Introduction to iOS development with Swift

Lesson 5

Adrien Humilière
Trainline

adhumi+dant@gmail.com

- Table views
- Saving data

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

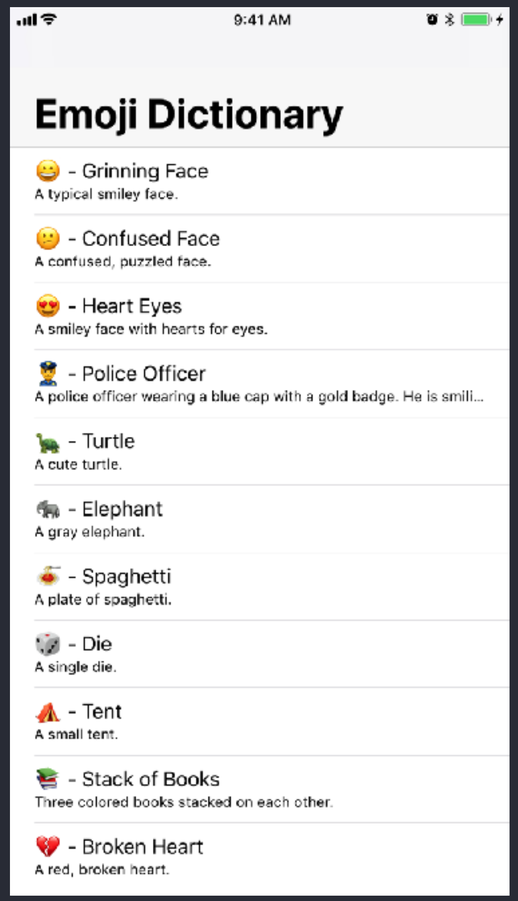
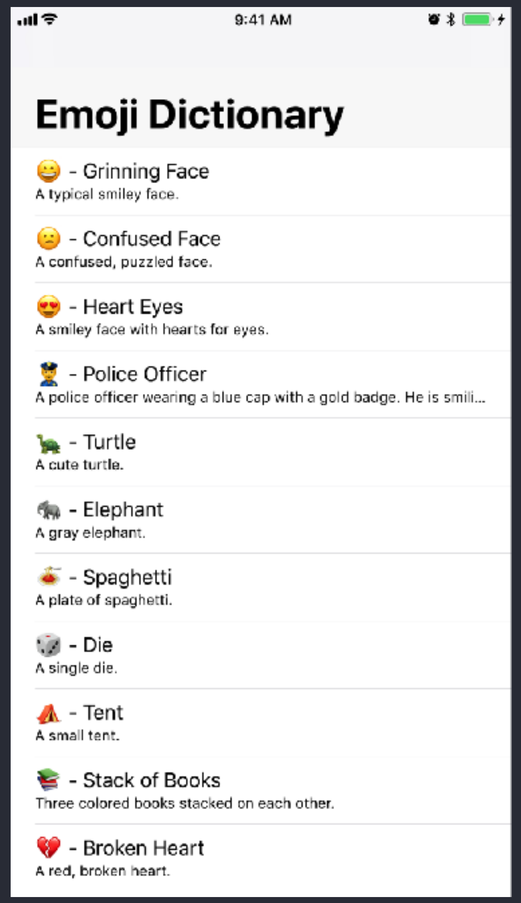
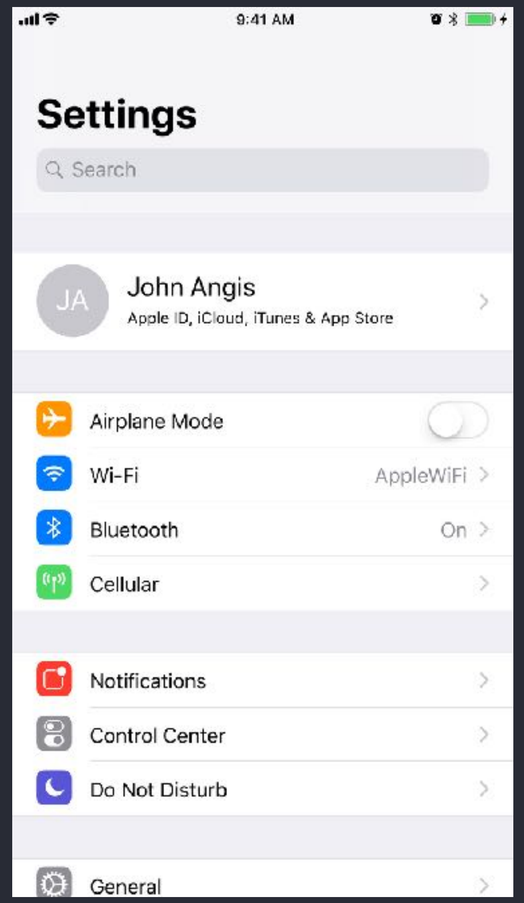


