EE 443/593 Mobile Application Development

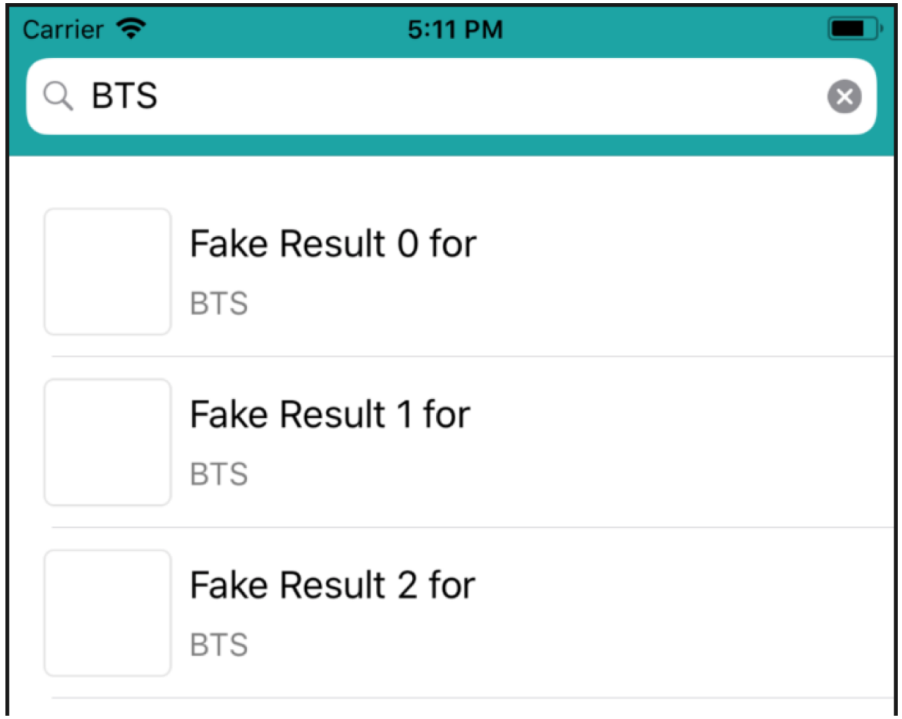
Irving Delgado

Lecture 24
Chapter 34: Networking

Store Search

Where are we?

- Ch 32: Implemented a SearchBar
- Ch 33: Custom Table Cells
- Implemented a “Fake” Data model
- Changed our tab to a teal color.
- Nib files for different search results.



