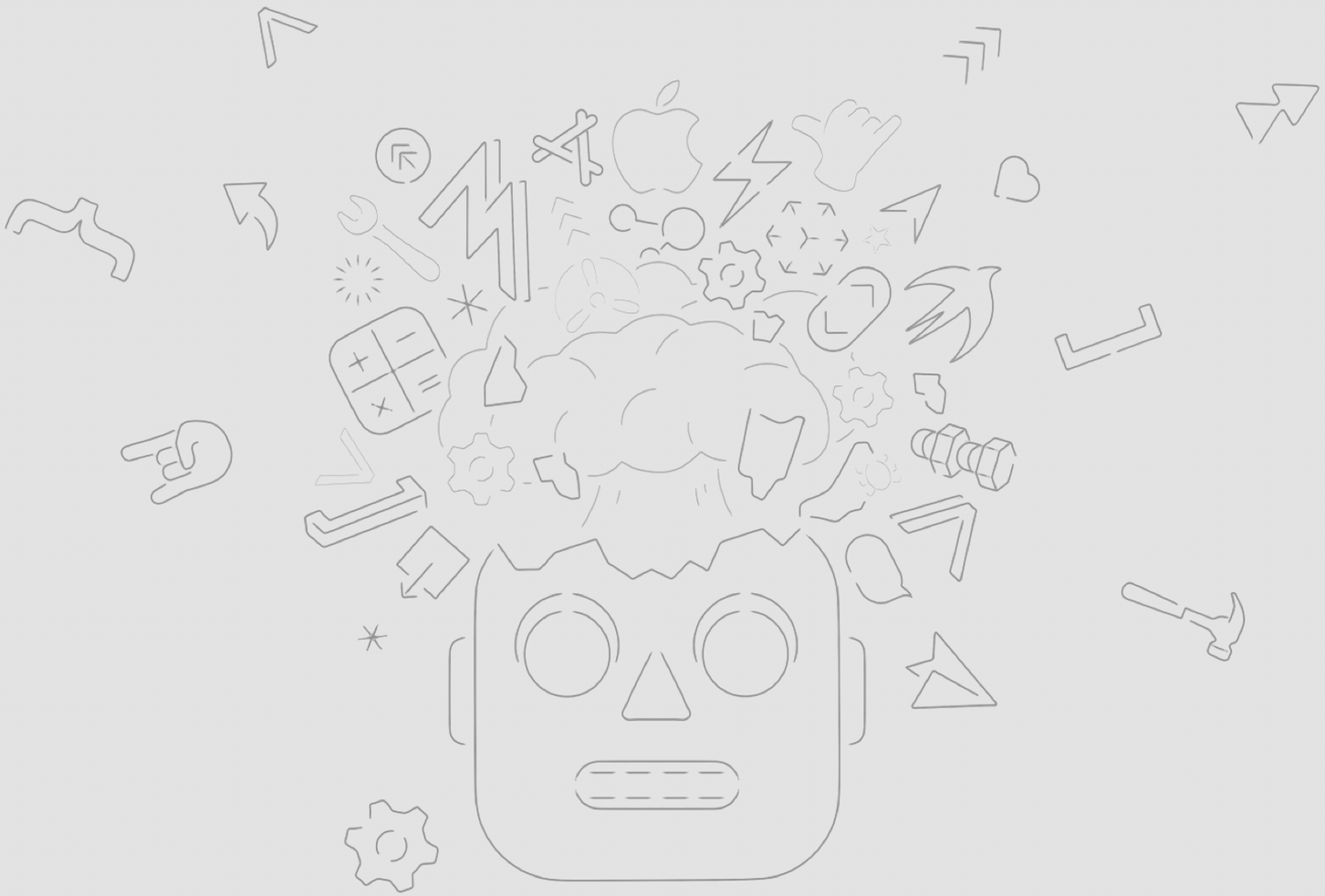
Lecture 9

Networking II



CIS 195-201 - Dominic Holmes

Today

- Review: DispatchQueue, JSON, Codable
- URLSession and HTTPS requests
- Using APIs in practice
- Live Demo

Semester Plan

- 2 more apps (1 assigned tonight, 1 assigned next week)
- 1 more tutorial
- Final Project
  - Starting with initial brainstorming next week