Table views

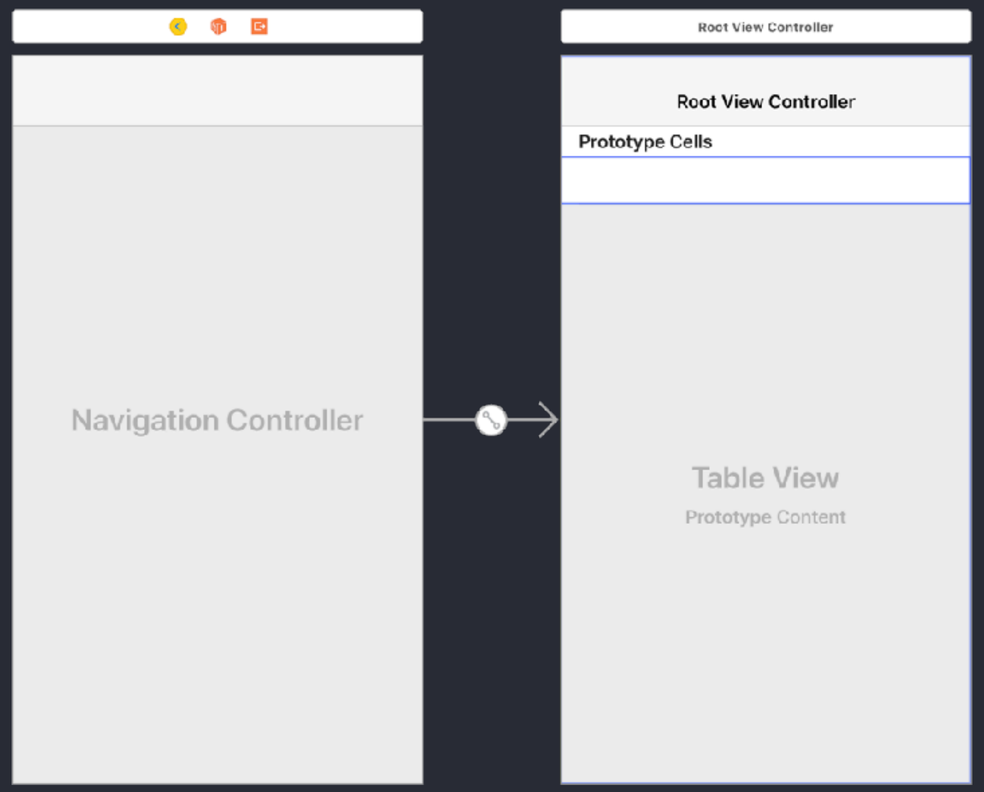

9:41 AM
Emoji Dictionary
- Grinning Face
A typical smiley face.
- Confused Face
A confused, puzzled face.
- Heart Eyes
A smiley face with hearts for eyes.
- Police Officer
A police officer wearing a blue cap with a gold badge. He is smili...
- Turtle
A cute turtle.
- Elephant
A gray elephant.
- Spaghetti
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

