framework training
We love technology

Building iOS Apps in Swift

Course notes for part 2

19th-21st September 2021

020 3137 3920 @FrameworkTrain

frameworktraining.co.uk

Contents List

Day 4 - 19th September 2021

Day 5 - 20th September 2021

Day 6 - 21st September 2021

Session 15: Data Storage with Codable

- Application sandbox
- Where to save data in iOS
- Storage choices
- Codable for local state
- *Lab 15: Extended lab bringing together work on tables, data access and storing data locally.*

- This covers lesson 1.7 of Development in Swift Data Collections

Common data patterns in our apps

- We enter data and save it - examples: to-do list, shopping list, blog
- We enter data and save it to the cloud - examples: questionnaires, meter readings
- We use existing local data - examples: our conference app, language learning
- We use cloud-based data - examples: power plant readings, conference app if the data changed over time

Where do we save data in our app?

- Application sandbox - Your App runs in a protected area on the device
- Within this area, there is a defined structure of Directories that you can access
- Where you find what:
- <App Home>/AppName.app - the default Bundle directory
- <App Home>/Documents - backed up to iCloud potentially
- <App Home>/Library/Preferences
- <App Home>/Library/Caches
- <App Home>/tmp
- For simulator you can find <App Home> by printing out its URL

3

What format do we save data in our app?

- Several different popular techniques for storing and retrieving data on the iPhone:
- JSON - read and write
- Property List Serialization - Mechanism to store a selected number of data types in a property list (plist)
- SQLite - small memory footprint SQL database
- Core Data - more complex system, but it does offer a powerful way to manage data and relationships

4

What about XML?

- XML is supported in iOS but is not a popular choice of structure - especially since JSON parsing has become so easy
- XML can be read from storage in the same way as we will see for JSON
- You can then call XMLParser with associated parse objects to interpret the XML
- The following tutorial gives an example of doing this:
  - https://www.ioscreator.com/tutorials/parse-xml-ios-tutorial

JSON easiest for simple data

- Codable makes it trivial for flat file records that are made up of data types that are themselves Codable (Int, Double, String, Array, Date etc.)
- More work if you have the kind of relationships in our Conference Data - in those cases you might use Core Data to persist the relationships

Case Study: Creating Speaker Data

- How do we get all the data into our Conference app? At present, we do it by initialising lots of data objects.
- If we encode the data as JSON, we can save it to a file, and then read the file and decode it into data objects in our conference app
- To do the data creation, we could make a ConferenceCreator app (but to simplify things we will just create Speaker data for now)

7

Step 1 (repeat previous things)

- Make new iOS app called ConferenceCreator
- Delete ViewController and screen
- Add NavigationController from Object library - it comes with a built-in tableviewcontroller
- Make the NavigationController into the initial view controller
- Give the table cell in the tableview controller style Basic, and identifier SpeakerCell

8

Step 2 - add in provided files

- Add the provided files to the project
  - Speaker.swift is the struct for a Speaker
  - ConferenceData is a singleton for accessing the list of speakers - we will add to it as we go on

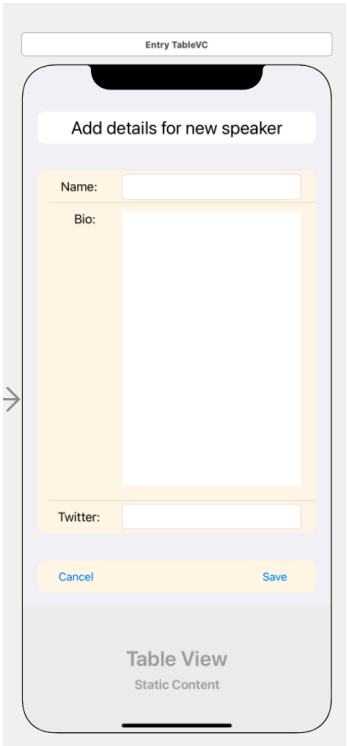
Step 3 - make code for speaker table

- Do File/New/File and create a tableview controller code file called SpeakerTableVC, and associate it with the table view controller
- Fill in the code to populate the table view controller - make a speaker list called speakers and set it:
  - var speakers = ConferenceData.sharedInstance.allSpeakers
- Have one section in the table, with speakers.count cells, and the speaker name in each cell
- Run now, app should show list of (two) speakers

Step 4 - link to another tableview controller

- Add a second table view controller to the main storyboard
- Give the table view static cells, with three sections, and make its style inset grouped
- Set items as shown on next slide

Layout for static tableview controller



First section - one cell, with a label acting as a header

Second section - three cells making up an entry form

First cell has a label “Name:” and a TextField. Set the Textfield options to input a Name, with words capitalised and no spell checking or correction.

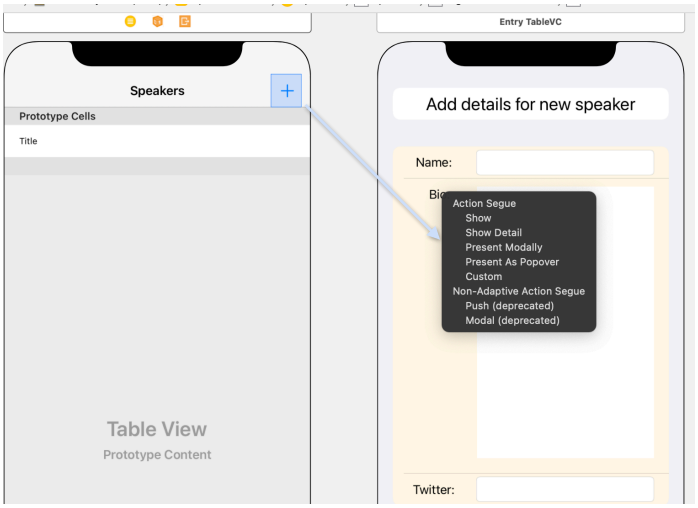
Second cell has a label “Bio” and a TextView for inputting more that one line. Size of cell expanded to allow for lots of text.

