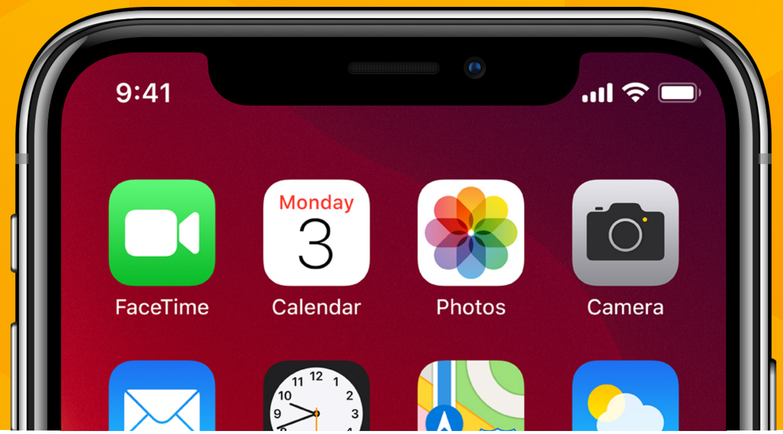


iOS Application Development

Lecture 8: Working with the Web Part 2 • Advanced Data DisplayPart 1

Prof. Dr. Jan Borchers
Media Computing Group
RWTH Aachen University

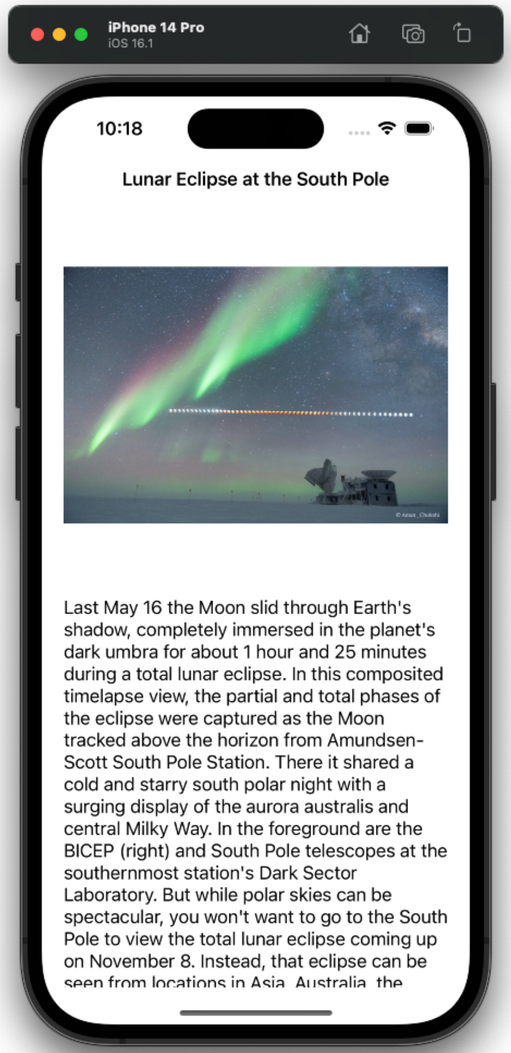
WS ’22/’23 • hci.rwth-aachen.de/ios

RWTH AACHEN UNIVERSITY

Working With the Web

NASA Astronomy Picture of the Day App

1. Create Url ✓
2. Request Data with API keys ✓
3. Create a Swift model
4. Decode JSON
5. Update UI



The Swift Model

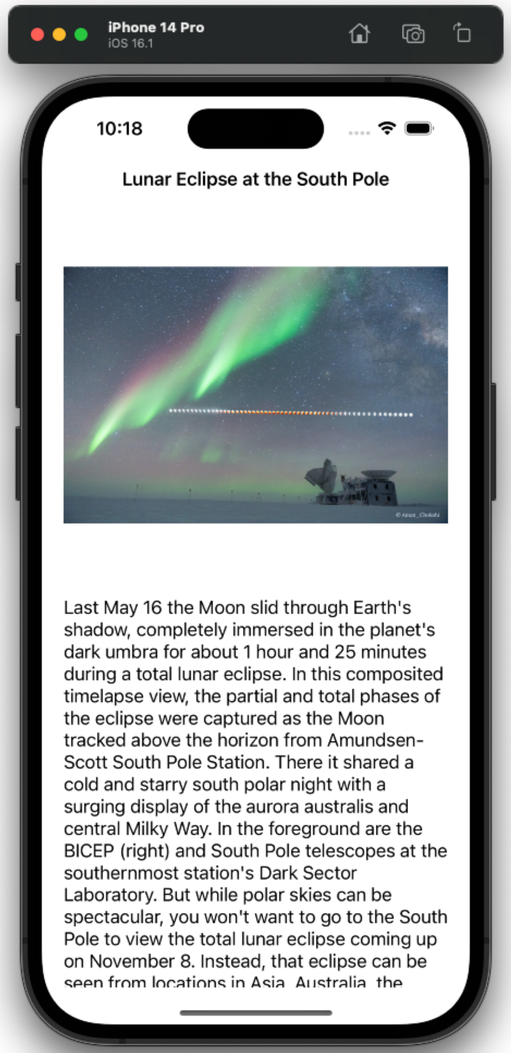
- The PhotoInfo model:

```
{
  "date": "2005-02-22",
  "explanation": "Are Saturn's auroras like
Earth's?  To help answer this question, the
Hubble Space Telescope and the Cassini
spacecraft monitored Saturn's South Pole
simultaneously as Cassini closed in on the gas
giant in January 2004.  Hubble snapped images
in ultraviolet light, while Cassini recorded
radio emissions and monitored the solar wind.
Like on Earth,
…",
  "hdurl": "http://apod.nasa.gov/apod/image/
0502/saturnauroras_hst_big.jpg",
  "media_type": "image",
  "service_version": "v1",
  "title": "Persistent Saturnian Auroras",
  "url": "http://apod.nasa.gov/apod/image/0502/
saturnauroras_hst.jpg"
}
```

```
struct PhotoInfo: Codable {
    var title: String
    var description: String
    var url: URL
    var copyright: String?

    enum CodingKeys: String, CodingKey {
        case title
        case description = "explanation"
        case url
        case copyright
    }
}
```

NASA Astronomy Picture of the Day App

1. Create Url ✓
2. Request Data with API keys ✓
3. Create a Swift model ✓
4. Decode JSON
5. Update UI



Decode JSON

```
var components = URLComponents(string: "https://api.nasa.gov/planetary/apod")!
components.queryItems = [
    "api_key": "DEMO_KEY",
    "date": "2013-07-16"
].map { URLQueryItem(name: $0.key, value: $0.value) }

// Perform the network request
Task {