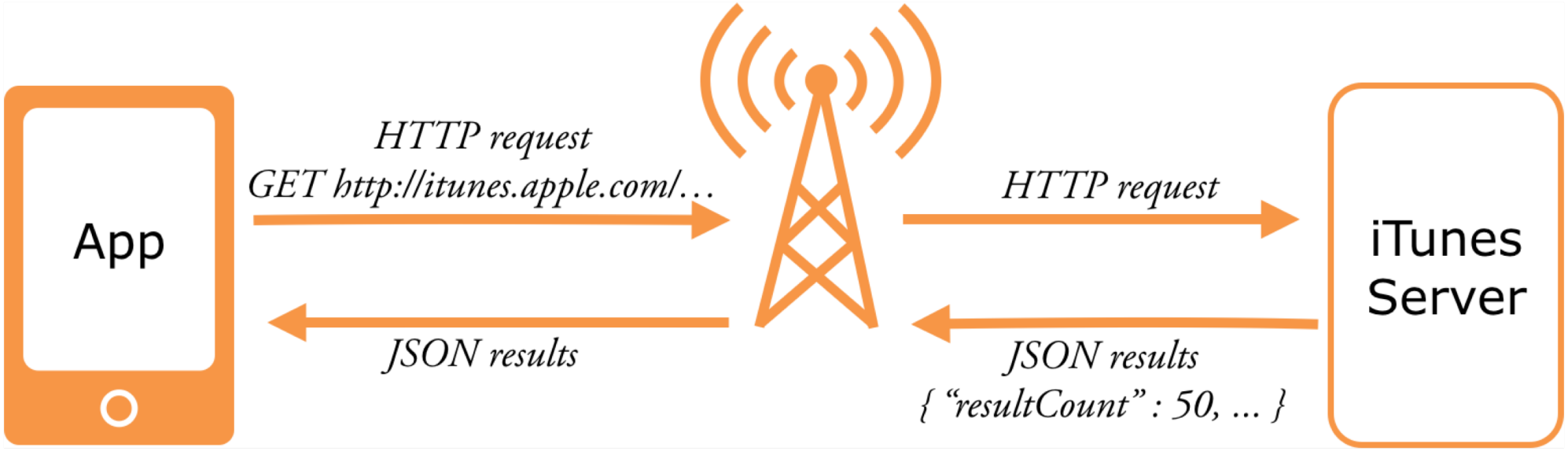
To Do List

- Query the iTunes web service
- Send an HTTP Request
- Parse JSON
- Work with the JSON Results
- Sort the Search Results

Query the iTunes web service

Query the iTunes web service

- Example GET request.

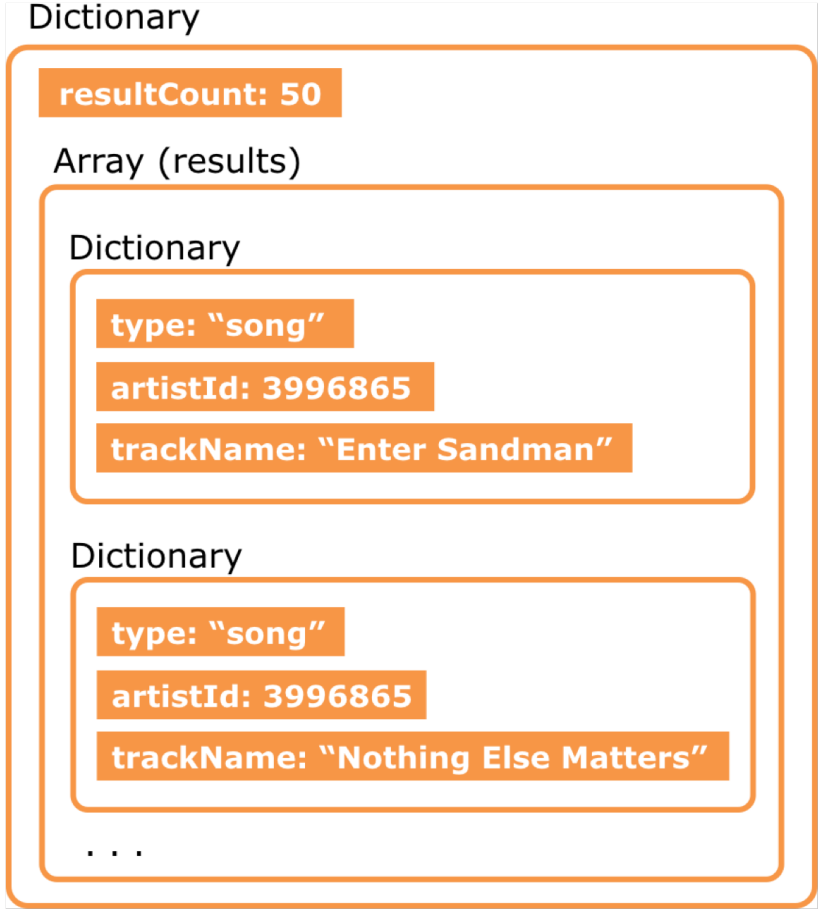


JSON

- JavaScriptObjectNotation
- Commonly used to send structured data back-and-forth between servers and clients.
- Similar to XML.

iTunes JSON

- iTunes JSON data consists of a dictionary with array object containing multiple Dictionaries.



NM STATE

iTunes JSON Example

NM STATE

Synchronous or Asynchronous Networking?

Synchronous Networking

- Synchronous Networking = Bad!
- Performs HTTP request on Main Thread.
- Easier to program than Asynchronous.
- Makes the UI Unresponsive during the network call.

Asynchronous Networking

- Typically runs on a background thread.
- Does not lock up the app during a network request.
- Done through queue's or GCD (Grand Central Dispatch).
- Put the job in a closure and pass the closure through a queue.

Send an HTTP Request

Making a URL Request

Method for URL Request

➤ Add a new method to **SearchViewController.swift**:

```
// MARK:- Private Methods
func iTunesURL(searchText: String) -> URL {
  let urlString = String(format:
      "https://itunes.apple.com/search?term=%@", searchText)
  let url = URL(string: urlString)
  return url!
}
```

- Use string format to resemble an iTunes search.
- Return string as a URL object.