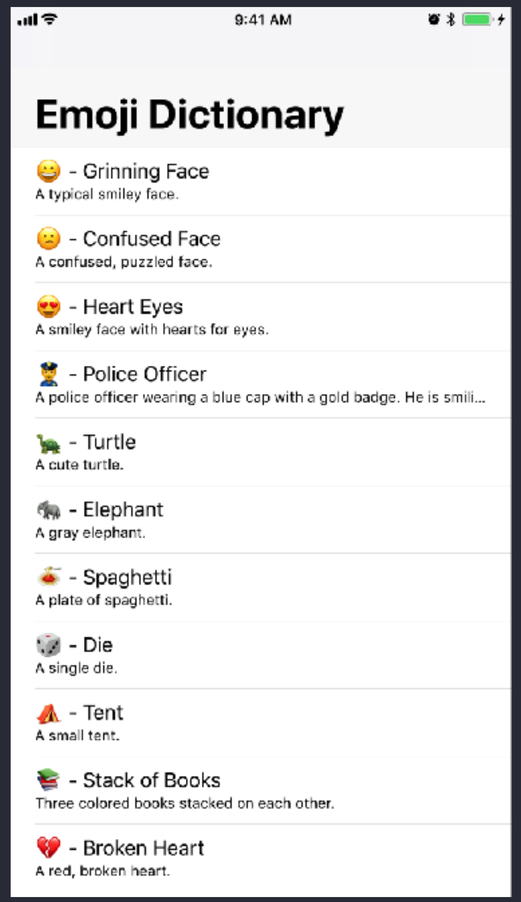
Anatomy of a table view

Two possible approaches to add table views:

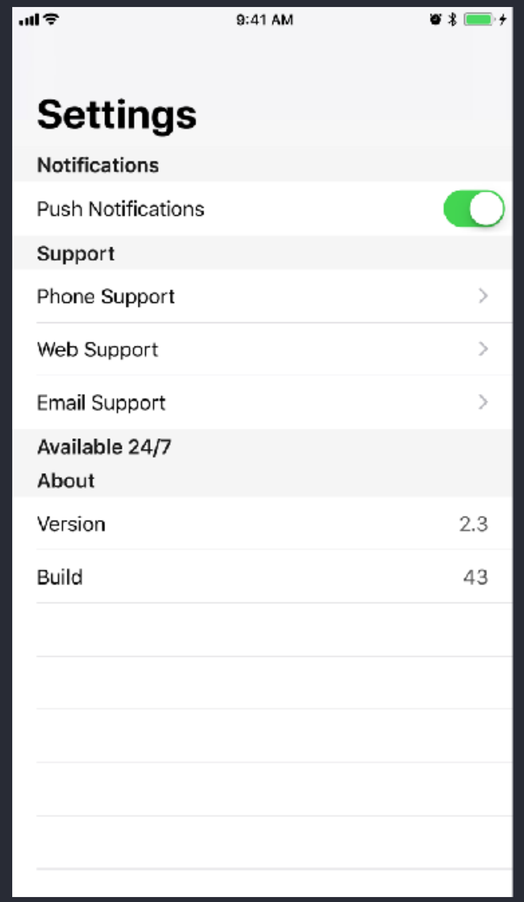
- Add a table view instance directly to a view controller's view
- Add a table view controller to your storyboard