Third cell has label Twitter and a TextField

Third section contains two buttons

Step 5 - Link SpeakerTableVC static table

- Add bar button to SpeakerTableVC, and make it a system Add button
- Control-Drag from the Add button to the static table and make the relationship *Presents Modally,* then click on the segue and make the presentation full screen



Step 6 - Code for static table

- We need to make a tableview controller file to associate with the static table - call it SpeakerEntryTableVC
- As the table is static, we do not need to populate it, so can delete all the boiler plate code except viewDidLoad
- We do need to set up outlets for the input fields and actions for the buttons
- We treat them just as we would in a simple viewController

Step 7 - Setting up outputs and save action

- Click and drag from the storyboard to make outputs for speakerName, speakerBio and twitterHandle
- Click and drag from Save button to make action to saveEntry

```
7
8  import UIKit
9
10 class EntryTableVC: UITableViewController {
11
12
13     @IBOutlet weak var speakerName: UITextField!
14
15     @IBOutlet weak var speakerBio: UITextView!
16
17     @IBOutlet weak var twitterHandle: UITextField!
18
19     @IBAction func saveEntry(_ sender: Any) {
20
21     }
22
```

15

Step 8 - Add code to saveEntry

```
@IBAction func saveEntry(_ sender: Any) {
    // If name is blank, give an alert
    // else add speaker to speaker list and return to main speaker table
    if let name = speakerName.text,
             let bio = speakerBio.text, let twitter = twitterHandle.text {
      let id = name.trimmingCharacters(in: .whitespacesAndNewlines)
      if id == "" {
           let alert = UIAlertController( title: "Sorry",
                           message: "You need to add a name to save a speaker",
                           preferredStyle: .alert)
           alert.addAction(UIAlertAction(title: “OK",
                           style: .default, handler: nil))
           self.present(alert, animated: true, completion: nil)
      } else {
         ConferenceData.sharedInstance.addSpeaker(newSpeaker:
                  Speaker(id: id, name: name, biography: bio, twitter: twitter))
         dismiss(animated: true, completion: nil)
      }
    }
}
```

16

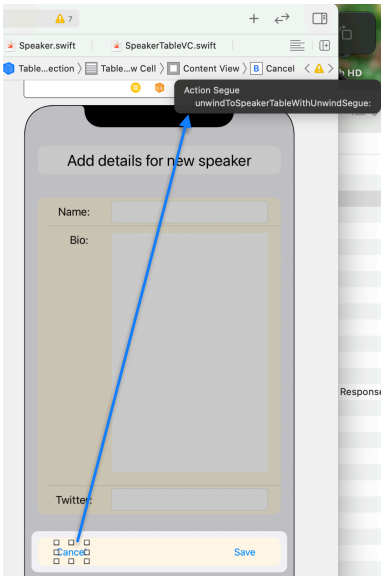
Step 9 - Make the Cancel button return without doing anything

- Add unWindSegue code to SpeakerTableVC
- In the main story board, click drag from the Save button to the top of the screen to link to the unwind segue

Code to put in SpeakerTableVC:

```
@IBAction func unwindToSpeakerTable(
                    unwindSegue: UIStoryboardSegue) {
}
```

Action to link unwind:



17

Step 10 - Run the app. We can add a speaker, but speaker list does not update - why?

- In code for SpeakerTableVC, the speaker list is set up at the start and not changed when we update the list
- We need to check it and redraw the list whenever we come back to SpeakerTableVC
- Best place to do this is in viewWillAppear:

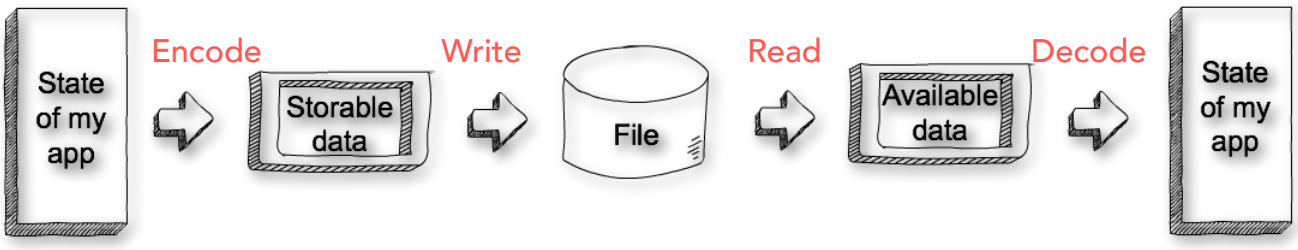
```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    speakers = ConferenceData.sharedInstance.allSpeakers.sorted()
    self.tableView.reloadData()
}
```

18

Run the app - the list updates, but if we start it again, it goes back to the two original items

- We need to save the list when we update it, and then read it back in when we start up
- Saving is a two stage process - we encode it as JSON, then we write it to the Documents directory

Common workflow



Step 11: Some useful boilerplate file handling code to add to ConferenceData

```
func urlToFileInDocuments( _ fileName: String ) -> URL {
// This function returns the address of the given file in Documents
    let docDirectory = FileManager.default.urls(for: .documentDirectory,
                                                in: .userDomainMask).first!
    let fileURL = docDirectory.appendingPathComponent(fileName)
    return fileURL
}


func fileExistsInDocuments( _ fileName: String ) -> Bool {
// This function tells you whether the named file already exists in Documents
    let fileManager = FileManager.default
    let dirPaths = NSSearchPathForDirectoriesInDomains(
                                    .documentDirectory, .userDomainMask, true)
    let docsDir = dirPaths[0]
    let filepathName = docsDir + "/\(fileName)"
    return fileManager.fileExists(atPath: filepathName)
}
```


Step 12 - Add new method to ConferenceData and call it

```
func saveSpeakers() {
        let readListURL =  urlToFileInDocuments("speakers.json")
        let encoder = JSONEncoder()
        if let data = try? encoder.encode(allSpeakers) {
            //Write the data to backing store.
            try? data.write(to: readListURL, options: .noFileProtection)
            print( "file is at \(readListURL)")
        }
    }
```

- The code above encodes allSpeakers as JSON and then writes it to file speakers.json
- It gives an error as the Speaker struct is not Codable, but as all its attributes are Codable, we just need to add Codable to the header of the Speaker struct
- We can then call saveSpeakers whenever we call method addSpeaker in ConferenceData

Step 13 - Run the app - the console will tell you where speakers.json is saved by the simulator

- If we find that directory (in Finder you choose Go/Go to Folder and provide the folder name), we will see the speakers.json file and can look at its contents
- However, if we restart the app, we will be back to seeing just two speakers, as we do not read back in the list we saved - so we need to do that
- We need to read the list in and decode it when we first open the app (we need to check it exists first, as it won't exist when we open the app the first time)

23

Step 14 - Give ConferenceData an initialiser that reads the saved file if it exists

- We add the following to ConferenceData
- When the singleton starts, it will look for the speakers.json file on disc, and read it into an array of Speaker objects

```
init(){
    if fileExistsInDocuments("speakers.json") {
        let readListURL =  urlToFileInDocuments("speakers.json")
        if let dataFromFile = try? Data(contentsOf: readListURL) {
            // Decode the plist back to state of program
            let decoder = JSONDecoder()
            if let loadedArray = try? decoder.decode([Speaker].self,
                                        from: dataFromFile) {
                allSpeakers = loadedArray
            }
        }
    }
}
```