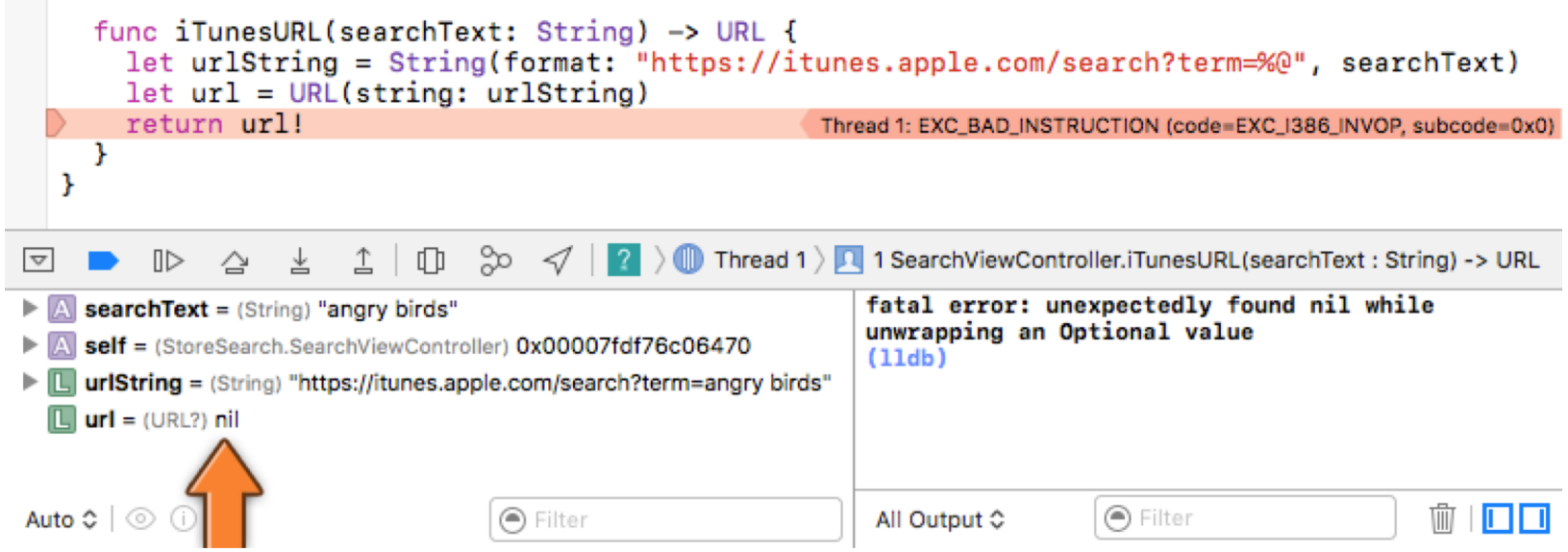
Making a URL Request

- Update searchBarSearchButtonClicked to perform iTunes search.

```
func searchBarSearchButtonClicked(_ searchBar: UISearchBar) {
  if !searchBar.text!.isEmpty {
    searchBar.resignFirstResponder()

    hasSearched = true
    searchResults = []

    let url = iTunesURL(searchText: searchBar.text!)
    print("URL: '\(url)'")

    tableView.reloadData()
  }
}
```

App Crash?

- A space is not a valid character in a URL. Other symbols like < or > also aren't valid and must be escaped.



URL Encoding

- String can do this encoding for us to allow use of spaces and other characters with addingPercentEncoding method.

```
func iTunesURL(searchText: String) -> URL {
  let encodedText = searchText.addingPercentEncoding(
      withAllowedCharacters: CharacterSet.urlQueryAllowed)!
  let urlString = String(format:
      "https://itunes.apple.com/search?term=%@", encodedText)
  let url = URL(string: urlString)
  return url!
}
```

- UTF-8 is a version of Unicode that is efficient for storing text.
- % sign denotes an escaped character and comes with a code.

```
func performStoreRequest(with url: URL) -> String? {
  do {
   return try String(contentsOf: url, encoding: .utf8)
  } catch {
   print("Download Error: \(error.localizedDescription)")
   return nil
  }
}
```

- Do-try-catch block to handle potential networking errors.

```
if let jsonString = performStoreRequest(with: url) {
  print("Received JSON string '\(jsonString)'")
}
```

- When in doubt, UTF-8 will almost always work.