Swift 5.2 Released 🎉

- https://swift.org/blog/swift-5-2-released/
  - Better compiler errors, warnings, and code completion
- https://github.com/twostraws/whats-new-in-swift-5-2
  - Playground with the new code features
- Download **Xcode 11.4 (req. Catalina) to use Swift 5.2**

DispatchQueue

Main

```
DispatchQueue.main.async {
    // perform code on the main thread
    // asynchronously (NOT in order)
}
```

The docs:

- **DispatchQueue**
- **Main**
- **Async**

Main

```
DispatchQueue.global(qos: .background).async {

    // perform code on a background thread
    // still asynchronously (NOT in order)

}
```

The docs:
- **DispatchQueue**
- **Main**
- **Async**

JSON

JSON

- JavaScript Object Notation
  - The most common way to pass information around the Web and communicate with servers

JSON Examples

```
{
  "calendar": [
    {
      "end": "2020-03-15",
      "name": "Spring Term Break",
      "start": "2020-03-07"
    },
    {
      "end": "2020-03-16",
      "name": "Classes Resume",
      "start": "2020-03-16"
    }
  ]
}
```

```
{
  "document": {
    "venue": [
      {
        “id": 593,
        "dateHours": [
          {
            "date": "2019-10-20",
            "meal": [
              {
                "close": "15:00:00",
                "open": "11:00:00",
                "type": "Brunch"
              },
              {
                "close": "20:00:00",
                "open": "17:00:00",
                "type": "Dinner"
              }
            ]
          },
```

Open Object with “{”

Field of type Int

Open Array with “[”

Field of type String OR Date, depending how you look at it

Field of type Array<Object>

Field of type String. Could be defined as an Enum?

JSON Examples

- **api.pennlabs.org/dining/venues**
  - Returns a list of dining halls on campus
- **developer.nps.gov/api/v1/parks?api_key=<YOUR-API-KEY>**
  - Returns a list of national parks, with an api key passed into the header
- **https://api.pennlabs.org/laundry/halls**
  - Returns a list of laundry rooms

What is an API?

- "Application Programming Interface"
- From the national parks service API docs: *"An application programming interface (API) is a set of requirements that allows one application to talk to another."*
- Put simply: **it is a URL that you visit, which returns a JSON file *(or other data)***
- An essential component of any API is a defined *structure* for the data.
  - For example, the `/parks` route will always return a list of parks, and each park will always have a name and a id. In this case, '/parks' is an *endpoint*.
- APIs typically have different **endpoints** — each one returns a different set of information

Codable

Codable in a nutshell

- This is just a protocol, that you can choose to adopt or not
- This makes it **very easy** to convert data from JSON-format (or many other formats!) into Swift Structs
  - This is usually a difficult and error-prone process (manually decoding dictionaries). Swift makes it easy.
- `typealias Codable = Encodable & Decodable`

Codable Example

```
{
 "end": "2020-03-15",
 "name": "Spring Term Break",
 "start": "2020-03-07"
}
```

```
struct UniversityEvent {
    let end: String
    let name: String
    let start: String
}
```

Before Codable, we would create a dictionary from the json….
And then manually check each key.

Before Codable

```
{
 "end": "2020-03-15",
 "name": "Spring Term Break",
 "start": "2020-03-07"
}
```

```
struct UniversityEvent {
    let end: String
    let name: String
    let start: String
}
```

```
let myJSONDict: Dictionary<String, Any> = // convert JSON to dict...

if let end = myJSONDict["end"] as? String,
   let name = myJSONDict["name"] as? String,
    let start = myJSONDict["start"] as? String {

    let myEvent = UniversityEvent(end: end, name: name, start: start)
}
```

Before Codable, we would create a dictionary from the json….
And then manually check each key…

Before Codable: This is Hard!

```
let myJSONDict: Dictionary<String, Any> =
// convert JSON to dict...

if let end = myJSONDict["end"] as?
String,
   let name = myJSONDict["name"] as?
String,
    let start = myJSONDict["start"] as?
String {

    let myEvent = UniversityEvent(end:
end, name: name, start: start)
}
```

Decoding this into a dictionary in a TYPE SAFE way — not easy!


```
{
  "document": {
    "venue": [
      {
        “id": 593,
        "dateHours": [
          {
            "date": "2019-10-20",
            "meal": [
              {
                "close": "15:00:00",
                "open": "11:00:00",
                "type": "Brunch"
              },
              {
                "close": "20:00:00",
                "open": "17:00:00",
                "type": "Dinner"
              }
            ]
          },
```

So what does Codable do?

```
{
 "end": "2020-03-15",
 "name": "Spring Term Break",
 "start": "2020-03-07"
}
```

```swift
struct UniversityEvent: Codable
{
    let end: String
    let name: String
    let start: String
}
```

Just conforming to Codable lets us transform our struct into JSON!

