Container Views

Outline

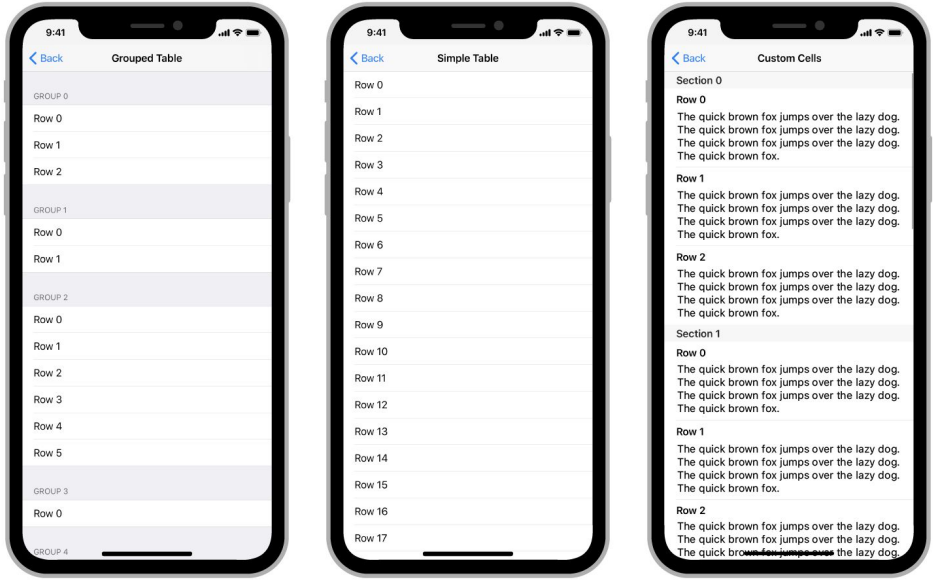
1. UIScrollView
2. UITableView
3. UICollectionView

1. UIScrollView

- A view that allows the scrolling and zooming of its contained views. `UIScrollView` is the superclass of several `UIKit` classes including `UITableView` and `UITextView`.
- The central notion of a `UIScrollView` object (or, simply, a scroll view) is that it is a view whose origin is adjustable over the content view.
- The scroll view must know the size of the content view so it knows when to stop scrolling; by default, it “bounces” back when scrolling exceeds the bounds of the content.
- The object that manages the drawing of content displayed in a scrollView should tile the content’s subviews so that no view exceeds the size of the screen. As users scroll in the scroll view, this object should add and remove subviews as necessary.

2. UITableView

- A table view displays a single column of vertically scrolling content, divided into rows and sections. Each row of a table displays a single piece of information related to your app. Sections let you group related rows together.



2. UITableView (cont)

- Table views are a collaboration between many different objects, including:

  - Cells. A cell provides the visual representation for your content. You can use the default cells provided by `UIKit` or define custom cells to suit the needs of your app.

  - Table view controller. You typically use a `UITableViewController` object to manage a table view. You can use other view controllers too, but a table view controller is required for some table-related features to work.

  - Your data source object. This object adopts the `UITableViewDataSource` protocol and provides the data for the table.

  - Your delegate object. This object adopts the `UITableViewDelegate` protocol and manages user interactions with the table's contents.

2. UITableView (cont)

- `UITableViewDataSource` protocol contains methods adopted by the object you use to manage data and provide cells for a table view.

- Table views manage only the presentation of their data, they do not manage the data itself.

2. UITableView (cont)

- Other responsibilities of the data source object include:
  - Reporting the number of sections and rows in the table.
  - Providing cells for each row of the table.
  - Providing titles for section headers and footers.
  - Configuring the table's index, if any.
  - Responding to user- or table-initiated updates that require changes to the underlying data.

2. UITableView (cont)

- Only two methods of this protocol are required, and they are shown in the following example code.

```swift
// Return the number of rows for the table.
override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return 0
}

// Provide a cell object for each row.
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    // Fetch a cell of the appropriate type.
    let cell = tableView.dequeueReusableCell(withIdentifier: "cellTypeIdentifier", for: indexPath)

    // Configure the cell's contents.
    cell.textLabel!.text = "Cell text"

    return cell
}
```

2. UITableView (cont)

- `UITableViewDelegate` protocol contains methods for managing selections, configuring section headers and footers, deleting and reordering cells, and performing other actions in a table view.

- The table view specifies rows and sections using IndexPath objects.

2. UITableView (cont)

- Use the methods of this protocol to manage the following features:

  - Create and manage custom header and footer views.
  - Specify custom heights for rows, headers, and footers.
  - Provide height estimates for better scrolling support.
  - Indent row content.
  - Respond to row selections.
  - Respond to swipes and other actions in table rows.
  - Support editing the table's content.