Parse JSON

JSON or XML?

JSON

XML

```
<?xml version="1.0" encoding="utf-8"?>
<iTunesSearch>
  <resultCount>5</resultCount>
  <results>
    <song>
      <artistName>Metallica</artistName>
      <trackName>Enter Sandman</trackName>
    </song>
    <song>
      <artistName>Metallica</artistName>
      <trackName>Nothing Else Matters</trackName>
    </song>
    . . . and so on . . .
  </results>
</iTunesSearch>
```

NM STATE

Documentation

- You do not get to choose the data type your network call returns.
- Read API documentation carefully.
- Consider it a luxury if you have options between XML and JSON!

Parsing the JSON Data

- To parse the data, we need to add a data model for the results wrapper that resembles the JSON results.
- Model the results wrapper to include a result count and an Array of SearchResults objects.
- Update performStoreRequest method to output Data instead of String.
- Add a parsing Method to utilize the JSON Decoder.
- JSON Decoder Object converts the response data from the server into a temporary Results Array from which we can extract the properties we want.

Printing the Results

- Results returned an array of StoreSearch Objects.

- Modify Data Model to conform to CustomStringConvertable protocol to allow an object to have a custom string representation

Initial Results

- Doesn't quite tell us anything:

```
URL: 'https://itunes.apple.com/search?term=Metallica'
Got results: [StoreSearch.SearchResult, StoreSearch.SearchResult,
StoreSearch.SearchResult, StoreSearch.SearchResult,
StoreSearch.SearchResult, StoreSearch.SearchResult,
StoreSearch.SearchResult, StoreSearch.SearchResult,
StoreSearch.SearchResult, StoreSearch.SearchResult,
StoreSearch.SearchResult, StoreSearch.SearchResult,
. . . ]
```

NM STATE

What if there is no network connection?

- Network Errors are among the many things that can come between you and a Network Request.
- Add a method to show an alert if there is a connection error so the user knows.

Work with the JSON Results

Work with JSON Results

- Successfully sent a network request to iTunes
- Successfully parsed JSON data into an array of SearchResults Objects
- Add Support for other iTunes Products
- Modify Data Model to include other iTunes products.

iTunes Products

- Songs, music videos, movies, TV Shows, podcasts
- Audio Books
- Software (apps)
- E- Books

NM STATE

Coding Keys

- CodingKeys enumeration
- Lets the Codable protocol know how you want the search properties matched to the JSON data.
- Provide a case for all properties in a class.

```
enum CodingKeys: String, CodingKey {
  case imageSmall = "artworkUrl60"
  case imageLarge = "artworkUrl100"
  case storeURL = "trackViewUrl"
  case genre = "primaryGenreName"
  case kind, artistName, trackName
  case trackPrice, currency
}
```

Different Data Structures

- Need to account for multiple data types such as audiobooks, music etc.
- Non-music results return no results such as authors like Stephen King.
- Need to add property declarations for all the iTunes products.

Showing the product type

Switch Statements

- iTunes has multiple products with which we can account for using a switch statement.
- Must have a case for all possible values.
- Defaults to “Unknown” should something not be there.
- Case statements do not automatically ”fall through” from one case to another as in Objective C.

```
var type:String {
  let kind = self.kind ?? "audiobook"
  switch kind {
  case "album": return "Album"
  case "audiobook": return "Audio Book"
  case "book": return "Book"
  case "ebook": return "E-Book"
  case "feature-movie": return "Movie"
  case "music-video": return "Music Video"
  case "podcast": return "Podcast"
  case "software": return "App"
  case "song": return "Song"
  case "tv-episode": return "TV Episode"
  default: break
  }
  return "Unknown"
}
```

NM STATE

Sort the Search Results

Sorting

- Sorting the results by alphabet or other metric increases readability.
- Swift Array already has a method to sort!

```
searchResults.sort(by: { result1, result2 in
  return result1.name.localizedStandardCompare(
         result2.name) == .orderedAscending
})
```

NM STATE

Improving the sorting code

- Trailing closures put the closure after the method name to improve readability.
- Operator overloading allows us to take standard operators (like + or *) and apply them to our own objects.
- Create a function named <
- < is our new sorting function outside the SearchResults.swift class.