Anatomy of a table view



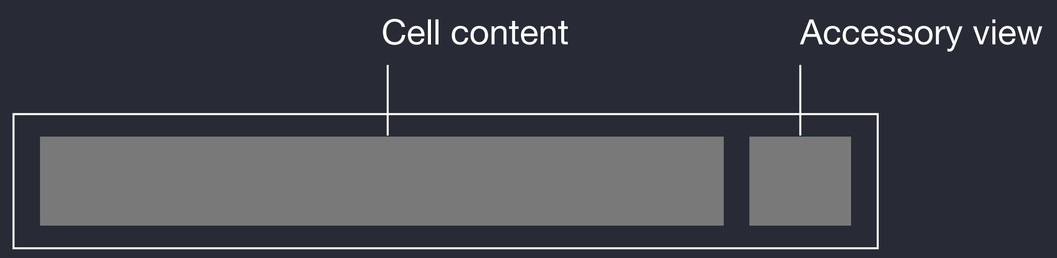
Plain



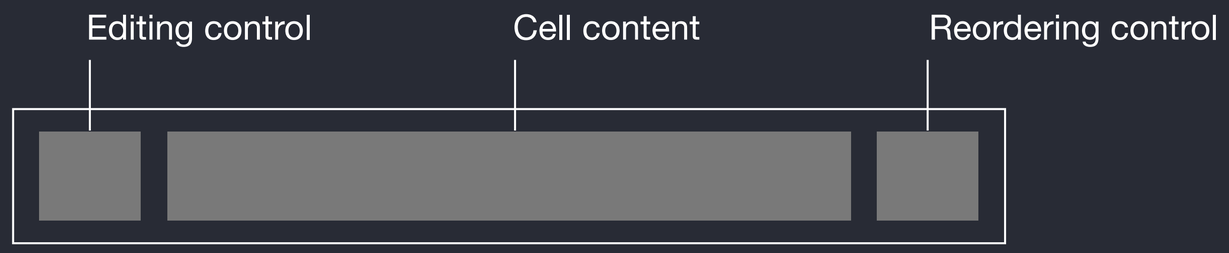
Grouped

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

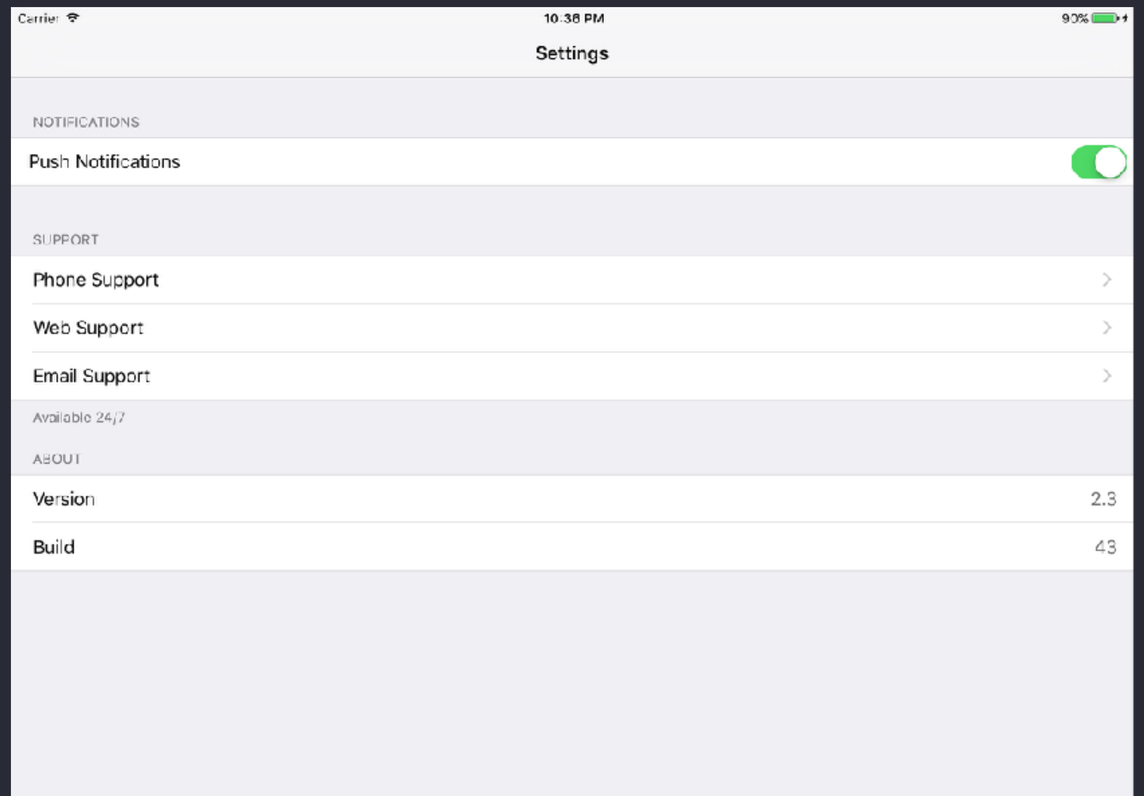
Anatomy of a table view - cells

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

