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
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:

```swift
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
    }
}
```

```json
{
    "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"
}
```
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
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)
    }
}
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
    }
}
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
    }
}

Async Calls

Prof. Dr. Jan Borchers: iOS Application Development
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!)

    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)
}

enum PhotoInfoError: Error, LocalizedError {
    case itemNotFound
}
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
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

Last May 16 the Moon slid through Earth’s shadow, completely immersed in the planet’s dark umbra for about 1 hour and 25 minutes during a total lunar eclipse. In this compositied timelapse view, the partial and total phases of the eclipse were captured as the Moon tracked above the horizon from Amundsen-Scott South Pole Station. There is shared a cold and starry south polar night with a surging display of the aurora australis and central Milky Way. In the foreground are the BICEP (Right) and South Pole telescopes at the southernmost station’s Dark Sector Laboratory. But while polar skies can be spectacular, you won’t want to go to the South Pole to view the total lunar eclipse coming up on November 8. Instead, that eclipse can be seen from latitudes in Asia, Australia, the...
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

```swift
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
Anatomy of a Collection View

• Similar to UITableView APIs
  • Cell dequeuing 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 UICollectionViewFlowLayout 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