Codable uses the variable name as the key, and the actual value as the value.

The key thing with Codable: **every nested type inside a Codable struct must ALSO be Codable.** Basic types like String and Int get this functionality for free.

So what does Codable do?

```
{
 "end": "2020-03-15",
 "name": "Spring Term Break",
 "start": "2020-03-07"
}
```

```swift
let myEvent = UniversityEvent(end: "10-21-20",
                              name: "Event Name",
                              start: "10-19-20")

// Convert to JSON
let json = try! JSONEncoder().encode(myEvent)
```

Codable Example

```
{
  "calendar": [
    {
      "end": "2020-03-15",
      "name": "Spring Term Break",
      "start": "2020-03-07"
    },
    {
      "end": "2020-03-16",
      "name": "Classes Resume",
      "start": "2020-03-16"
    }
  ]
}
```

```
struct Calendar: Codable {
    let calendar: [UniversityEvent]

    struct UniversityEvent: Codable {
        let end: String
        let name: String
        let start: String
    }
}
```

www.swiftjson.guide

The best Codable guide I've found. Especially needed for date behavior, CodingKeys, etc.

URLSession

How do we get data from a Server in Swift?

- Say we know that this API returns a valid json file: *https://api.github.com/users/DominicHolmes*
- How do we use that JSON's data in our swift code?

How do we get data from a Server in Swift?

https://api.github.com/users/DominicHolmes

1. Write a **Codable** representation of the JSON in Swift

2. Create a valid **URL** in Swift

3. Use **URLSession** to perform a HTTPS request from Swift

4. Decode the data from response into a Swift struct

5. Use the decoded response in our code. UI updates on main thread.

1. Write a Codable representation of the data in Swift.


```
{
  "login": "DominicHolmes",
  "id": 12633255,
  "node_id": "MDQ6VXNlcjEyNjMzMjU1",
  "avatar_url": "https://avatars1.githubusercontent.com/u/12633255?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/DominicHolmes",
  [. . . content omitted . . .]
  "name": "dominic",
  "company": "@pennlabs ",
  "blog": "dominic.land",
  "location": "philly",
  "email": null,
  "hireable": true,
  "bio": "iOS lead @pennlabs",
  "public_repos": 27,
  "public_gists": 0,
  "followers": 30,
  "following": 41,
  "created_at": "2015-05-27T21:17:26Z",
  "updated_at": "2020-03-18T01:56:02Z"
}
```

1. Write a Codable representation of the data in Swift.


```
{
  "login": "DominicHolmes",
  "id": 12633255,
  "node_id": "MDQ6VXNlcjEyNjMzMjU1",
  "avatar_url": "https://avatars1.githubusercontent.com/u/12633255?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/DominicHolmes",
  [. . . content omitted . . .]
  "name": "dominic",
  "company": "@pennlabs ",
  "blog": "dominic.land",
  "location": "philly",
  "email": null,
  "hireable": true,
  "bio": "iOS lead @pennlabs",
  "public_repos": 27,
  "public_gists": 0,
  "followers": 30,
  "following": 41,
  "created_at": "2015-05-27T21:17:26Z",
  "updated_at": "2020-03-18T01:56:02Z"
}
```

1. Write a Codable representation of the data in Swift.

```
{
  "avatar_url": "https://
avatars1.githubusercontent.com/u/12633255?v=4",
  "name": "dominic",
  "bio": "iOS lead @pennlabs",
  "followers": 30,
  "following": 41
  "created_at": "2015-05-27T21:17:26Z",
}
```

