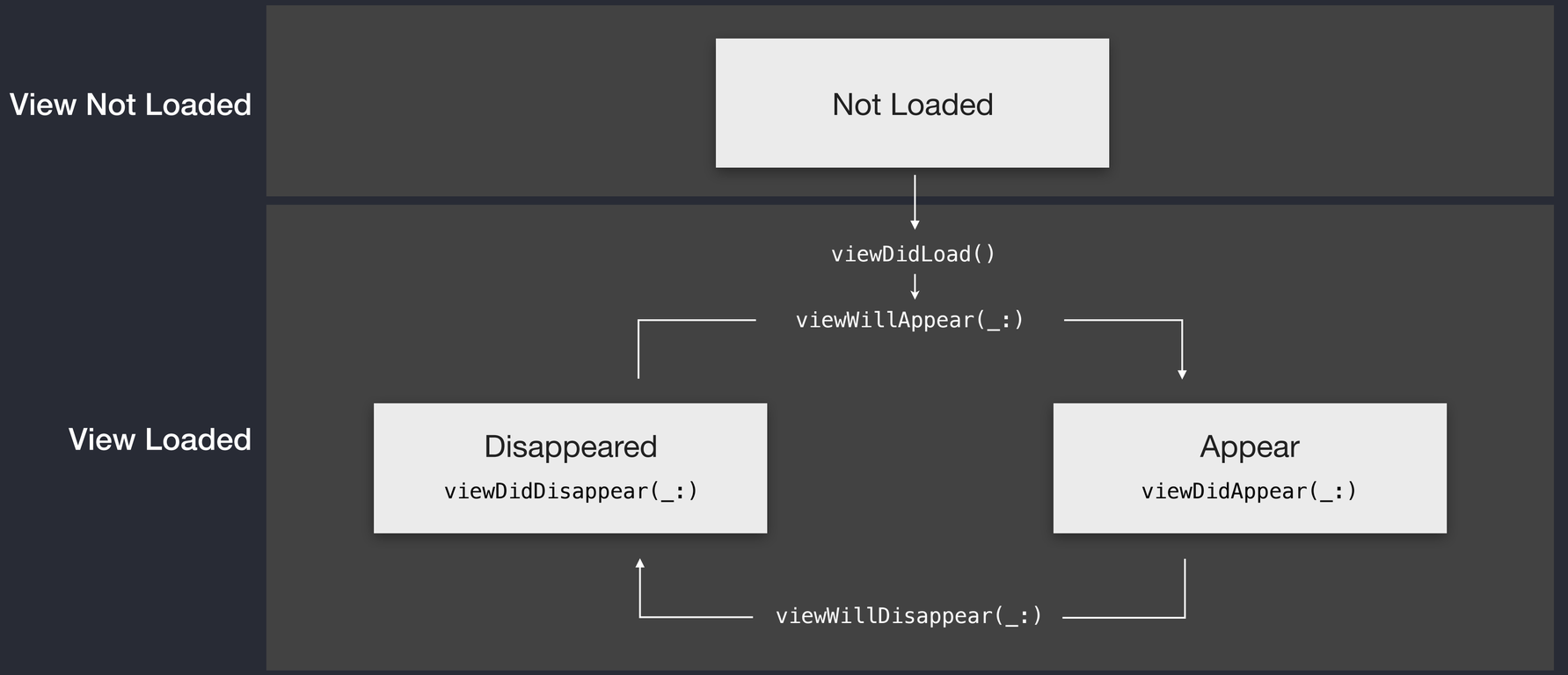
Even more tab items

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

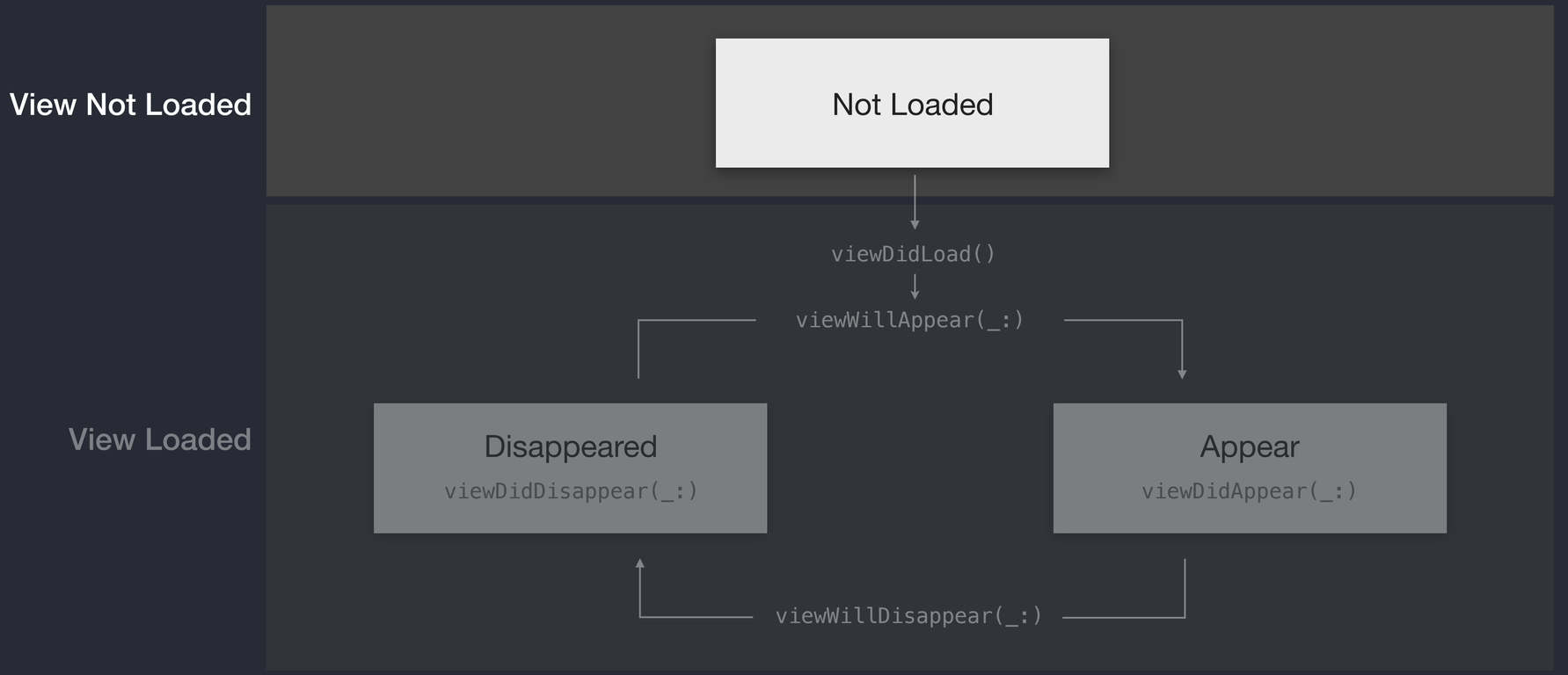