    let (data, response) = try await URLSession.shared.data(from: components.url!)
    let jsonDecoder = JSONDecoder()
    if let httpResponse = response as? HTTPURLResponse,
        httpResponse.statusCode == 200,
        let photoInfo = try? jsonDecoder.decode(PhotoInfo.self, from: data) {
        print(photoInfo)
    }
}
```

Async Calls

```
func fetchPhotoInfo() -> PhotoInfo{
    // Build the URL
    var components = URLComponents(string: "https://api.nasa.gov/planetary/apod")!
    components.queryItems = [
        "api_key": "DEMO_KEY",
        "date": "2013-07-16"
    ].map { URLQueryItem(name: $0.key, value: $0.value) }

    // Perform the network request
    let (data, response) = try await URLSession.shared.data(from: components.url!)
    let jsonDecoder = JSONDecoder()

    if let httpResponse = response as? HTTPURLResponse,
        httpResponse.statusCode == 200,
        let photoInfo = try? jsonDecoder.decode(PhotoInfo.self, from: data) {
        return photoInfo
    }
}
```

'async' call in a function that does not support concurrency

Add 'async' to function 'fetchPhotoInfo()' to make it asynchronous Fix

Errors thrown from here are not handled

Async Calls

```
func fetchPhotoInfo() async throws -> PhotoInfo{
    // Build the URL
    var components = URLComponents(string: "https://api.nasa.gov/planetary/apod")!
    components.queryItems = [
        "api_key": "DEMO_KEY",
        "date": "2013-07-16"
    ].map { URLQueryItem(name: $0.key, value: $0.value) }

    // Perform the network request
    let (data, response) = try await URLSession.shared.data(from: components.url!)
    let jsonDecoder = JSONDecoder()

    if let httpResponse = response as? HTTPURLResponse,
        httpResponse.statusCode == 200,
        let photoInfo = try? jsonDecoder.decode(PhotoInfo.self, from: data) {
        return photoInfo
    }
}
```

Missing return in global function expected to return 'PhotoInfo'