1. Write a Codable representation of the data in Swift.

```swift
struct GithubProfile: Codable {
    let name: String
    let bio: String
    let imageUrl: URL
    let createdAt: Date
    let followers: Int
    let following: Int

    enum CodingKeys: String, CodingKey {

        // Custom names for these 2 properties
        case imageUrl = "avatar_url"
        case createdAt = "created_at"

        // Default names for all the rest
        case name, bio, followers, following
    }
}
```

2. Create a valid URL in Swift

```swift
// Construct a URL by assigning its parts to a URLComponents value
var components = URLComponents()
components.scheme = "https"
components.host = "api.github.com"
components.path = "/users/dominicholmes"

// This will give us the constructed URL as an optional (URL?)
let url = components.url
```

2. Create a valid URL in Swift

```
// Construct a URL by assigning its parts to a URLComponents value
var components = URLComponents()
components.scheme = "https"
components.host = "api.github.com"
components.path = "/users/dominicholmes"

// This will give us the constructed URL as an optional (URL?)
let url = components.url
```

Or…

```
// Construct URL from a string (nil if string not a valid URL)
let url = URL(string: “https://api.github.com/users/dominicholmes")
```

3. Use URL session to perform the request from Swift

```
guard let url = url else {
    fatalError("url is nil")
}
```

3. Use URL session to perform the request from Swift

```
guard let url = url else {
    fatalError("url is nil")
}

// Define the request (task)
let task = URLSession.shared.dataTask(with: url) { data, response, error in

    // Notice the "{ _, _, _ in" means this is a closure
    // Thus, this closure is run when the request completes
    if error == nil {
        print("No errors!")
    }
}
```

3. Use URL session to perform the request from Swift

```
guard let url = url else {
    fatalError("url is nil")
}

// Define the request (task)
let task = URLSession.shared.dataTask(with: url) { data, response, error in

    // Notice the "{ _, _, _ in" means this is a closure
    // Thus, this closure is run when the request completes
    if error == nil {
        print("No errors!")
    }
}
```

3. Use URL session to perform the request from Swift

```swift
guard let url = url else {
    fatalError("url is nil")
}

// Define the request (task)
let task = URLSession.shared.dataTask(with: url) { data, response, error in

    // Notice the "{ _, _, _ in" means this is a closure
    // Thus, this closure is run when the request completes
    if error == nil {
        print("No errors!")
    }
}

// Actually starts the request!!! Very important line!
task.resume()
```

3. Use URL session to perform the request from Swift

```
guard let url = url else {
    fatalError("url is nil")
}

// Define the request (task)
let task = URLSession.shared.dataTask(with: url) { data, response, error in

    // Notice the "{ _, _, _ in" means this is a closure
    // Thus, this closure is run when the request completes
    if error == nil {
        print("No errors!")
    }
}

// Actually starts the request!!! Very important line!
task.resume()
```

3a. What are data, response, error?

From SwiftBySundell:

- `data` : Will either contain the bytes that were downloaded, or `nil` if an error occurred.
- `response` : A representation of the response that was received. Contains things like the MIME type and encoding of the downloaded data, and can be type casted to `HTTPURLResponse` (if we wish to get more HTTP-specific info, like the status code).
- `error` : Any error that was encountered, or `nil` if the operation was a success.

4. Decode the data from response into a Swift struct

```swift
// Try to decode the API response

let task = URLSession.shared.dataTask(with: url) { data, response, error in

    if error == nil, let data = data {

        if let profile = try? JSONDecoder().decode(GithubProfile.self, from: data) {
            dump(profile)
        } else {
            print("Decoding failed.")
        }

    }
}

task.resume()
```

4. Decode the data from response into a Swift struct

```
// Try to decode the API response

let task = URLSession.shared.dataTask(with: url) { data, response, error in

    if error == nil, let data = data {

        if let profile = try? JSONDecoder().decode(GithubProfile.self, from: data) {
            dump(profile)
        } else {
            print("Decoding failed.")
        }

    }
}

task.resume()
```

Prints: "Decoding failed."

