Stanford CS193p

Developing Applications for iOS

Spring 2020

Lecture 13

Today

- Persistence
  - Storing stuff between application launches ...
  - `UserDefaults`
  - Codable/JSON
  - `UIDocument` (UIKit feature worth mentioning)
  - Core Data
  - Cloud Kit
  - File system

Persistence

- UserDefaults
  Simple. Limited (Property Lists only). Small.
- Codable/JSON
  Clean way to turn almost any data structure into an interoperable/storable format.
- UIDocument
  Integrates the Files app and "user perceived documents" into your application.
  This is really the way to do things when you have a true document like EmojiArt has.
  Is UIKit-based (no SwiftUI interface to it yet) so UIKit compatibility code is required.
  Probably not something you'll use for your final project (since this is a SwiftUI course).
- Core Data
  Powerful. Object-Oriented. Elegant SwiftUI integration.

Persistence

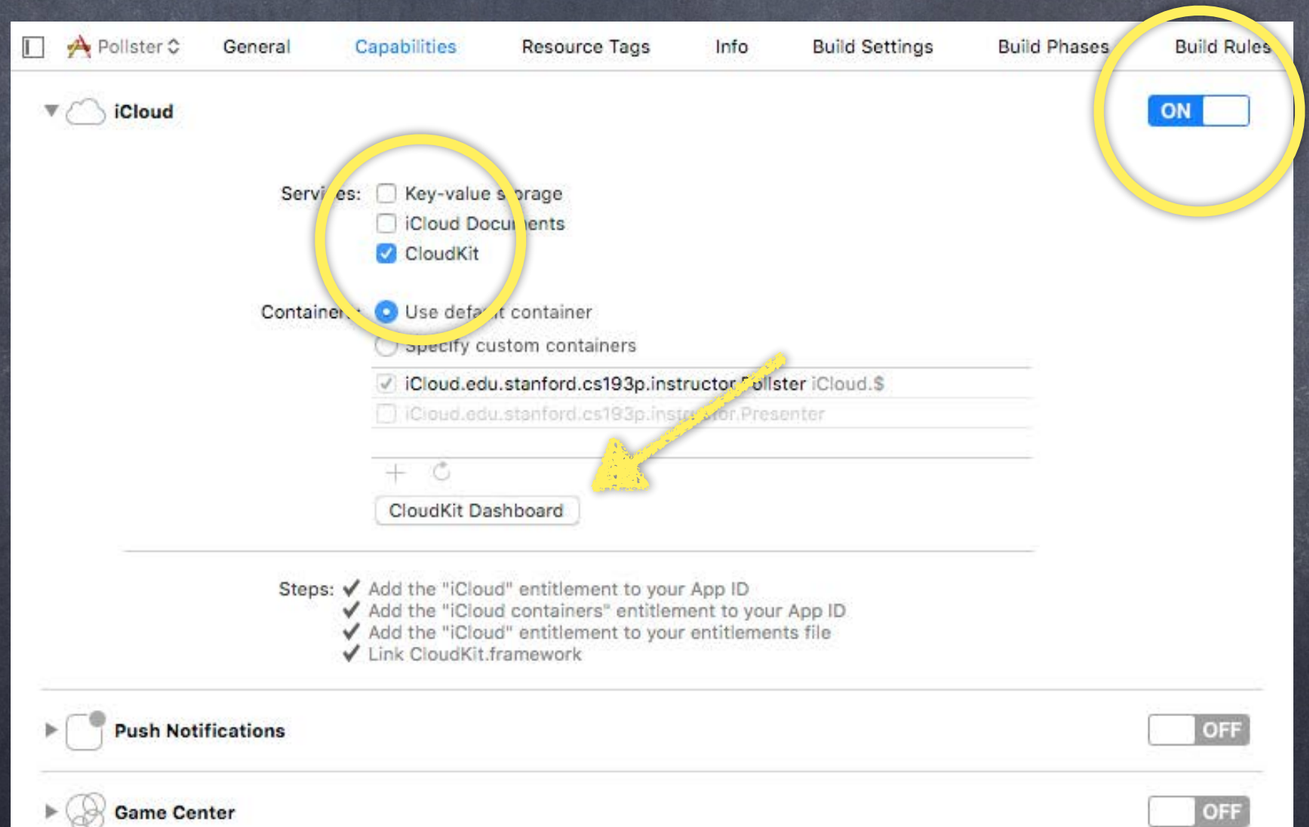
- Cloud Kit
  - Storing data into a database in the cloud (i.e. on the network).
  - That data thus appears on all of the user's devices.
  - Also has its own "networked `UserDefaults`-like thing".
  - And plays nicely with Core Data (so that your data in Core Data can appear on all devices).
  - We'll go over the basics of this via slides (insufficient time to do a demo though, sorry!).
  - An ambitious final project API to choose, but doable.
- `FileManager/URL/Data`
  - Storing things in the Unix file system that underlies iOS.
  - We'll demo this by storing EmojiArt documents in the file system instead of `UserDefaults`.