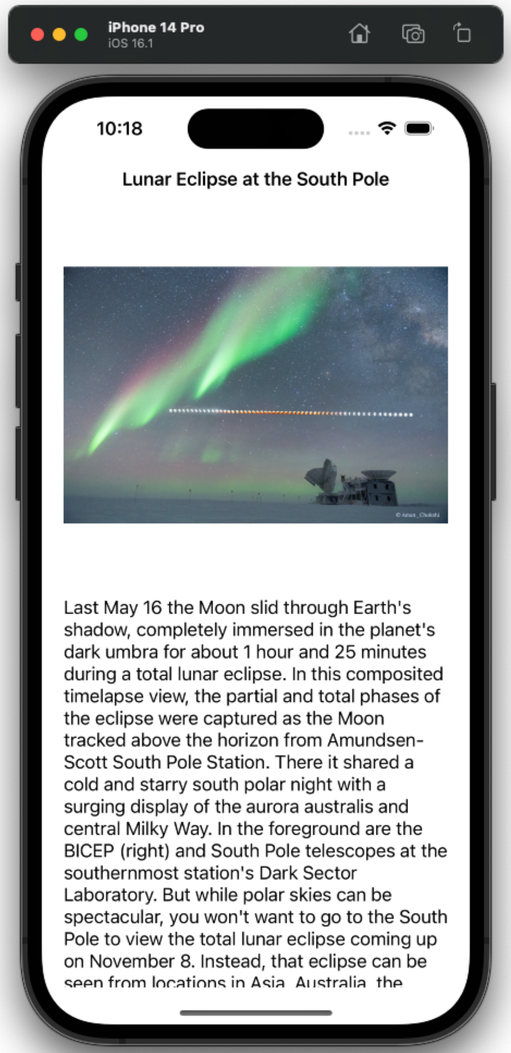
Async Calls

```swift
enum PhotoInfoError: Error, LocalizedError {
    case itemNotFound
}
```

```swift
func fetchPhotoInfo() async throws -> PhotoInfo{
    // Buid the URL
    var components = URLComponents(string: "https://api.nasa.gov/planetary/apod")!
    components.queryItems = [
        "api_key": "DEMO_KEY",
        "date": "2013-07-16"
    ].map { URLQueryItem(name: $0.key, value: $0.value) }

    // Perform the network request
    let (data, response) = try await URLSession.shared.data(from: components.url!)

    guard let httpResponse = response as? HTTPURLResponse,
              httpResponse.statusCode == 200 else {
            throw PhotoInfoError.itemNotFound
    }

    let jsonDecoder = JSONDecoder()
    let photoInfo = try jsonDecoder.decode(PhotoInfo.self, from: data)
    return(photoInfo)
}
```

NASA Astronomy Picture of the Day App

1. Create Url ✓
2. Request Data with API keys ✓
3. Create a Swift model ✓
4. Decode JSON ✓
5. Update UI



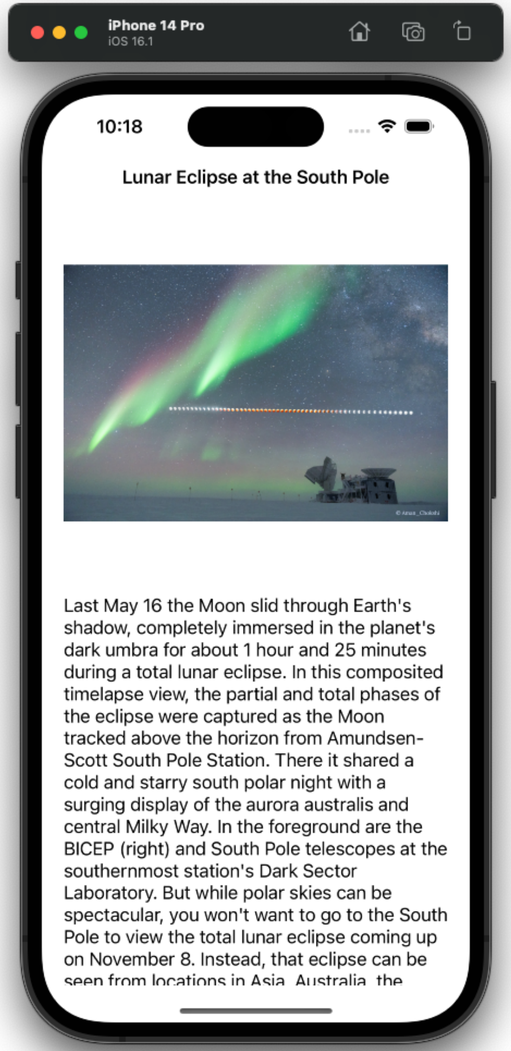
Update the UI

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

    Task {
        do {
            let photoInfo = try await fetchPhotoInfo()
            updateUI(with: photoInfo)
        } catch {
            updateUI(with: error)
        }
    }
}
```