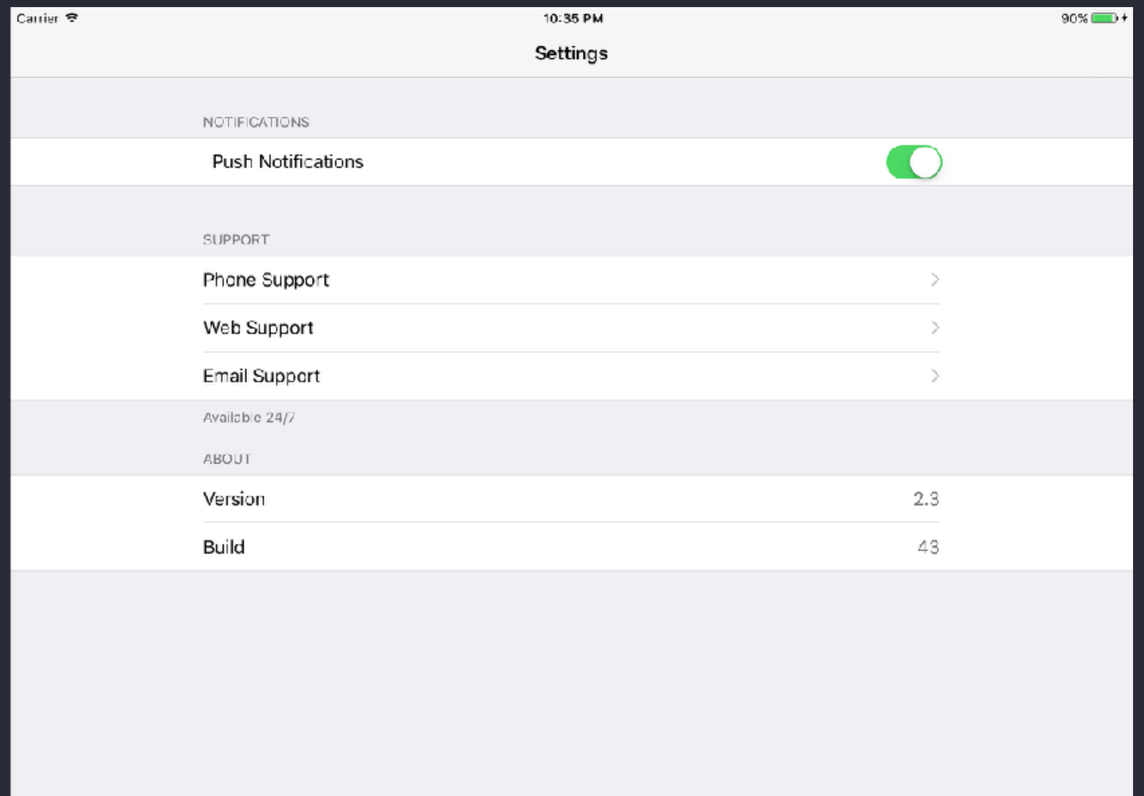
Anatomy of a table view - cells

→ Set `tableView.cellLayoutMarginsFollowReadableWidth` to `true`

Default



Adjusted



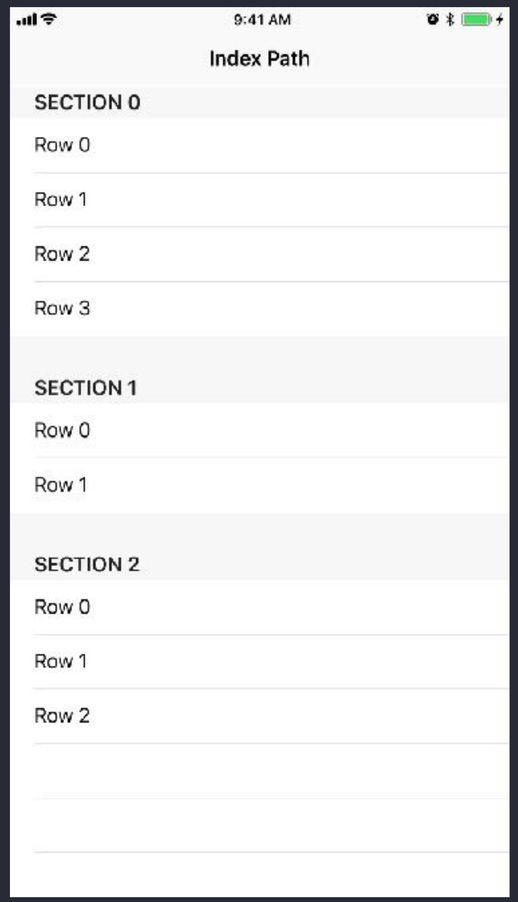
Index paths

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)
```

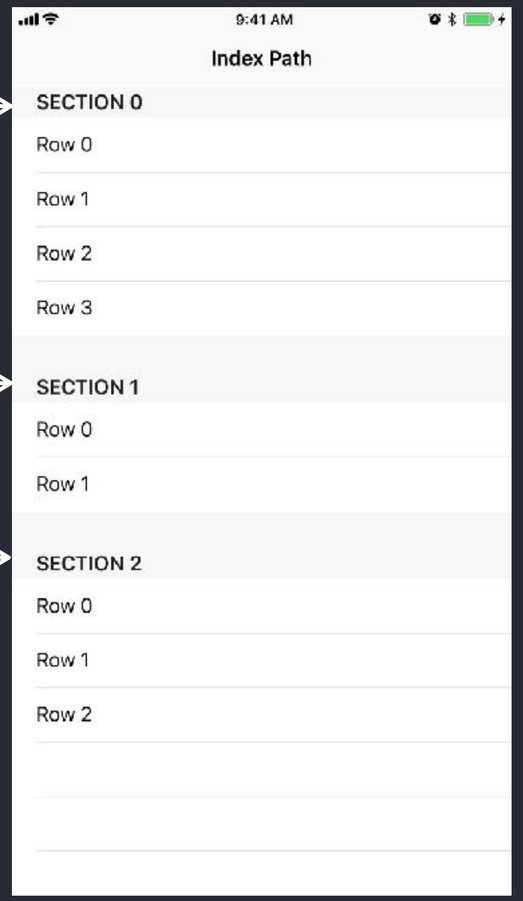
Table view protocols