View Controller Life Cycle

View controller life cycle



View controller life cycle

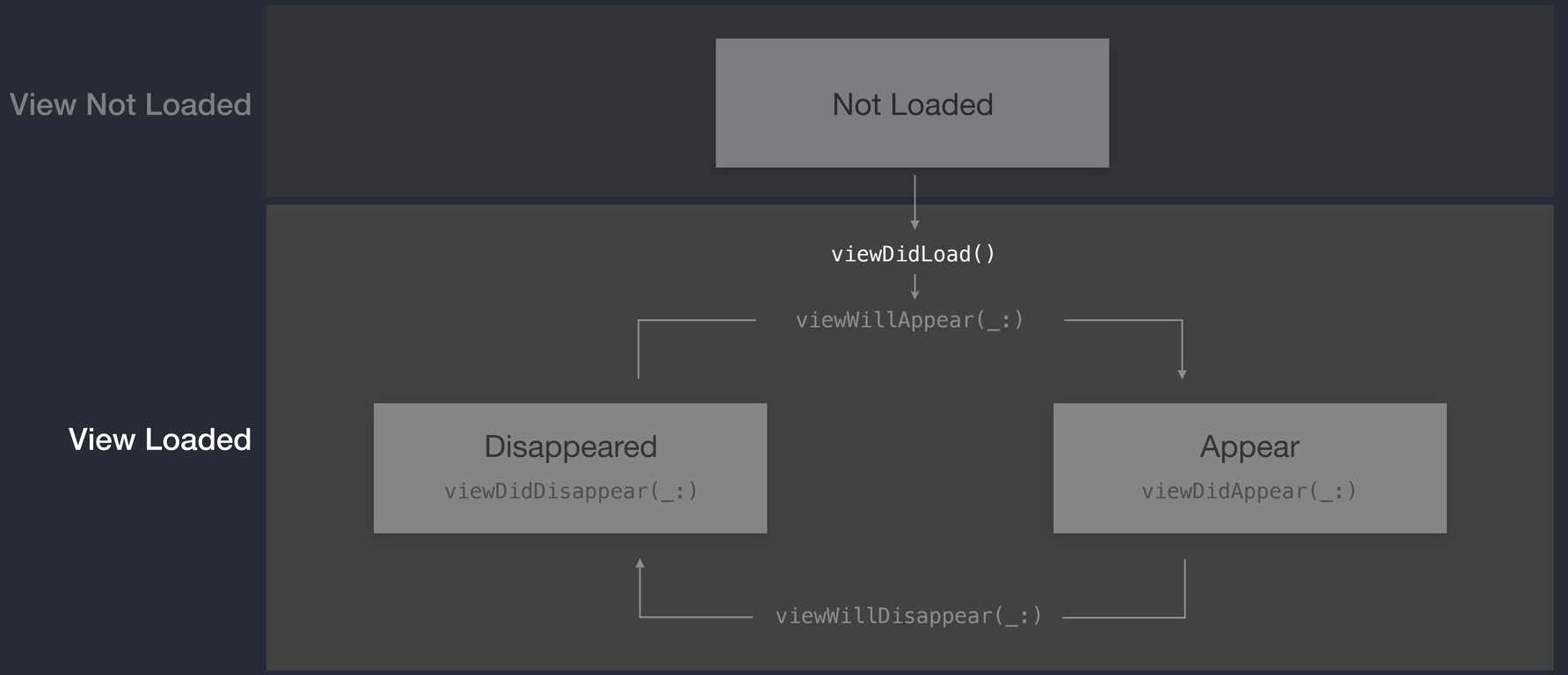