4. Decode the data from response into a Swift struct

```
// Try to decode the API response

let task = URLSession.shared.dataTask(with: url) { data, response, error in
    if error == nil, let data = data {
        if let profile = try? JSONDecoder().decode(GithubProfile.self, from: data) {
            dump(profile)
        } else {
            print("Decoding failed.")
            try! JSONDecoder().decode(GithubProfile.self, from: data)
        }
    }
}
task.resume()
```

Add this line to see the issue

4. Decode the data from response into a Swift struct

```
Decoding failed.
Fatal error: 'try!' expression unexpectedly raised an
error: Swift.DecodingError.typeMismatch(Swift.Double,
Swift.DecodingError.Context(codingPath:
[CodingKeys(stringValue: "created_at", intValue: nil)],
debugDescription: "Expected to decode Double but found a
string/data instead.", underlyingError: nil)): file
networking.playground, line 46
```

What’s the issue?

4. Decode the data from response into a Swift struct

```
Decoding failed.
Fatal error: 'try!' expression unexpectedly raised an
error: Swift.DecodingError.typeMismatch(Swift.Double,
Swift.DecodingError.Context(codingPath:
[CodingKeys(stringValue: "created_at", intValue: nil)],
debugDescription: "Expected to decode Double but found a
string/data instead.", underlyingError: nil)): file
networking.playground, line 46
```

createdAt is our “Date” type. By default, Codable expects the date to be expressed as a Double, but ours is a string. These are just different Date formats — we can just tell the JSONDecoder which one to expect!

4. Decode the data from response into a Swift struct

```swift
// Try to decode the API response

let task = URLSession.shared.dataTask(with: url) { data, response, error in
    if error == nil, let data = data {

        let customDecoder = JSONDecoder()
        // Change the expected date format

        customDecoder.dateDecodingStrategy = .iso8601

        if let profile = try? customDecoder.decode(GithubProfile.self, from: data) {
            dump(profile)
        }
    }
}

task.resume()
```

4. Decode the data from response into a Swift struct

```swift
// Try to decode the API response

let task = URLSession.shared.dataTask(with: url) { data, response, error in
    if error == nil, let data = data {

        let customDecoder = JSONDecoder()
        // Change the expected date format

        customDecoder.dateDecodingStrategy = .iso8601

        if let profile = try? customDecoder.decode(GithubProfile.self, from: data) {
            dump(profile)
        }
    }
}

task.resume()
```

This works! It prints out the Profile:

```
▿ __lldb_expr_11.GithubProfile
  - name: "dominic"
  - bio: "iOS lead @pennlabs"
  ▿ imageUrl: https://avatars1.githubusercontent.com/u/12633255?v=4
    - _url: https://avatars1.githubusercontent.com/u/12633255?v=4 #0
      - super: NSObject
  ▿ createdAt: 2015-05-27 21:17:26 +0000
    - timeIntervalSinceReferenceDate: 454454246.0
  - followers: 30
  - following: 41
```

5. Use the decoded response in our code

We have to be careful because *URLSession performs its tasks on a background thread*.

Our (data, response, error) closure is thus *also called on a background thread***.**

So any UI we update from inside that closure must use a **DispatchQueue** to perform updates on the **main thread**.

5. Use DispatchQueue to make UI updates, based on the decoded response

```
let task = URLSession.shared.dataTask(with: url) { data, response, error in
    if error == nil, let data = data {
        let customDecoder = JSONDecoder()
        customDecoder.dateDecodingStrategy = .iso8601
        if let profile = try? customDecoder.decode(GithubProfile.self, from: data) {


            DispatchQueue.main.async {
                 self.myLabel.text = profile.bio
                 // other UI updates (ex. tableView.reloadData())
            }

        }
    }
}

task.resume()
```

This example is available on my Github:

https://github.com/DominicHolmes/cis-195-s20/blob/master/snippets/networking.swift

Swift Package Manager

SPM

- What if we want to use 3rd party code in iOS?
  - Examples: Sign in with Google, Kingfisher, RXSwift, Alamofire...
  - There's a massive ecosystem of 3rd party frameworks!! This is a great way to add tons of functionality without much effort.
- Swift Package Manager
  - Relatively new, built by Apple — makes adding package dependencies really easy
  - There are other older, more established package managers — namely **Cocoapods** and **Carthage.** We will use Cocoapods next week.

Live Demo: Trending Github Repos

https://github-trending-api.now.sh/repositories?language=swift&since=weekly

https://github.com/onevcat/Kingfisher

Due Before Next Class

- **App 6: Networking**

Links

- Piazza: **tiny.cc/cis195-piazza**