Cloud Kit

- Cloud Kit
  - A database in the cloud. Simple to use, but with very basic "database" operations.
  - Since it's on the network, accessing the database could be slow or even impossible.
  - This requires some thoughtful (i.e. asynchronous) programming.
  - No time for demo this quarter, but check Spring of 2015-16's iTunesU for full demo.
- Important Components
  - Record Type - like a `class` or `struct`
  - Fields - like `var`s in a `class` or `struct`
  - Record - an "instance" of a Record Type
  - Reference - a "pointer" to another Record
  - Database - a place where Records are stored
  - Zone - a sub-area of a Database
  - Container - collection of Databases
  - Query - a Database search
  - Subscription - a "standing Query" which sends push notifications when changes occur

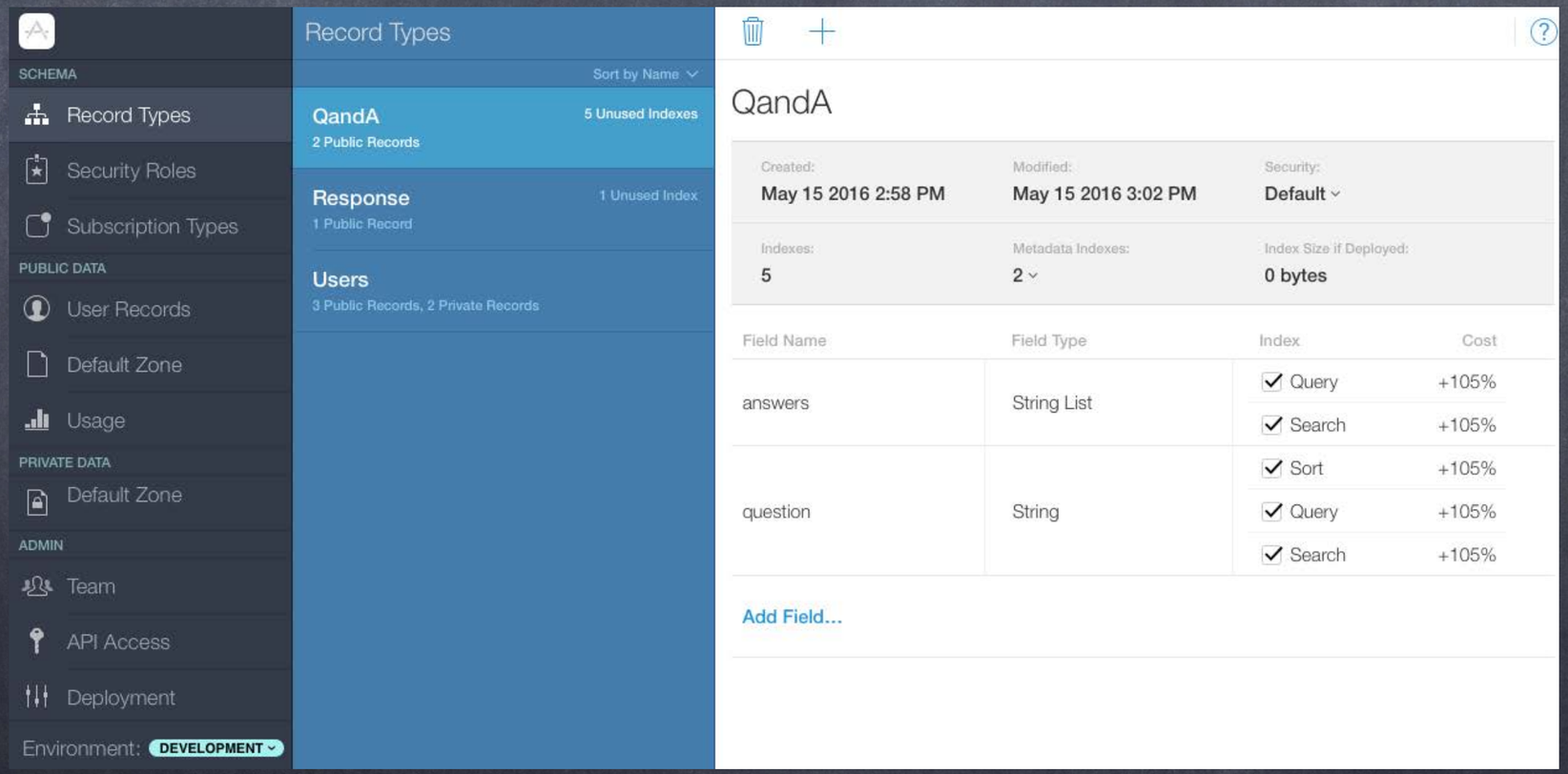
Cloud Kit

- You must enable iCloud in your Project Settings
  - Under Capabilities tab, turn on iCloud (On/Off switch).
  - Then, choose CloudKit from the Services.
  - You'll also see a CloudKit Dashboard button which will take you to the Cloud Kit Dashboard.



Cloud Kit

- Cloud Kit Dashboard
  - A web-based UI to look at everything you are storing.
  - Shows you all your Record Types and Fields as well as the data in Records.
  - You can add new Record Types and Fields and also turn on/off indexes for various Fields.