View event management

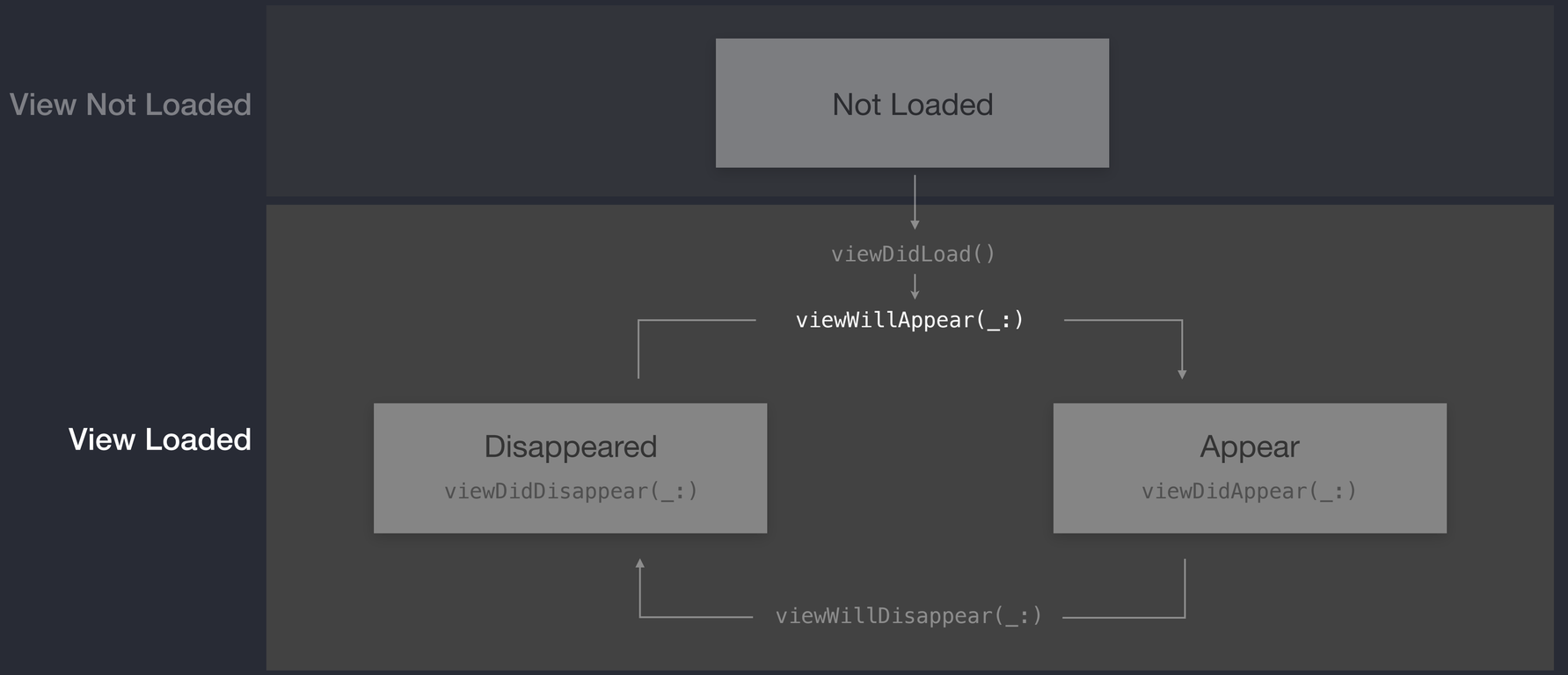
```
viewWillAppear(_:)
viewDidAppear(_:)
viewWillDisappear(_:)
viewDidDisappear(_:)
```