24

Comments / expanding this

- We now have an app which creates a list of speakers and saves it
- If we wanted to use this to make all the conference data, we would have arrays of talks and locations and then add all three things into a struct which included the separate arrays
- It would be Codable and can be written /read in one go
- We can add the created JSON file into our previous Conference app with code that reads the data and it should work as it did before

25

Optional exercise

- We have provided a new version of ConferenceData, and a JSON file called confinfo.json with the same data as before
- Try adding it into the Conference app you made in session 11 during the first three days of the course

26

Session 16: Unit tests in Xcode

- Identifying what to unit test
- Setting up tests
- Running tests in XCTest
- *Lab 16: Writing Unit Tests for iOS Apps*

- This material is not covered in the Develop in Swift books

1

Automated Testing in iOS

- Xcode provides templates both for Unit Tests and for UI Testing
- Best strategy is to include them when you start a new project
- They can be added to your project by hand later on, by choosing New/File/Target and then selecting Unit Tests or UI Tests to add to your app
- We will make a new app including testing, and then write tests for a cube function

2

What do we test in Unit testing

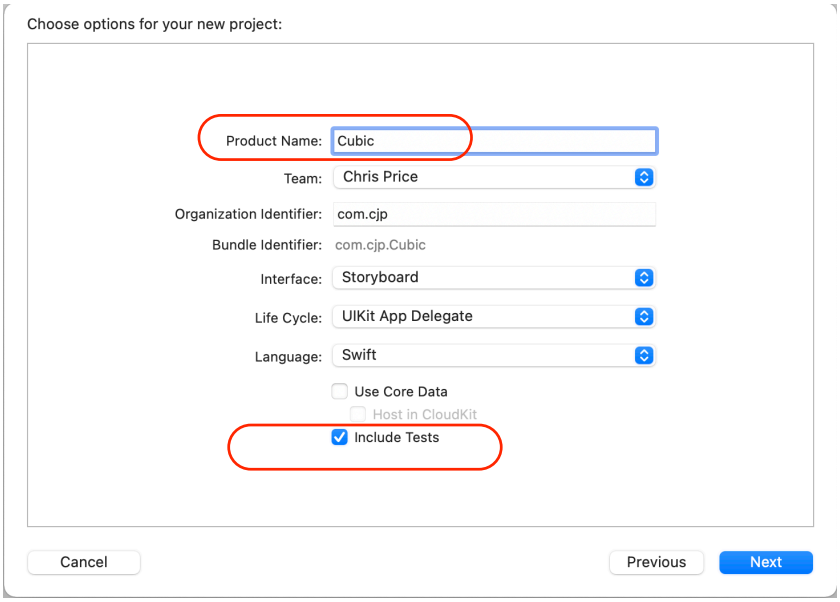
- Unit Tests are appropriate for the back end of your system
  - For example, testing that a cube function returns the correct answer for good values
  - For example, checking that photographs exist for all speakers at the conference

Lets look at easy test situation first

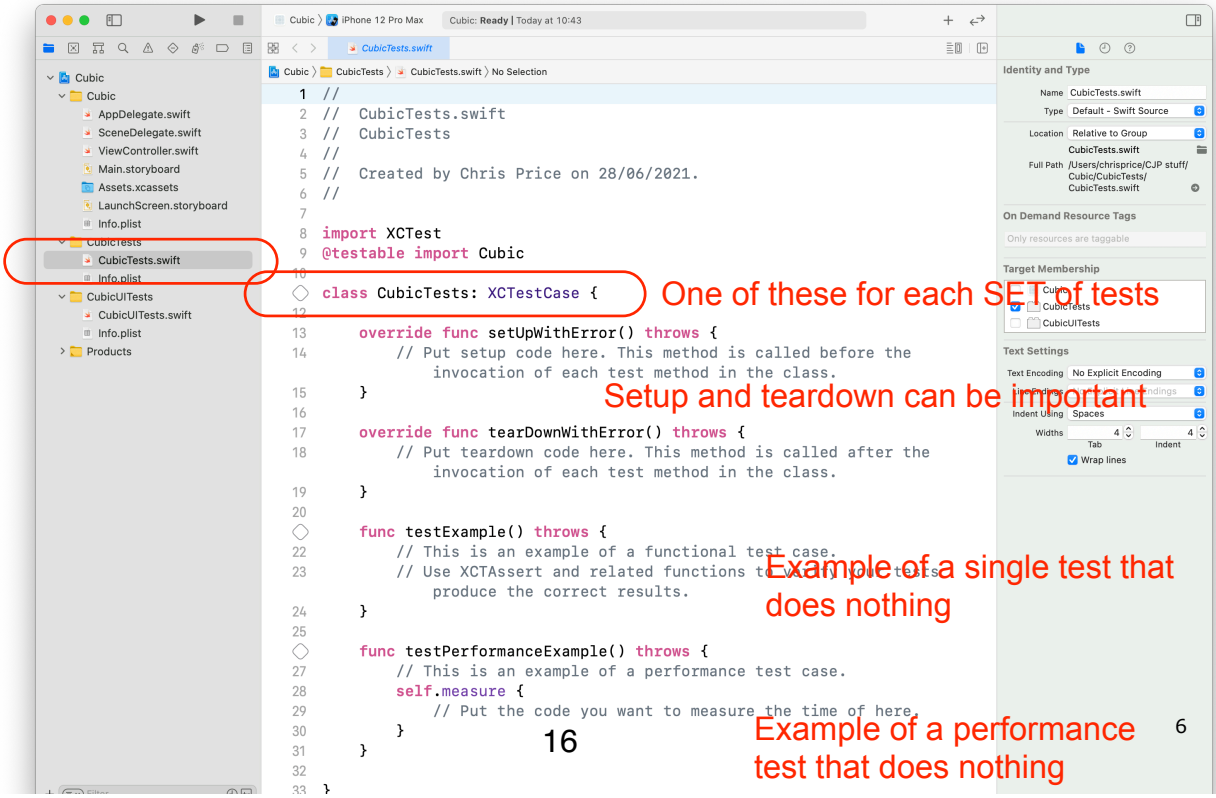
- We will create a program with a math function that returns the cube of a number
- We will write tests to check the function works

Step 1 of testing cube function

- Make new iOS app called Cubic - when choosing options, tick to include testing



Draft template for Unit Testing is provided in CubicTests.swift



Step 2 of testing cube function

- Create Swift file called DoMaths.swift and populate it with a DoMaths class with a stub function for cube as follows:

```
class DoMaths {

    func cube(_ number: Int) -> Int {
        return -1
    }

}
```