Cloud Kit

- Dynamic Schema Creation
  - But you don't have to create your schema in the Dashboard.
  - You can create it "organically" by simply creating and storing things in the database.
  - When you store a record with a new, never-before-seen Record Type, it will create that type.
  - Or if you add a Field to a Record, it will automatically create a Field for it in the database.
  - This only works during Development, not once you deploy to your users.

Cloud Kit

- What it looks like to create a record in a database

```
let db = CKContainer.default.public/shared/privateCloudDatabase
let tweet = CKRecord("Tweet")
tweet["text"] = "140 characters of pure joy"
let tweeter = CKRecord("TwitterUser")
tweet["tweeter"] = CKReference(record: tweeter, action: .deleteSelf)
db.save(tweet) { (savedRecord: CKRecord?, error: NSError?) -> Void in
    if error == nil {
        // hooray!
    } else if error?.errorCode == CKErrorCode.NotAuthenticated.rawValue {
        // tell user he or she has to be logged in to iCloud for this to work!
    } else {
        // report other errors (there are 29 different CKErrorCodes!)
    }
}
```

Cloud Kit

- What it looks like to query for records in a database

```
let predicate = NSPredicate(format: “text contains %@“, searchString)
let query = CKQuery(recordType: “Tweet”, predicate: predicate)
db.perform(query) { (records: [CKRecord]?, error: NSError?) in
    if error == nil {
        // records will be an array of matching CKRecords
    } else if error?.errorCode == CKErrorCode.NotAuthenticated.rawValue {
        // tell user he or she has to be logged in to iCloud for this to work!
    } else {
        // report other errors (there are 29 different CKErrorCodes!)
    }
}
```

Cloud Kit

- Standing Queries (aka Subscriptions)

  One of the coolest features of Cloud Kit is its ability to send push notifications on changes.

  All you do is register an `NSPredicate` and whenever the database changes to match it, boom!

  Unfortunately, we don't have time to discuss push notifications this quarter.

  If you're interested, check out the `UserNotifications` framework.

File System

- Your application sees iOS file system like a normal Unix filesystem
  - It starts at /.
  - There are file protections, of course, like normal Unix, so you can't see everything.
  - In fact, you can only read and write in your application's "sandbox".
- Why sandbox?
  - Security (so no one else can damage your application)
  - Privacy (so no other applications can view your application's data)
  - Cleanup (when you delete an application, everything it has ever written goes with it)
- So what's in this "sandbox"?
  - Application directory — Your executable, .jpgs, etc.; not writeable.
  - Documents directory — Permanent storage created by and always visible to the user.
  - Application Support directory — Permanent storage not seen directly by the user.
  - Caches directory — Store temporary files here (this is not backed up).
  - Other directories (see documentation) ...

File System

- Getting a path to these special sandbox directories

  `FileManager` (along with `URL`) is what you use to find out about what's in the file system.

  You can, for example, find the URL to these special system directories like this …

```
let url: URL = FileManager.default.url(
    for directory: FileManager.SearchPathDirectory.documentDirectory, // for example
    in domainMask: .userDomainMask  // always .userDomainMask on iOS
    appropriateFor: nil, // only meaningful for "replace" file operations
    create: true // whether to create the system directory if it doesn't already exist
)
```

- Examples of `SearchPathDirectory` values

  `.documentDirectory`, `.applicationSupportDirectory`, `.cachesDirectory`, etc.

URL

- Building on top of these system paths

  URL methods:

  ```
  func appendingPathComponent(String) -> URL
  func appendingPathExtension(String) -> URL  // e.g. "jpg"
  ```

- Finding out about what's at the other end of a URL

  ```
  var isFileURL: Bool  // is this a file URL (whether file exists or not) or something else?
  func resourceValues(for keys: [URLResourceKey]) throws -> [URLResourceKey:Any]?
  ```

  Example keys: `.creationDateKey`, `.isDirectoryKey`, `.fileSizeKey`

File System

- Data

  Reading binary data from a URL …

  ```
  init(contentsOf: URL, options: Data.ReadingOptions) throws
  ```

  The options are almost always `[]`.

  Notice that this function `throws`.

  Writing binary data to a URL …

  ```
  func write(to url: URL, options: Data.WritingOptions) throws -> Bool
  ```

  The `options` can be things like `.atomic` (write to tmp file, then swap) or `.withoutOverwriting`.

  Notice that this function `throws`.

File System

- `FileManager`
  - Provides utility operations.
  - e.g., `fileExists(atPath: String) -> Bool`
  - Can also create and enumerate directories; move, copy, delete files; etc.
  - Thread safe (as long as a given instance is only ever used in one thread).
  - Also has a `delegate` you can set which will have functions called on it when things happen.
  - And plenty more. Check out the documentation. And check out the demo ...

Demo

- EmojiArt Documents
  - Probably the best way to store EmojiArt documents would be using `UIDocument`.
  - That's a UIKit thing and beyond the scope of this SwiftUI course
  - But let's at least take a look at storing EmojiArt documents in the file system.
  - (Which will be much more reasonable than `UserDefaults`!)