```
func updateUI(with photoInfo: PhotoInfo) {
    Task {
        do {
            let image = try await fetchImage(from:
                photoInfo.url)
            title = photoInfo.title
            imageView.image = image
            descriptionLabel.text =
                photoInfo.description
            copyrightLabel.text = photoInfo.copyright
        } catch {
            updateUI(with: error)
        }
    }
}
```

NASA Astronomy Picture of the Day App

1. Create Url ✓
2. Request Data with API keys ✓
3. Create a Swift model ✓
4. Decode JSON ✓
5. Update UI ✓



Concurrency

Multi Threading in iOS

- Run multiple tasks at the same time
- Run slow or expensive tasks in the background
- Free the main thread so it responds to the UI

Synchronous and Asynchronous

- Synchronous
  - One task completes before another begins
  - Ties up the main thread (main queue)
- Asynchronous
  - Multiple tasks run simultaneously on multiple threads (concurrency)
  - Tasks run in the background thread (background queue)
  - Frees up the main thread

Swift Concurrency

- Swift uses Actors to protect against concurrent updates
- A special Actor called the MainActor is used for UIKit
  - Standard UIKit controllers use the MainActor
  - Safe to update UI in a `Task`'s closure that was created in the context of the MainActor
  - Code after a method that can suspend (marked with await) will run synchronously in the context of the MainActor

Grand Central Dispatch

- Allows your app to execute multiple tasks concurrently on multiple threads
- Assigns tasks to "dispatch queues" and assigns priority
- Controls when your code is executed

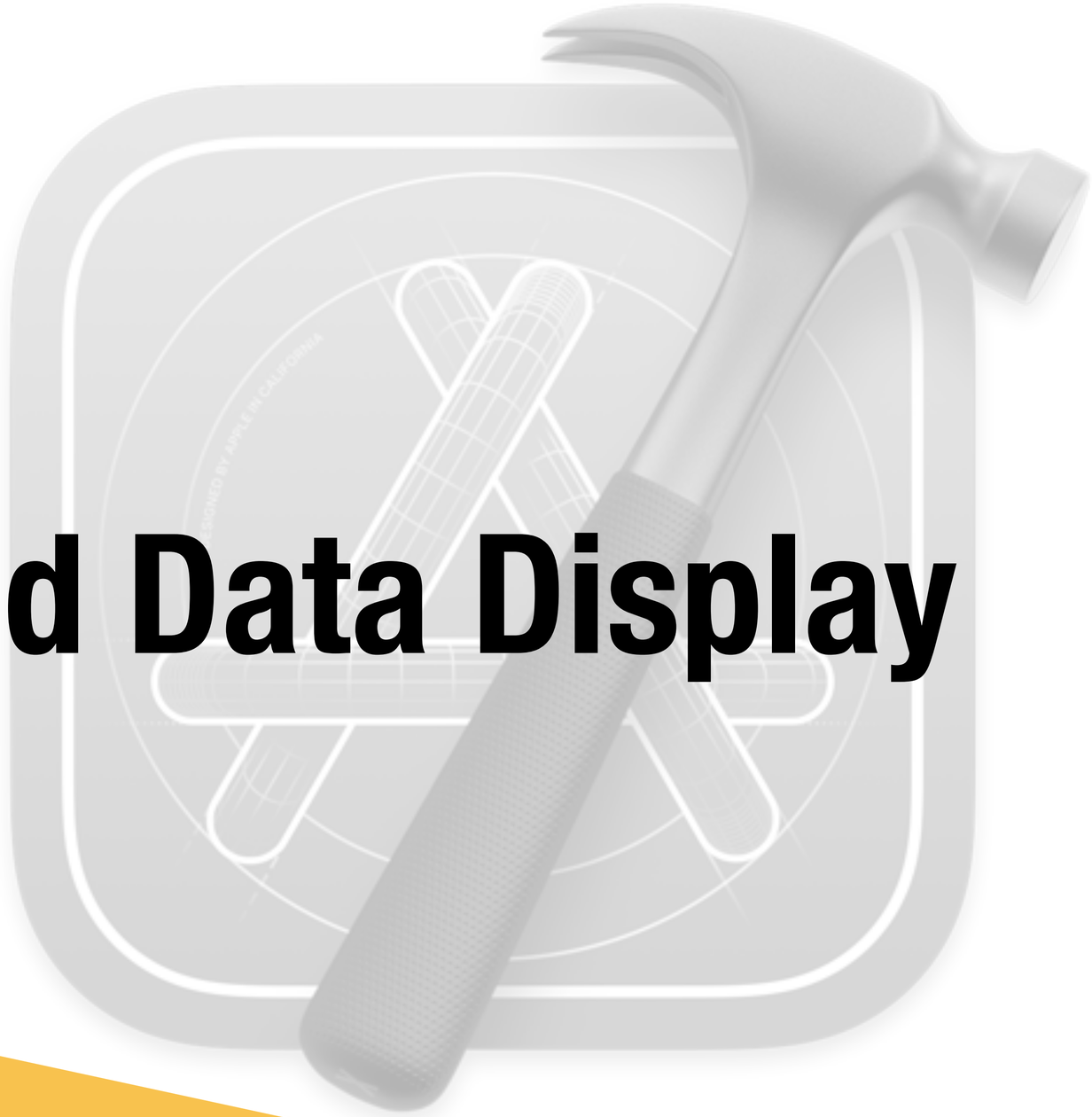
Grand Central Dispatch

- Main queue
  - Created when an app launches
  - Highest priority
  - Used to update the UI and respond quickly to user input
- Background queues
  - Lower priority
  - Used to run long-running operations

Dispatch Queue