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

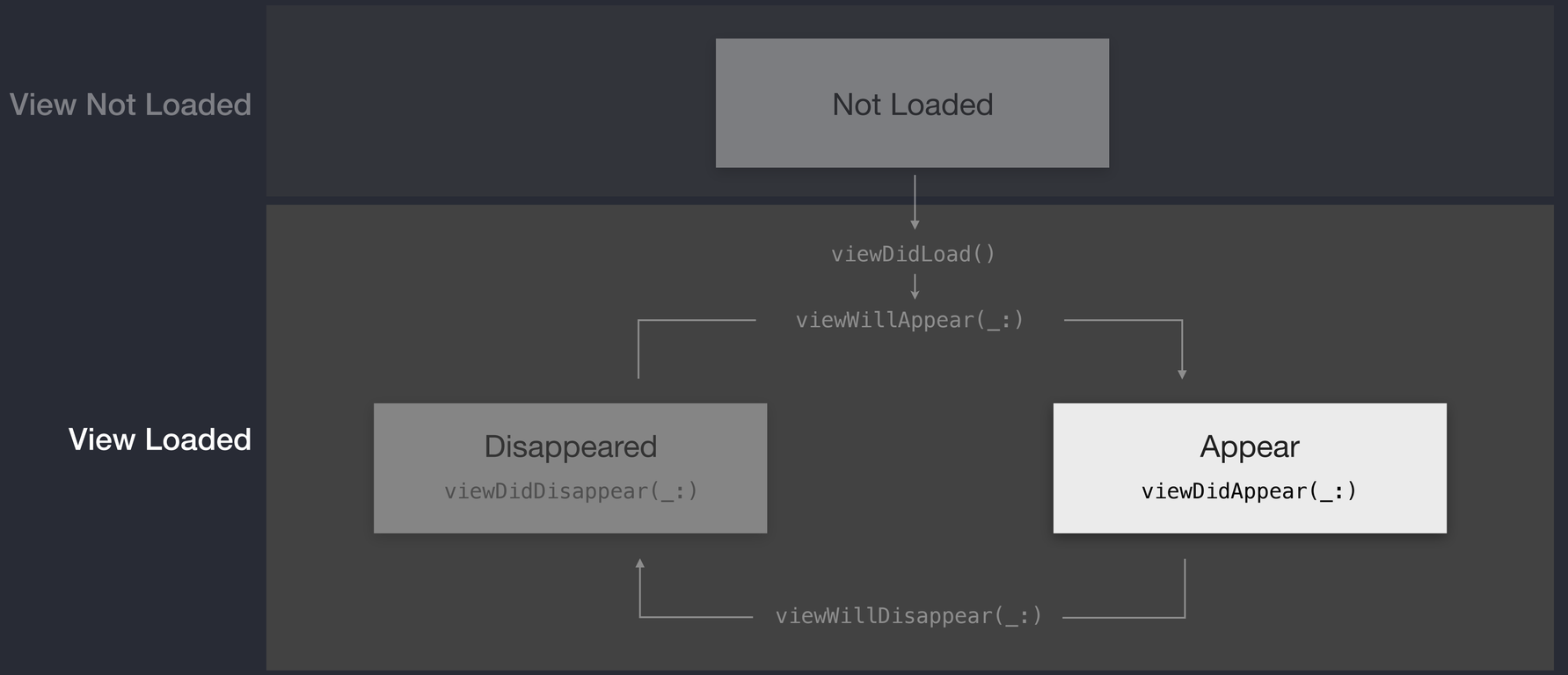
View event management