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

UITableViewDataSource

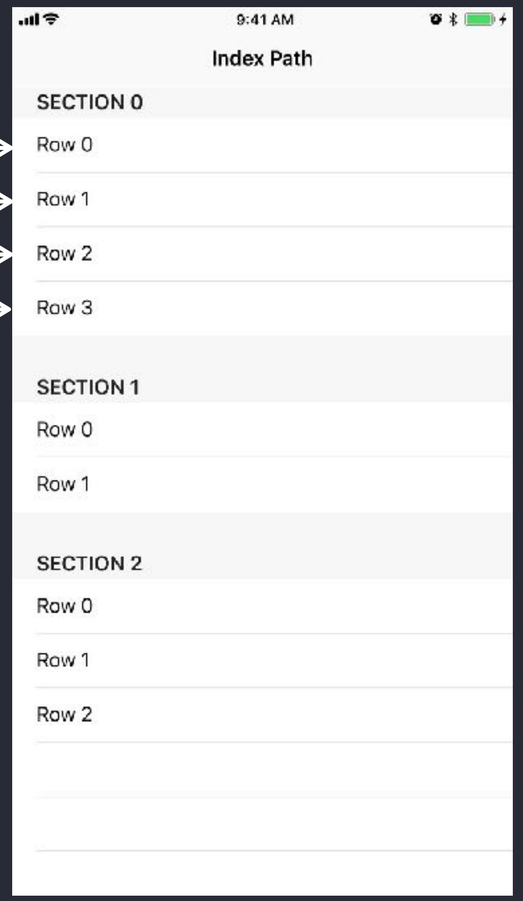
```
optional func numberOfSections(in
  tableView: UITableView) -> Int
```

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



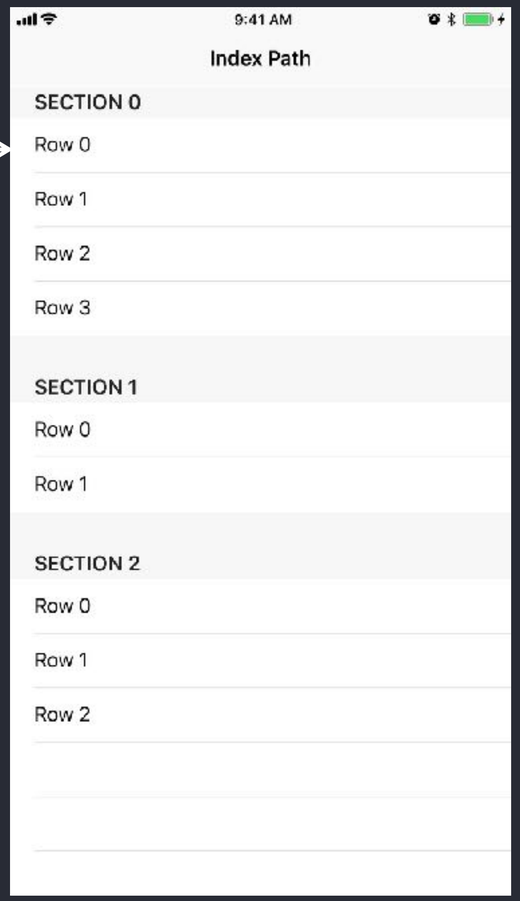
UITableViewDataSource