Lecture 24 (continued):
Chapter 35: Asynchronous Networking

To Do List

- Extreme Synchronous Networking
- The Activity Indicator
- Make it Asynchronous

Extreme Synchronous Networking

Is Synchronous Networking Evil?

- Stress test the store search by slowing down the network connection.
- Adjust the amount of data an app receives by adding &limit= and any number to the end of the url string. (e.g &limit=200)
- Slow down the app by increasing the amount of data from a search result.

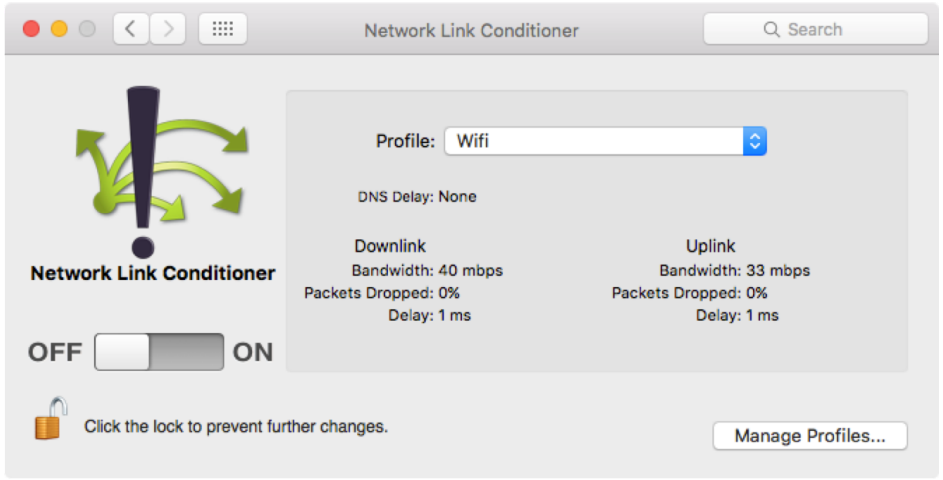
NM STATE

Simulate a Slow Connection

Slowing Down the Connection

- Additional Tools for Xcode available.
- Network Link Conditioner simulates connections.
- Demonstrates the dangers of synchronous networking

Network Link Conditioner



NM STATE

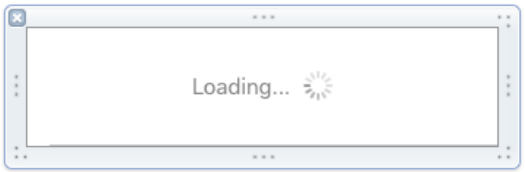
The Activity Indicator

The Activity Indicator

- We want to let the user know when the app is busy.
- The busy indicator should appear in the table cell.
- Add a nib file (.xib) for the situation when the app is busy.

Activity Indicator Table View Cell

- Create new empty nib file named LoadingCell.xib
- Add a label and activity indicator in storyboard
- Adjust the parameters to center and fit our custom table cell. The Result:



- Add to our TableViewCellIdentifiers structure
- Register in ViewDidLoad()