```
DispatchQueue.global(qos: .background).async {
    // Do some background work
    DispatchQueue.main.async {
        // Update the UI to indicate the work has been completed
    }
}
```

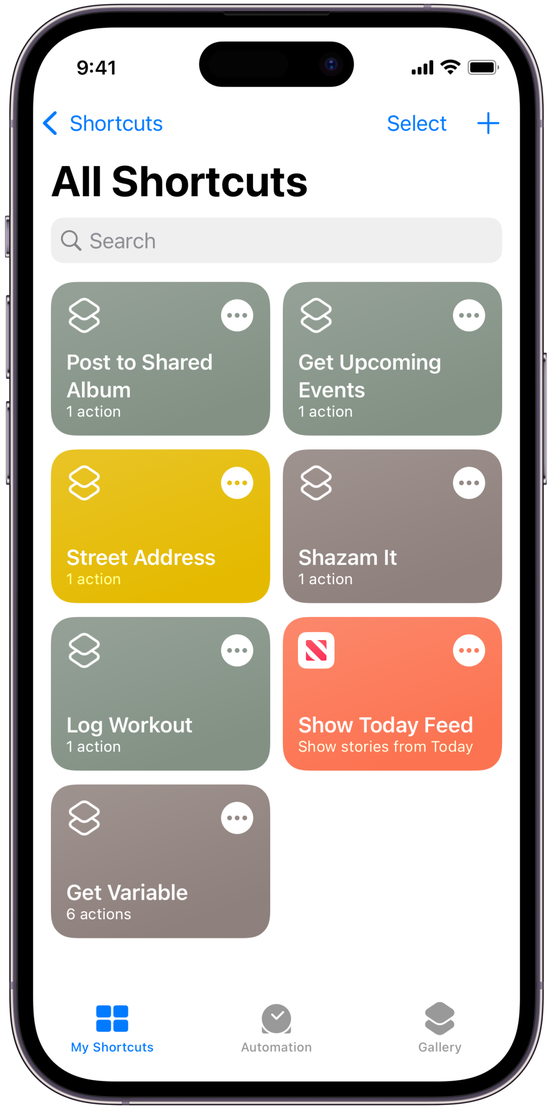
Advanced Data Display



CollectionView

Collection Views

- Implemented by the UICollectionView class
- A subclass of UIScrollView
  - Displays a collection of items using a separate layout object
  - Displays zero to many items
  - Nearly infinite layout options



9:41

Shortcuts

Select

All Shortcuts

Search

Get Details of App Store App
1 action

Add New Reminder
1 action

Show Today Feed
Show stories from Tod...

Street Address
1 action

Get Travel Time
1 action

Post to Shared Album
1 action

Get Upcoming Events
1 action

My Shortcuts | Automation | Gallery

9:41

Browse

On My iPhone

Search

Budget
2:43 PM
163 KB

Invoice
2:47 PM
179 KB

Potluck recipe
2:45 PM
239 KB

Project Ideas
0 items

Resumé
2:41 PM
229 KB

5 items, Zero KB available

Recents | Browse

9:41

All Photos

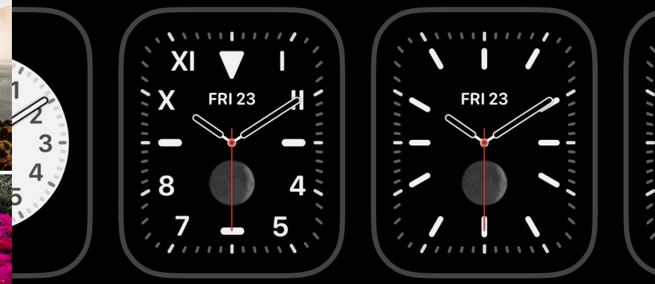
Select

All Photos | For You | Albums | Search

9:41