7

Step 3 - write some tests

- In class CubicTests, delete the boilerplate tests and add some real tests:

```
func testCubeOfZero() throws {
    let maths = DoMaths()
    XCTAssertEqual(maths.cube(0), 0)
}

func testCubeOfOne() throws {
    let maths = DoMaths()
    XCTAssertEqual(maths.cube(1), 1)
}

func testCubeOf10() throws {
    let maths = DoMaths()
    XCTAssertEqual(maths.cube(10), 1000)
}
```

8

Running the tests

- In CubicTests, you will see little diamonds next to the class declaration and each test, to run all tests, either click the class diamond, or type CMD-u

Fixing the failures

- Update cube as shown and rerun the tests and they should pass

```
class DoMaths {

    func cube(_ number: Int) -> Int {
        return number * number * number
    }

}
```

Using setup

- As all our tests need *maths* set up, we can do that in setup and simplify all tests:

```
class CubicTests: XCTestCase {

    var maths: DoMaths!

    override func setUpWithError() throws {
        maths = DoMaths()
    }

    func testCubeOfZero() throws {
        XCTAssertEqual(maths.cube(0), 0)
    }

    func testCubeOfOne() throws {
        XCTAssertEqual(maths.cube(1), 1)
    }

    func testCubeOf10() throws {
        XCTAssertEqual(maths.cube(10), 1000)
    }
```

11

Adding performance tests

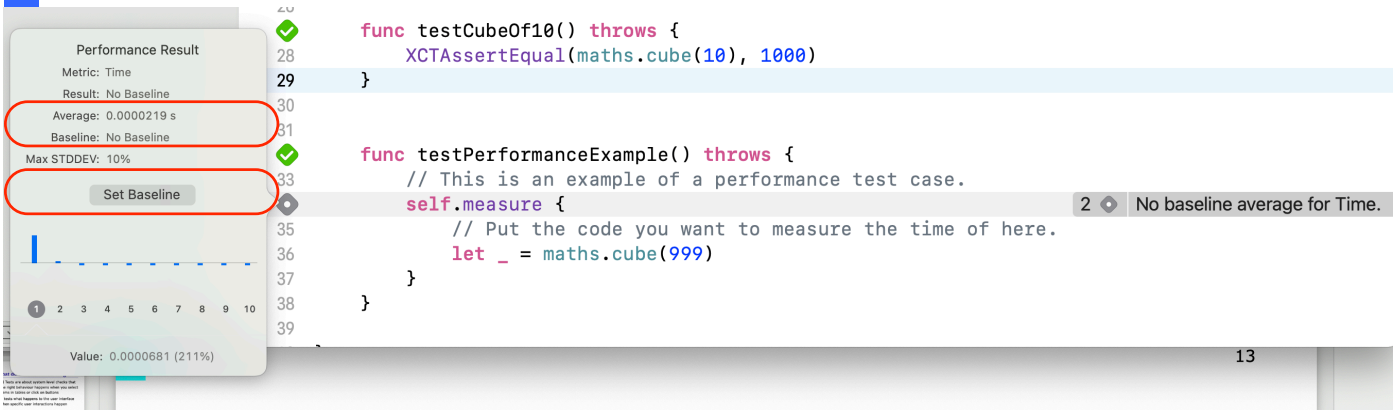
- We can take Apple's example performance test and use it to assess our cube function
- When we run this test, we get a grey diamond next to the measurement, and can look at the value

```
func testPerformanceCube() throws {
    self.measure {
        let _ = maths.cube(999)
    }
}
```

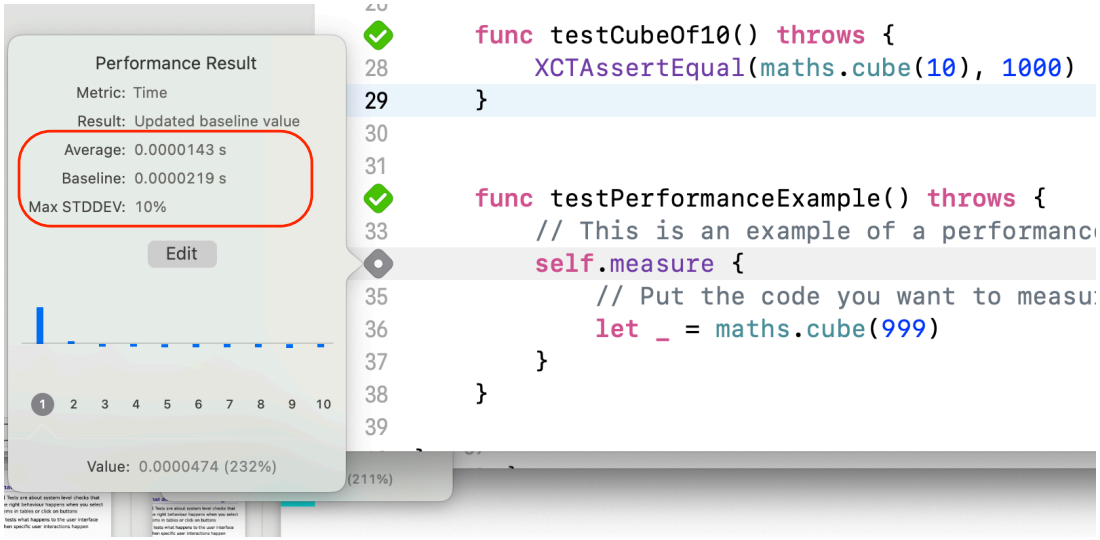

12

Noting performance results

- Clicking on diamond shows timing result
- We can click on Set Baseline to be able to compare future results



Run again to see if changed



Replacing cube

- If we replace cube with the version below, the unit tests will still pass, but performance is shown much worse

```
func cube(_ number: Int) -> Int {
    var answer = 0
    if number == 0 {return number}
    for _ in 1...number {
        answer = answer + number
    }
    return answer * number
}
```

```
31
    func testPerformanceExample() throws {
33      // This is an example of a performance test case.
        self.measure {                                   Time: 0.000 sec (1551% worse)
35          // Put the code you want to measure the time of here.
36          let _ = maths.cube(999)
37      }
38  }
```

Try adding tests to Conference app with JSON

- We want to look at each of the speakers, and check that the photograph exists for that person
- We will need to add unit testing to the conference app that uses JSON and then go through each speaker and check that the speaker photo exists
- Use the conference app with JSON included in this session

What is needed

- Select File/New/Target, and under iOS, select Unit Testing Bundle
- Need to add *@testable import Conference* at the top of the new tests so everything gets imported
- Write a test that iterates through all the speakers and fails if the following test fails for any of them:
  - if let image = UIImage(named: speaker.id)

17

UI testing in Xcode

- UI Tests are about system level checks that the right behaviour happens when you select items in tables or click on buttons
- It tests what happens to the user interface when specific user interactions happen

What do we test in UI testing

- UI Tests are about system level checks that the right behaviour happens when you select items in tables or click on buttons
- It tests what happens to the user interface when specific user interactions happen

Different from unit tests

- Unit tests can access variables and code within your app
- UI tests can only interface with an app in the way that the user can
- UI tests can only check the state of the user interface after interactions, NOT the state of variables

3

Theory

- If you click in a UI test, you will see a red record button. Press it and it will start recording your actions
- You can assert what the UI should look like AFTER your actions
- It can then replay your actions
- If the assertion is not true, the test fails

4

Experience

- It does not seem to work very well
- Sometimes the code generated to reproduce the actions is bad
- Sometimes no code is generated
- Really interesting technology but not ready for serious use

Session 17: Data Storage with Core Data

- Ideas behind Core Data
- Persisting networks of objects
- *Lab 17: Replacing Codable with Core Data*

- This material is not covered in the Develop in Swift books

What is Core Data?

