Advanced Persistence
CMSC 436
CloudKit

Use with Core Data

Hosts your data in iCloud

Everything handled for you transparently!

Code changes: in Persistence.swift, change NSPersistentContainer to NSPersistentCloudKitContainer

Requires an Apple Developer account

We’re not going to bother with this
CocoaPods

Dependency manager

Written in Ruby

```bash
sudo gem install cocoapods
or
brew install cocoapods
```

Create a Podfile in your project

```bash
- cd my_project_dir
- pod init
```

Edit Podfile to add dependencies

```bash
pod install
```

When you open the project in Xcode, select the `.xcworkspace`, not the `.xcodeproj`!
Go ahead and use this (changing Location to whatever your project is called), since it sets up our next topic...

platform :ios, '10.0'

target 'Location' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # Add the Firebase pod for Google Analytics
  pod 'Firebase/Analytics'

  # Add the pods for any other Firebase products you want to use in your app
  # For example, to use Firebase Authentication and Realtime Database
  pod 'Firebase/Auth'
  pod 'Firebase/Database'

end
Firebase

NoSQL database (loosely structured data)

Another fine Google™ product

JSON object trees

Concurrency control

We will just scratch the surface
Types of Firebase Databases

Realtime Database and Cloud Firestore

Realtime Databases (what we’ll use) support only

- Simple data models
- One JSON tree
- Filtering or sorting, not both, in a single operation
- Only one geo-region (no geographically distributed replication of data)
Setting up an Account

You’ll need to have a Google account

Go to https://firebase.google.com/

Please note:

▶ You will be providing Google with your data
▶ You will be providing Google with your users’ data
▶ This is the same agreement you make with any cloud provider
▶ As of March 2021, there is still a lawsuit in progress over Google’s handling of private data, when tracking was explicitly disabled

Let’s go!
Setting Up a Firebase Account

See the video
JSON Trees

You will probably be creating trees of Codable types

JSON supports key-value mappings

Values may be

- Simple types (numbers, strings)
- Arrays of values (in `[]`)
- Dictionaries of key-value pairs (in `{}`)

Example:

```json
[
  {
    "name": "Alf",
    "source": "Tubi",
    "watching": false
  }
]
```
Optimizing Your Data Model for Firebase Realtime

Nested dictionaries create *object trees*

Selecting a node fetches the *entire* subtree rooted at that node

Make your data *shallow* and *wide*

Each fetch is relatively small
Connecting Your Project to Firebase

Go to https://console.firebase.google.com

Create a project

Select “iOS” app configuration

Copy the GoogleService-Info.plist file to your project
Making the Connection

See the video
Working with the Firebase JSON Tree

In your app:

- import Firebase
- Call FirebaseApp.configure() in the App’s init()

In your model:

- import FirebaseDatabase
- let rootRef = Database.database().reference() gets the root of the JSON tree
- let itemsRef = rootRef.child("items") gets child by key
- rootRef.childByAutoId() creates a child with a unique key
- itemsRef.setValue(val) writes to Firebase; val must be NSString, NSNumber, NSDictionary, or NSArray
- getData, observe, observeSingleEvent are a few of the ways to retrieve data (asynchronously)
A Data Type

```swift
struct DirEntry: Codable, Identifiable {
    var id: String?
    var name: String
    var phone: String
    var email: String

    var dict: NSDictionary? {
        if let idStr = id {
            let d = NSDictionary(dictionary: [
                "id": idStr, "name": name, "phone": phone, "email": email ])
            return d
        } return nil
    }

    static func fromDict(_ d: NSDictionary) -> DirEntry? {
        guard let name = d["name"] as? String else { return nil }
        guard let phone = d["phone"] as? String else { return nil }
        guard let email = d["email"] as? String else { return nil }
        return DirEntry(id: d["id"] as? String, name: name, phone: phone, email: email)
    }
}
```
Our Observable Object (1/2)

class Directory: ObservableObject {
    @Published var entries: [String:DirEntry] = [:]

    init() {
        let rootRef = Database.database().reference()
        rootRef.getData { err, snapshot in
            DispatchQueue.main.async {
                for child in snapshot.children {
                    if let item = child as? DataSnapshot {
                        if let val = item.value as? NSDictionary,
                            let de = DirEntry.fromDict(val),
                            let id = de.id { self.entries[id] = de }
                    }
                }
            }
        }
        rootRef.observe(.childAdded) { snapshot in
            if let v = snapshot.value as? NSDictionary,
                let de = DirEntry.fromDict(v),
                let id = de.id { self.entries[id] = de }
        }
        rootRef.observe(.childRemoved) { snapshot in
            self.entries.removeValue(forKey: snapshot.key)
        }
        rootRef.observe(.childChanged) { snapshot in
            if let v = snapshot.value as? NSDictionary,
                let de = DirEntry.fromDict(v),
                let id = de.id { self.entries[id] = de }
        }
    }
}
func addEntry(entry: inout DirEntry) {
    let rootRef = Database.database().reference()
    let childRef = rootRef.childByAutoId()
    entry.id = childRef.key
    if let val = entry.dict {
        childRef.setValue(val)
    }
}

func delEntry(id: String) {
    entries.removeValue(forKey: id)
    let rootRef = Database.database().reference()
    rootRef.child(id).removeValue()
}

We’ll use these in the View, as you’ll see in the demo!