2. UITableView (cont)

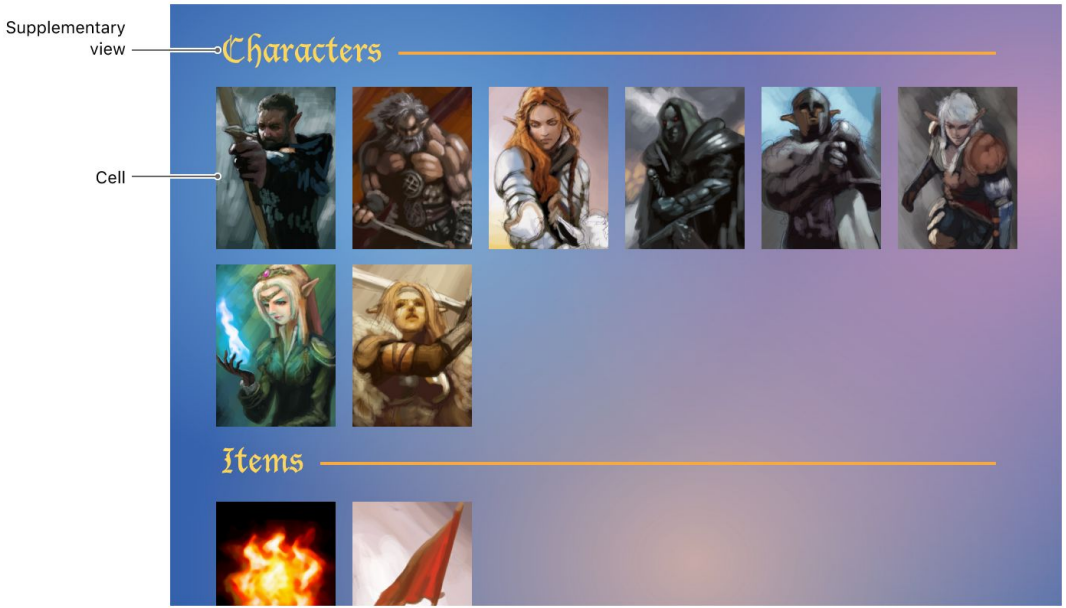
- The most 2 common methods when handling table view delegate:

```
// Tells the delegate that the specified row is now selected.
func tableView(_: UITableView, didSelectRowAt _: IndexPath) {}

// Tells the delegate that the specified row is now deselected.
func tableView(_: UITableView, didDeselectRowAt _: IndexPath) {}
```

3. UICollectionView

- An object that manages an ordered collection of data items and presents them using customizable layouts.



3. UICollectionView (cont)

- Collection views are a collaboration between many different objects, including:

  - Cells. A cell provides the visual representation for your content. You can use the default cells provided by `UIKit` or define custom cells to suit the needs of your app.

  - Collection view controller. You typically use a `UICollectionViewController` object to manage a collection view. You can use other view controllers too, but a collection view controller is required for some collection-related features to work.

  - Your data source object. This object adopts the `UICollectionViewDataSource` protocol and provides the data for the table.

  - Your delegate object. This object adopts the `UICollectionViewDelegate` protocol and manages user interactions with the table's contents.

3. UICollectionView (cont)

- `UICollectionViewDataSource` protocol contains methods adopted by the object you use to manage data and provide views for a collection view.

- A data source object represents your app's data model and vends information to the collection view as needed.

- It also handles the creation and configuration of cells and supplementary views used by the collection view to display your data.

3. UICollectionView (cont)

- Only two methods of this protocol are required, and they are shown in the following example code.

```swift
// Return number of items in the secified section
func collectionView(_: UICollectionView, numberOfItemsInSection _: Int) -> Int {
    return contacts.count
}

// Config the cell corresponds to the specified item in the collection view
func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
    let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "cellIdentifier", for: indexPath)
    cell.configCell(contacts[indexPath.row])
    return cell
}
```

3. UICollectionView (cont)

- `UICollectionViewDelegate` protocol defines methods that allow you to manage the selection and highlighting of items in a collection view and to perform actions on those items. The methods of this protocol are all optional.

- Many methods of this protocol take `IndexPath` objects as parameters. To support collection views, `UIKit` declares a category on `IndexPath` that enables you to get the represented item index and section index, and to construct new index path objects from item and index values.

3. UICollectionView (cont)

- The most 2 common methods when handling collection view delegate:

```
// Tells the delegate that the item at the specified index path was selected.
func collectionView(_: UICollectionView, didSelectItemAt _: IndexPath) {}

// Tells the delegate that the item at the specified path was deselected.
func collectionView(_: UICollectionView, didDeselectItemAt _: IndexPath) {}
```

Question & Answer?