- Core Data is Apple's persistent storage that makes it easy to save and restore a set of objects in memory
- It is especially good if you want to use the same set of objects on different computers - you can link you data model to Cloudkit
- You can also store data in iCloud that all your users can share

How does it work?

- You specify a set of entities and relationships for your data, and Core Data defines the coding data structures (e.g. the Speaker struct that we have been using in the Conference app)
- You can then create and save such structures - they are saved to a database file (you can choose the underlying format, but default is an SQL database)
- When you start up your app, you can read back in the set of objects that you were dealing with
- We will look at an example with a set of simple objects we have seen before - creating a list of Conference speakers

3

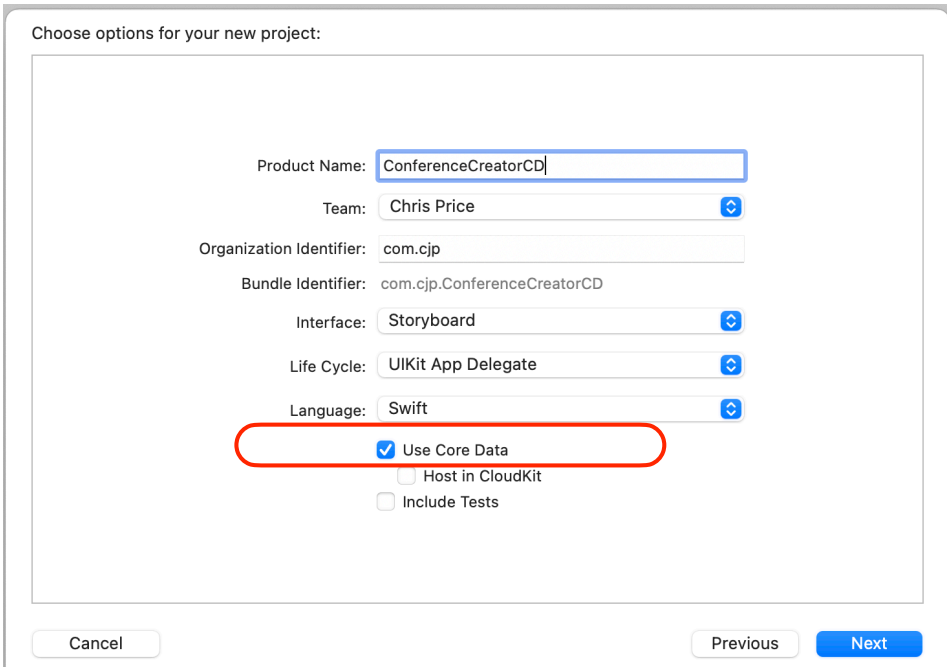
Conference Creator with CoreData

- We will make a new CoreData project and copy our views and view controllers across - this can be done, but has a few pitfalls
- We will re-implement the Model in CoreData (it was in JSON)

4

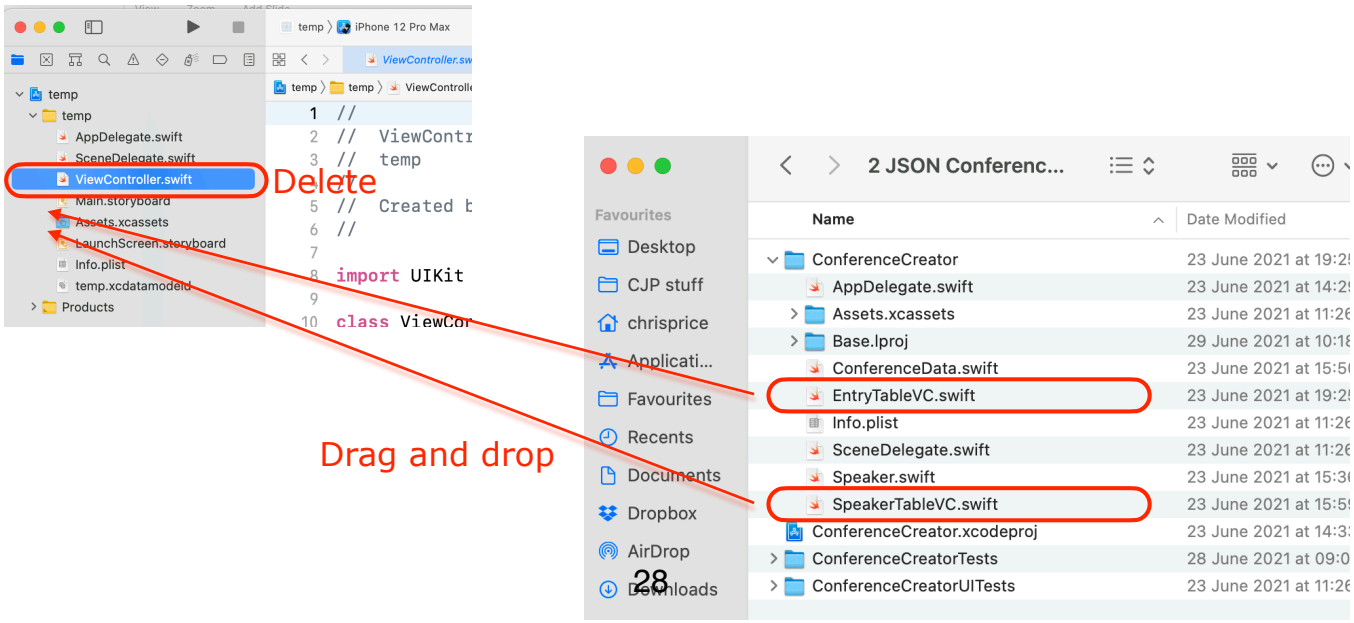
Step 1: Make new app with CoreData

- Make a new app called ConferenceCreatorCD as usual, but tick Use Core Data
- This adds extra code to App Delegate, and a data model