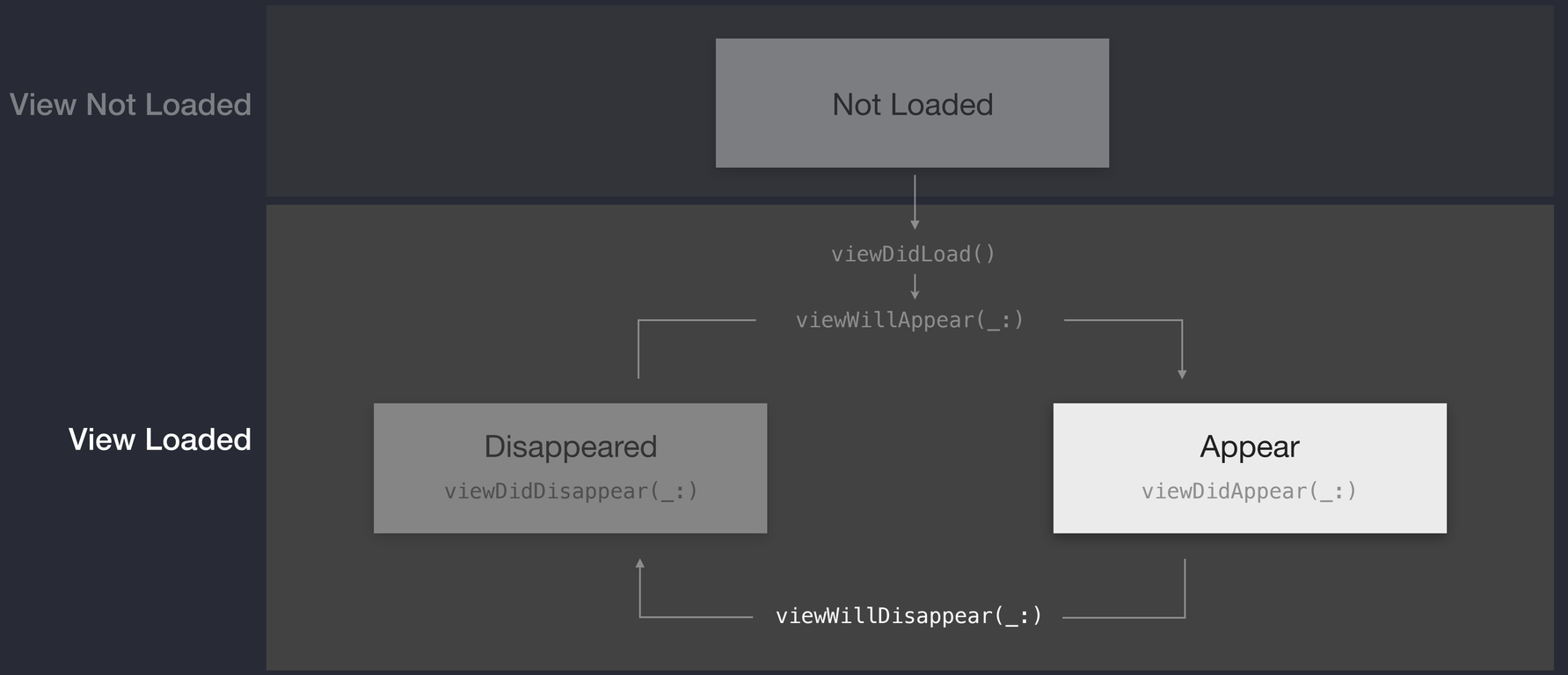
View event management



View event management

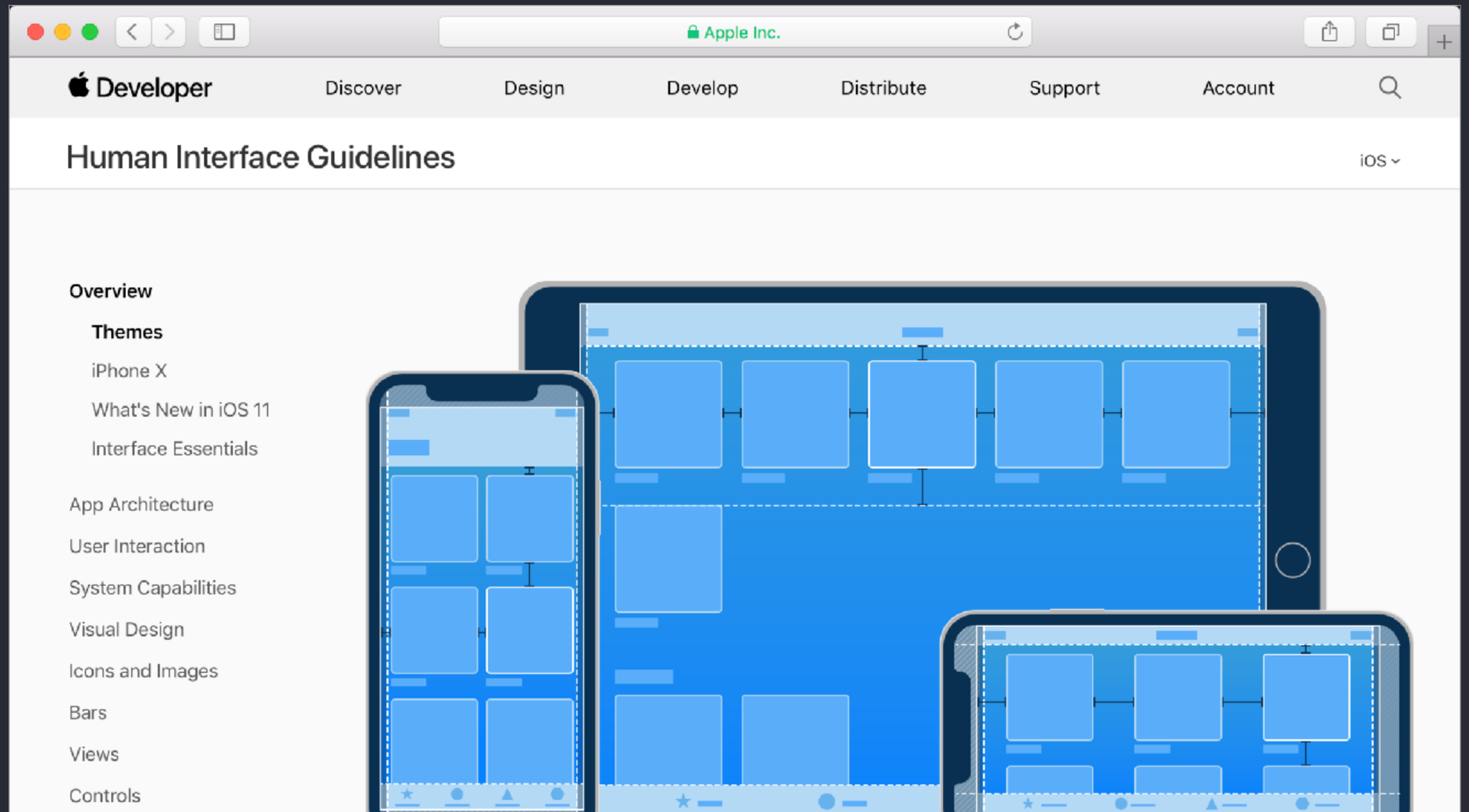