NM STATE

Update methods for LoadingCell

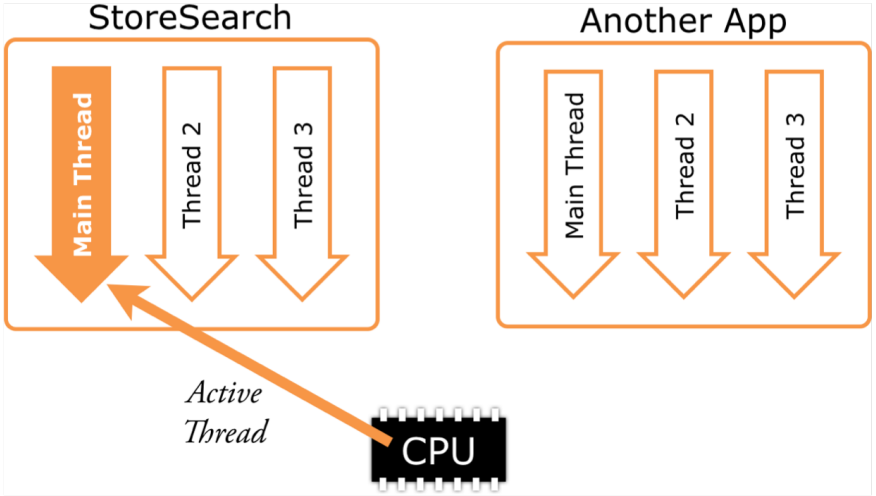
- Add a new Boolean instance variable isLoading
- Update numberOfRowsInSection to return the one row when isLoading is true.
- Update cellForRowAt to return an instance of the new LoadingCell with the animated spinner when isLoading is true
- Update willSelectRowAt to return a nil when the Loading Cell is active.
- Update searchBarSearchButtonClicked to set isLoading to true before the network request and reload the table to show the activity indicator.

Make it Asynchronous

CPU Threads

Synchronous Network effects

- By networking on the main thread, we disable the UI during the networking call.
- LoadingCell doesn't appear in this scenario because the network call will "stop" the app.
- Fix by making the network call in a background thread

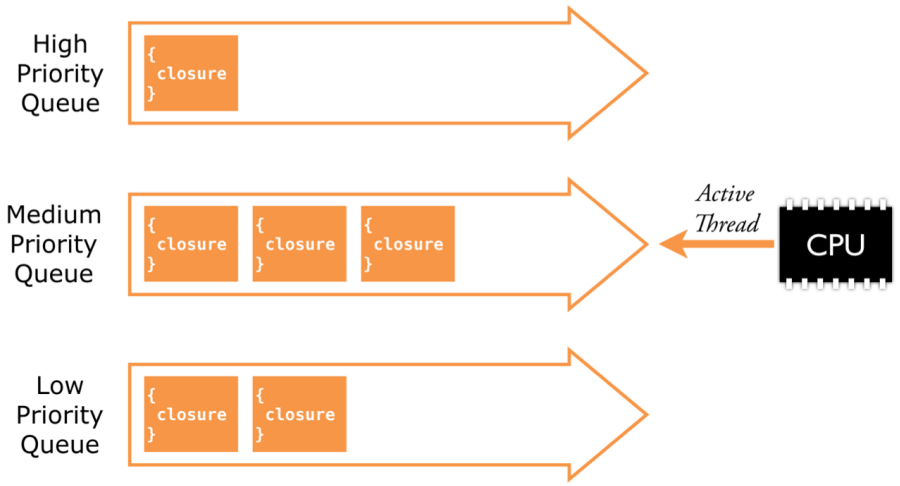


NM STATE

Grand Central Dispatch

Priority Queues

- Instead of making your own queue's iOS has more ways to start background processes.
- GCD has queues with different priorities
- Put job in a closure and pass to a queue.
- Queue's done in background one by one.



NM STATE

Put the web request in a background thread

Update searchBarButtonClicked

- Get a reference to the queue.
- Put our network call into a closure.
- Dispatch the closure with queue.async { code }

```
func searchBarSearchButtonClicked(_ searchBar: UISearchBar) {
  if !searchBar.text!.isEmpty {
    . . .
    searchResults = []
    // Replace all code after this with new code below
    // 1
    let queue = DispatchQueue.global()
    // 2
    queue.async {
      let url = self.iTunesURL(searchText: searchBar.text!)

      if let data = self.performStoreRequest(with: url) {
        self.searchResults = self.parse(data: data)
        self.searchResults.sort(by: <)
        // 3
        print("DONE!")
        return
      }
    }
  }
}
```

NM STATE

Put UI updates on the main thread

- Replace the print statement with another scheduled closure.
- New closure sets the loadingCell to false and returns the search results. Self is required because the code sits inside the closure.

GCD Queues

- Often follow a pattern:

```
let queue = DispatchQueue.global()
queue.async {
  // code that needs to run in the background

  DispatchQueue.main.async {
    // update the user interface
  }
}
```

NM STATE

The main thread checker

- Diagnostic setting which warns you about UI code running on background thread.
- Enabled in Edit Scheme from Xcode toolbar.
- Purple icon where the errors appear show us it’s a main thread UI warning.

Questions?