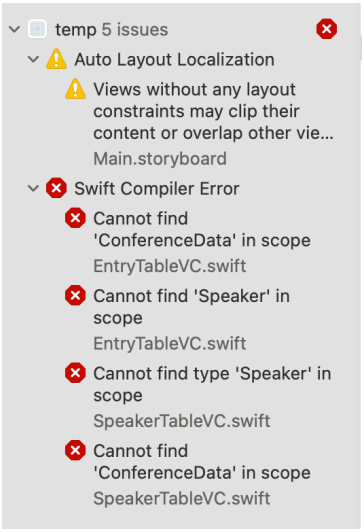
Step 2: Copy code from JSON version

- Delete ViewController and the start screen in Main.storyboard
- Open folder "2. JSON Conference Creator", and drag and drop SpeakerTableVC.swift and EntryTableVC into our new project

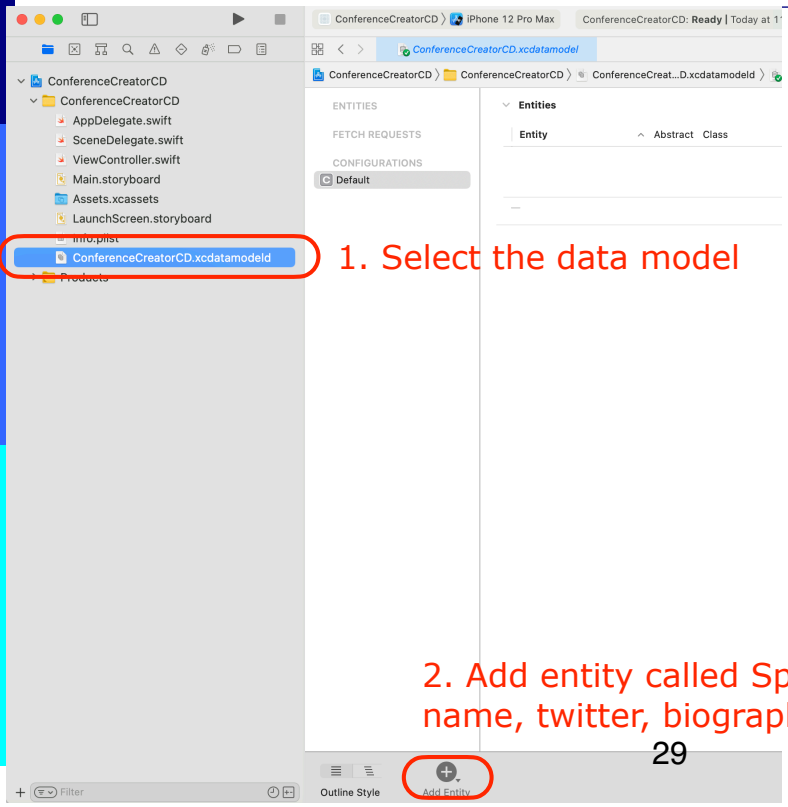


Step 3: Copy screens from old version

- Open JSON version in Xcode, and copy all three of the screens from Main.storyboard
- Paste them into Main.storyboard in the new CoreData project
- Select the NavigationController, and make it the Initial ViewController
- If you try running the app, you get 4 error messages
- We will define Speaker in CoreData, and that will fix two of the errors



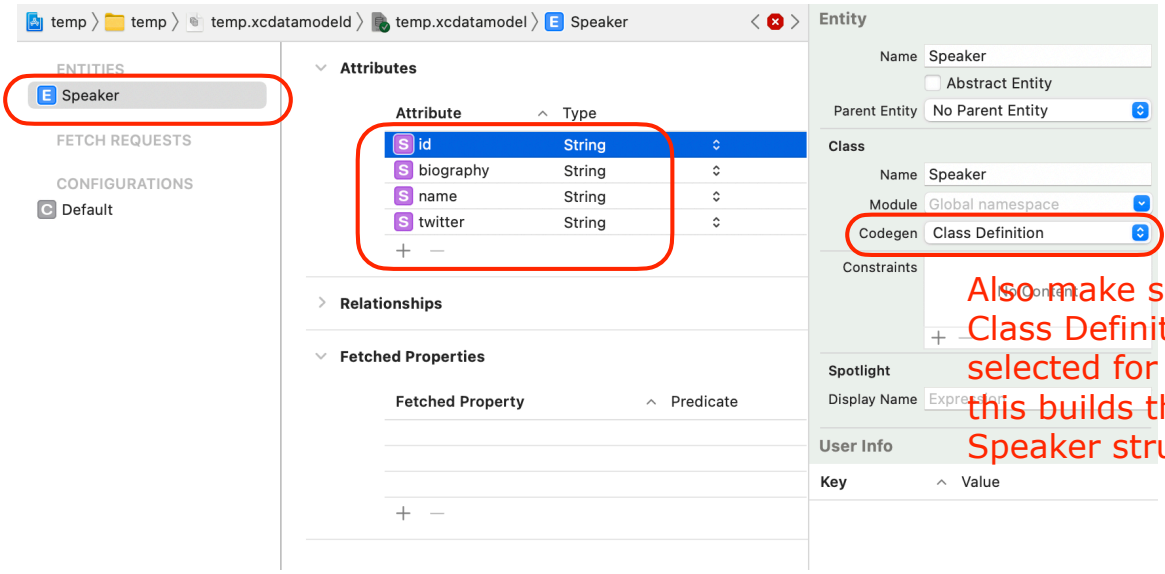
Step 4: Make a Speaker in the data model



1. Select the data model

2. Add entity called Speaker, add attributes - id, name, twitter, biography. Add them all as Strings

After Step 4: model looks like this



- Try running the app again, should now have 2 error messages

Step 5: Add code to EntryTableVC to save new speaker (also add "import CoreData")

```
func addSpeaker(id: String, name: String, biography: String, twitter: String) {
        // Get the managed context from the AppDelegate
        var managedContext: NSManagedObjectContext!
        let appDelegate = UIApplication.shared.delegate as? AppDelegate
        managedContext = appDelegate?.persistentContainer.viewContext

        // Make the new speaker and add it
        let entity = NSEntityDescription.entity( forEntityName: "Speaker",
                                                 in: managedContext)!
        let speaker = Speaker(entity: entity, insertInto: managedContext)
        speaker.id = id
        speaker.name = name
        speaker.biography = biography
        speaker.twitter = twitter
        try? managedContext.save()

        //Following code is just to find Documents (not really needed)
        let docDirectory = FileManager.default.urls(for: .documentDirectory,
                    in: .userDomainMask).first!
        let fileURL = docDirectory.appendingPathComponent("dummy")
        print( "\(fileURL)" )
}
```

Step 6: Replace line with error in EntryTableVC

- Instead of call to addSpeaker in ConferenceData, add the following code:

```
addSpeaker(id: id, name: name, biography: bio, twitter: twitter)
```