View event management

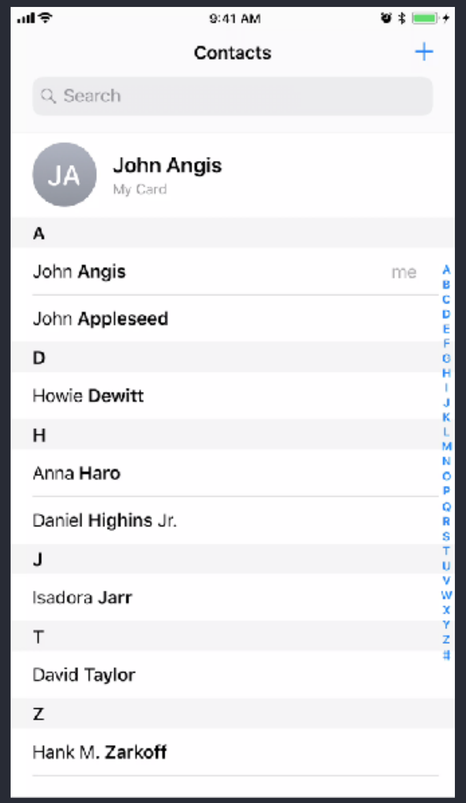


Building Simple Workflows

Human interface guidelines



Modal versus push