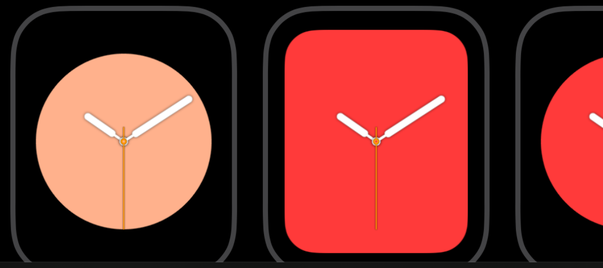
Face Gallery

California



Chronograph

Color



My Watch | Face Gallery | App Store

Anatomy of a Collection View

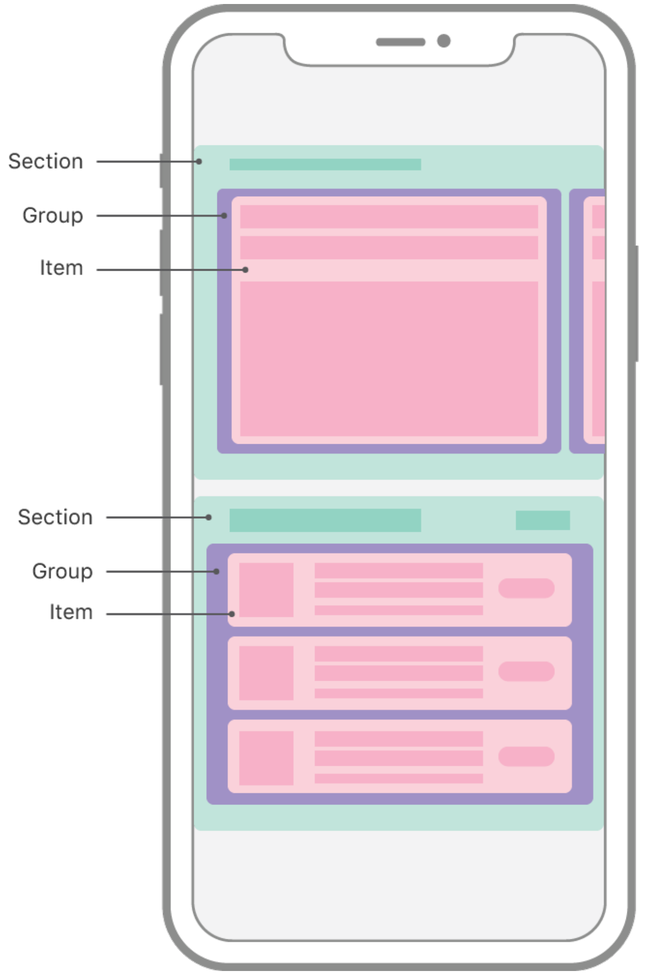
- Similar to UITableView APIs
  - Cell dequeueing and reuse
  - IndexPath-based
  - UIKit provides UICollectionViewController
- Cells
  - No built-in styles like table view
  - Add subviews to UICollectionViewCell's contentView or subclass
  - UICollectionViewListCell subclass does support content configuration that is similar to UITableViewCell

Collection View Layout

- All collection views rely on a separate layout object
- Layouts define how views appear in the collection view
- UICollectionViewLayout is an abstract base class
- UIKit provides UICollectionViewFlowLayout
- UICollectionViewFlowLayout handles many common cases, e.g. grids

Compositional Layout

- Compositional layout
  - Subclassing UICollectionViewLayout is a non-trivial task
  - Compositional layouts avoid that and provide greater flexibility
  - Create layouts programmatically
    - Define items, groups of items, and sections



CollectionView Demo