- EntryTableVC should now compile and save a newly created speaker - only one error to fix, in SpeakerTableVC, where we want to read in all saved Speakers

Step 7a: Add code to SpeakerTableVC

- Start by adding "import CoreData" here too
- Next we add the managedContext at the top of the class, as we will reuse it each time we enter the class

```
class SpeakerTableVC: UITableViewController {
  var managedContext: NSManagedObjectContext!
```

New code

- We set up the context once, in viewDidLoad:

```
let appDelegate = UIApplication.shared.delegate as? AppDelegate
managedContext = appDelegate?.persistentContainer.viewContext
```

Step 7b: Add code to SpeakerTableVC

- Replace the call to allSpeakers in viewDidAppear with the following code:

```
let fetch: NSFetchRequest<Speaker> = Speaker.fetchRequest()
fetch.predicate = NSPredicate(format: "id != nil")
do {
     let results = try managedContext.fetch(fetch)
     speakers = []
     for speaker in results {
         speakers.append(speaker)
     }
   } catch let error as NSError {
       print("Could not fetch \(error), \(error.userInfo)")
   }
```

13

Polishing app

- Our Speaker list used to call sorted, but our auto-defined Speaker is not Comparable
- If we add the following Extension to Speaker, then we can sort the list

```
extension Speaker: Comparable {
    public static func < (lhs: Speaker, rhs: Speaker) -> Bool {
        if let leftname = lhs.name, let rightname = rhs.name {
            return leftname < rightname
        }
        return false
    }
}
```


14

Things to note

- We have created a model, and CoreData takes care of the storing of objects
- We have seen reading and writing of objects
- We have seen how to extend an existing class (chapter 2.2 of Data Collections has more on this)
- Our example was simple - we can add relationships as well as attributes, and have linked objects
- CoreData is a big enough subject to have whole books on it - they are referenced in the course reading list

15

Session 18: Practical Animations

- How animations are used in iOS
- Delighting users with animations
- Types of animation that are possible
- *Lab 18: Building animations*

- This covers lesson 2.3 of Development in Swift Data Collections

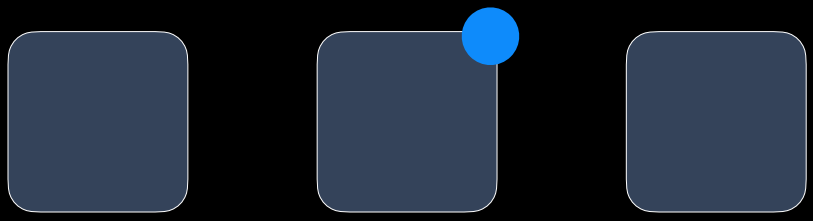
1

Animations in iOS

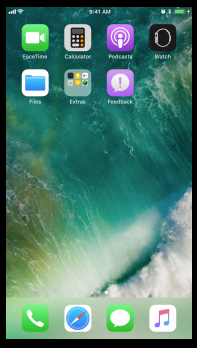
- As with Powerpoint, a lot is too much!
- Built-in animations already do some of this
  - When app starts / ends
  - When "Show"ing a new screen
  - When leaving a screen
  - When showing a modular screen
  - When scrolling
  - When reaching the end of a table

2

Why animate?
Direct the user's attention

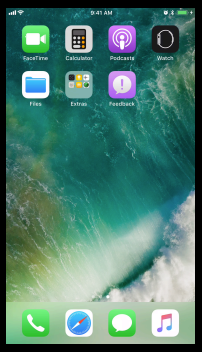


Why animate?
Keep the user oriented



Why animate?

Connect user behaviours



What can be animated?

`UIView`

- `alpha`
- `backgroundColor`
- `bounds`
- `center`
- `frame`
- `transform`

UIView animation methods

```
animate(withDuration:animations:)

animate(withDuration:animations:completion:)

animate(withDuration:delay:options:animations:completion
:)

animate(withDuration:delay:usingSpringWithDamping:initia
lSpringVelocity:options:animations:completion:)
```

Animation closures
animate(withDuration:animations:completion:)

```
UIView.animate(withDuration: 2.0, animations: {
    //animation closure
    viewA.alpha = 0.0
}) { (_: Bool) in
    //completion closure
    UIView.animate(withDuration: 2.0, animations: {
        //second animation closure
        viewB.alpha = 1.0
    })
}
```

Where might you add animations

- To show transitions
  - e.g. finishing a test
- To indicate completion of a transaction
  - e.g. adding an item to a basket
- To get the user's attention
  - e.g. because some mail has arrived

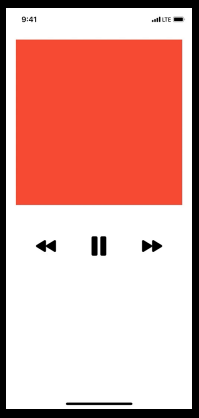
9

Practical Animation

Learn how to use the UIView class and closures to add animations that improve the presentation and functionality of your apps.

Create a wireframe—just the views, without actual functionality—of the Now Playing screen in the Music app.

(Page 349 of Data Collections book)



Session 19: Web browsing, web services

- Integrating web browsing into your app
- Calling web services and receiving data asynchronously
- Concurrency and Grand Central Dispatch
- Processing data and updating screens
- *Lab 19: Reading and processing data from a web service*
- This covers lessons 1.8, 2.4, 2.5, 2.6 of Development in Swift Data Collections

1

Easiest web access - web browser within your app

- If you want to show a web page, you can link to Safari to load it
- Loading it in a web browsing screen within your app is as easy and a better user experience

2

Easiest web access - web browser within your app

- If you want to show a web page, you can link to Safari to load it
- Loading it in a web browsing screen within your app is as easy and a better user experience
- Lets make an app that accesses Elm's web page - first make a simple app, and put a button on the home page

Add code to ViewController

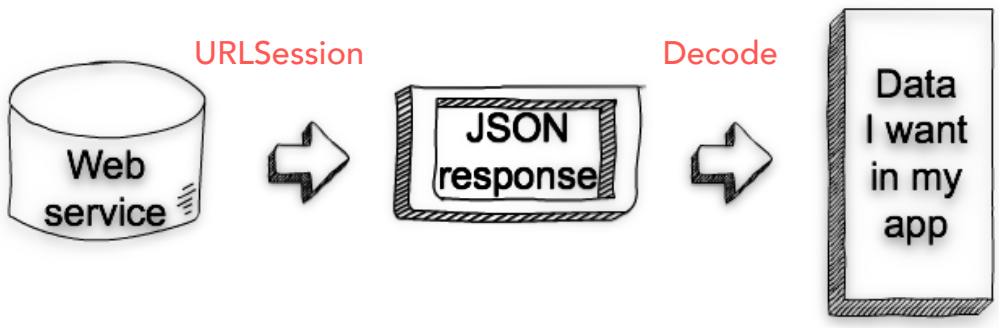
- Add following code to button action - also need to *import SafariServices*

```
@IBAction func loadURL(_ sender: UIButton) {
    // Do any additional setup after loading the view.
    if let pageURL = URL(string: "https://www.elm.sa/") {
       let browser = try SFSafariViewController.init(url: pageURL)
       present(browser, animated: true, completion: nil)
    }
}
```

Optional extra exercise

- Add a segmented control to allow the user to switch between the Arabic version of the web site and the English version of the site.