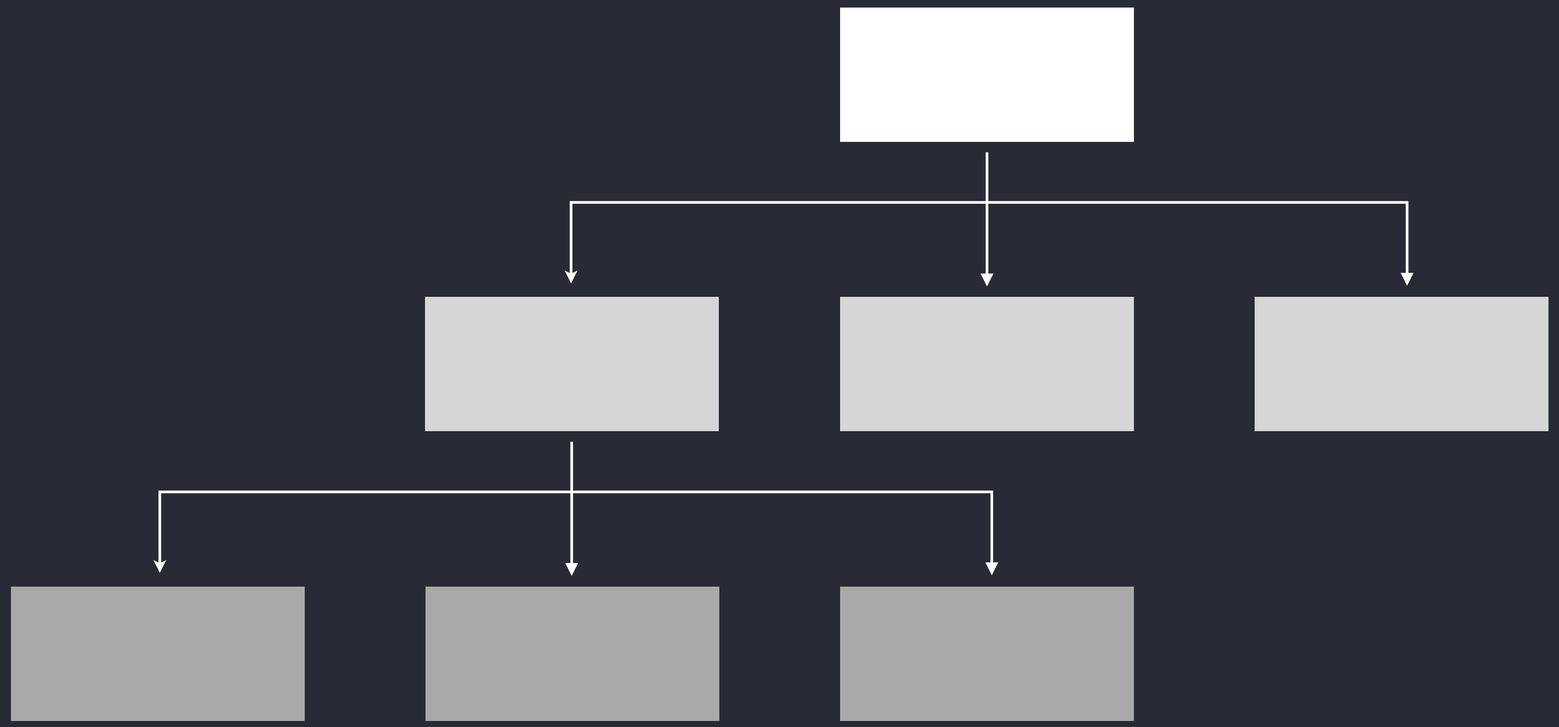


Navigation hierarchy

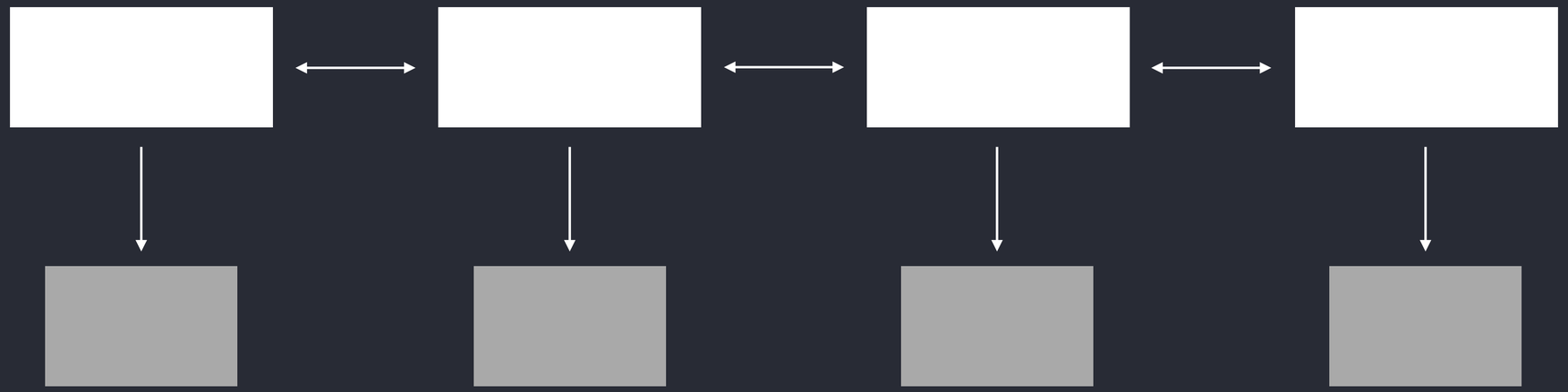
Three main types:

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

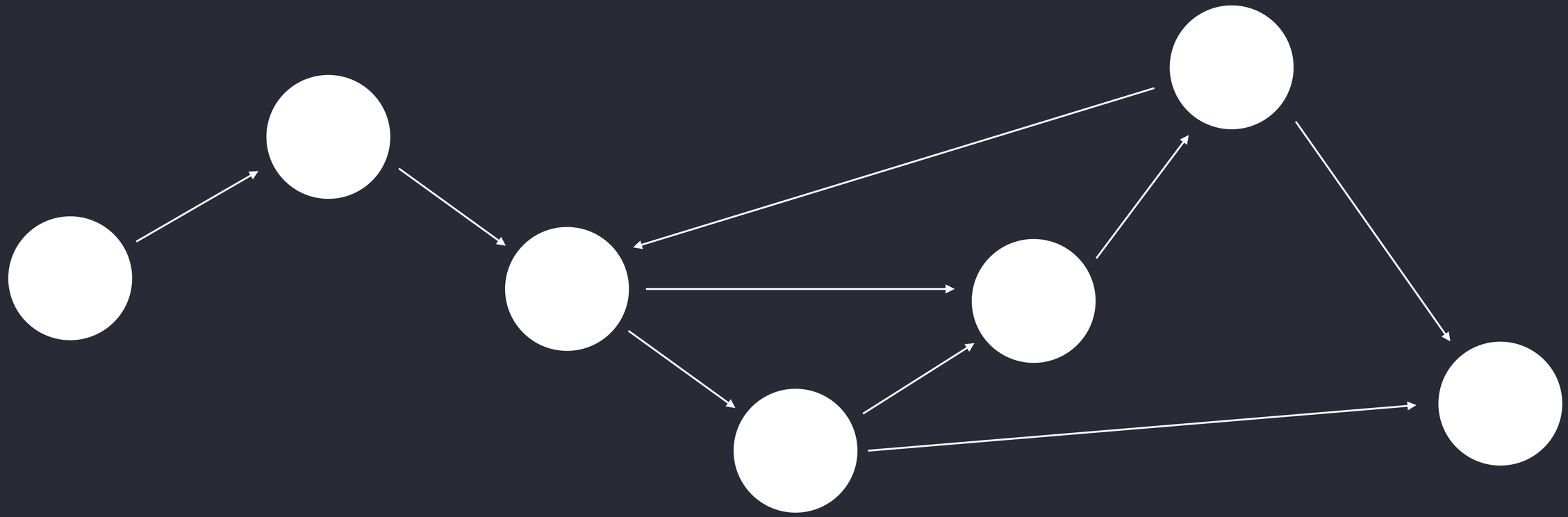
Hierarchical



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

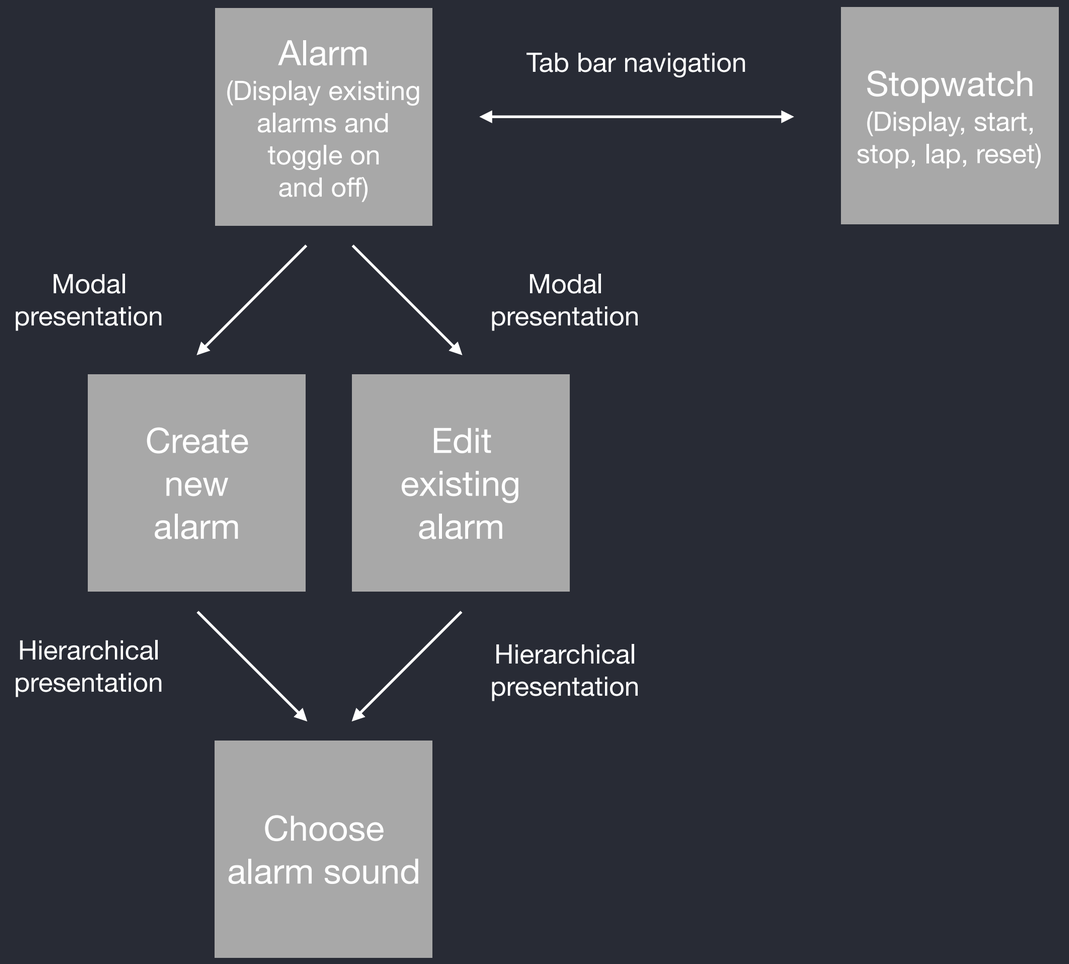
Example workflow

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



The End.