```
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()
```

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) {
   …
}
```

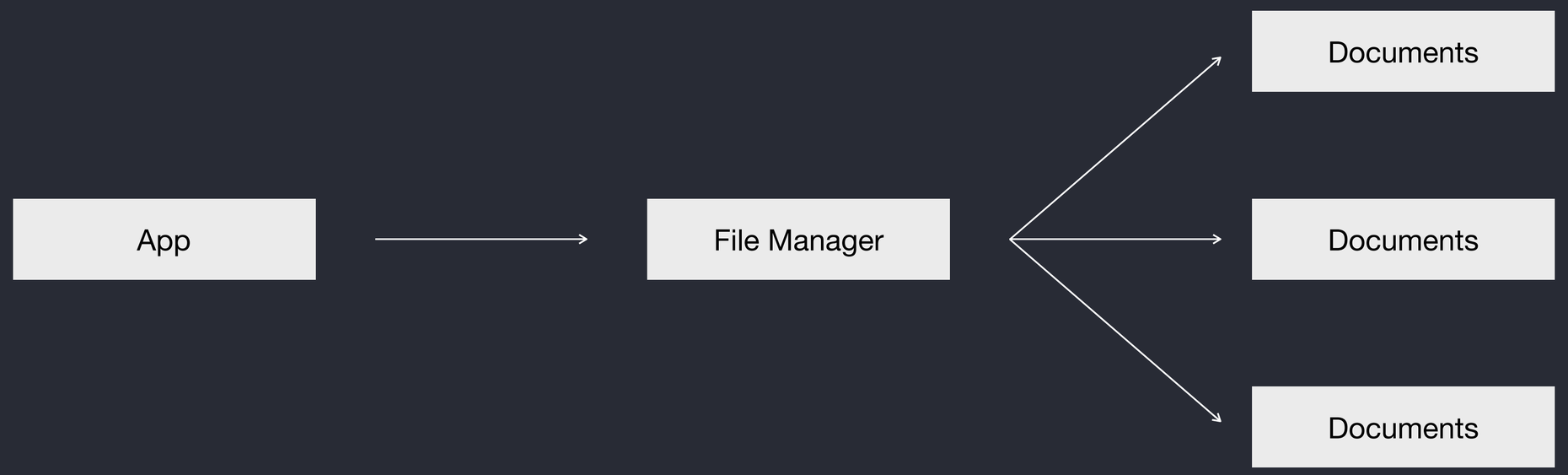
Encoding and decoding with Codable

```
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

The End.