Next level - load data from web service



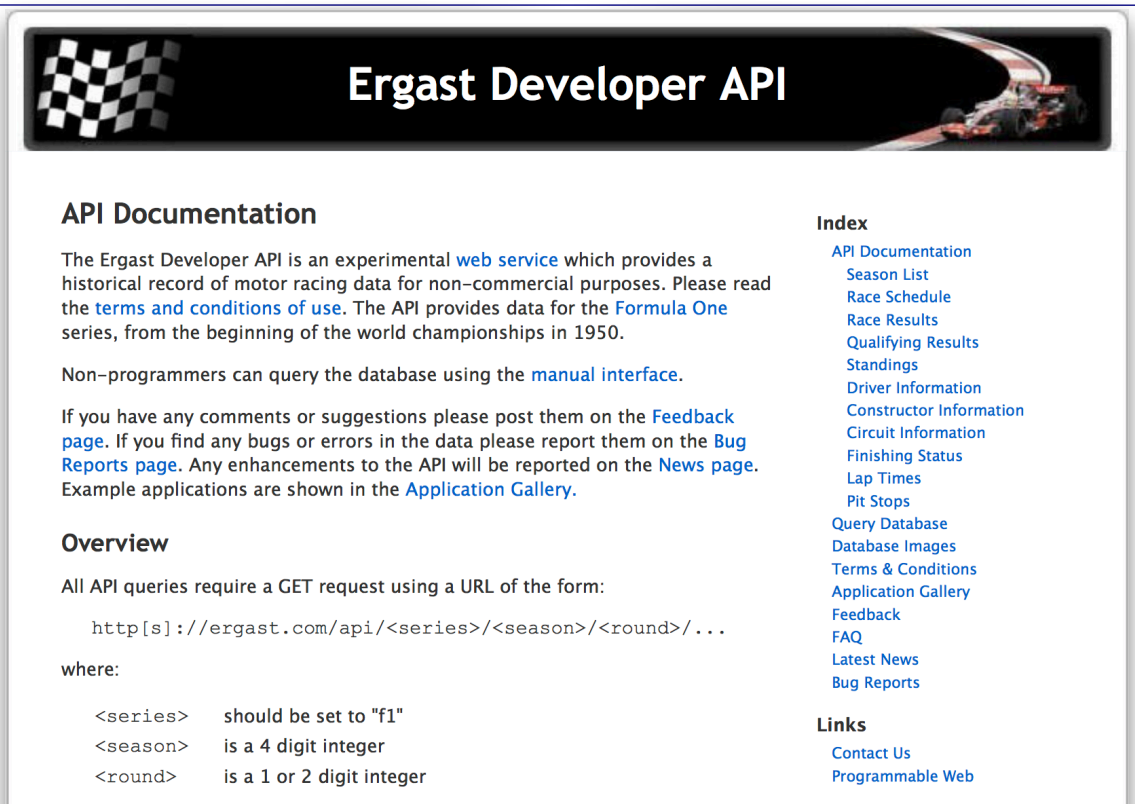
What do we have to do (Step 1)

- Make a URLSession on the URL that addresses the JSON web service that you want to access, and call "resume" to run it. The data will come back asynchronously, and the closure on the URLSession declaration tells you how to deal with it

What do we have to do (Step 2)

- Identify structure of the JSON and make a matching codable struct that parallels it, providing properties only for the JSON fields that you want to keep.
  - You decode the data into the codable structure you have defined
  - The decoder sorts through the JSON and pulls out the data you want into your codable struct

Our example web service



We will use a playground to explore getting the data and decoding it

- Playgrounds execute once synchronously by default, but we can make it handle asynchronous code by adding this at the top:

```
import PlaygroundSupport

PlaygroundPage.current.needsIndefiniteExecution = true
```

Step 1: Get the data with URLSession

- We can get info on all 2016 Grand Prix races with a web service call:

```
if let url = URL(string: "http://ergast.com/api/f1/2016.json") {
    let task = URLSession.shared.dataTask(with: url) { (data, response, error) in
        if let goodData = data {
            print( String(decoding: goodData, as: UTF8.self))
        }
    }
    task.resume()
}
```


HERE'S ONE I PREPARED EARLIER...


```
{"MRData":
    {"xmlns":"http:\/\/ergast.com\/mrd\/1.4",
     "series":"f1",
     "url":"http://ergast.com/api/f1/2016.json",
     "limit":"30",
     "offset":"0",
     "total":"21",
     "RaceTable":
        {"season":"2016",
         "Races":[
            {"season":"2016",
             "round":"1",
             "url":"https:\/\/en.wikipedia.org\/wiki\/2016_Australian_Grand_Prix",
             "raceName":"Australian Grand Prix",
             "Circuit":
                {"circuitId":"albert_park",
                 "url":"http:\/\/en.wikipedia.org\/wiki\/Melbourne_Grand_Prix_Circuit",
                 "circuitName":"Albert Park Grand Prix Circuit",
                 "Location"
                    {"lat":"-37.8497",
                     "long":"144.968",
                     "locality":"Melbourne",
                     "country":"Australia"}
                },
            "date":"2016-03-20",
            "time":"05:00:00Z"
            },
            {"season":"2016",
             "round":"2", ….

            }, ….
              ]
        }
    }
}
```


Part of this big MRData record

Data I actually want

In a record with a bunch of other stuff

There's an array of them in a field called Races

Part of a record called RaceTable

PICK OUT THE DIFFERENT LEVELS

4. Record attribute called MRData contains RaceTable

3. Record attribute called RaceTable contains Races

2. Array attribute of Race called Races

1. Race


```
{"MRData":
    {"xmlns":"http:\/\/ergast.com\/mrd\/1.4",
     "series":"f1",
     "url":"http://ergast.com/api/f1/2016.json",
     "limit":"30",
     "offset":"0",
     "total":"21",
     "RaceTable":
        {"season":"2016",
         "Races":[
            {"season":"2016",
             "round":"1",
             "url":"https:\/\/en.wikipedia.org\/wiki\/2016_Australian_Grand_Prix",
             "raceName":"Australian Grand Prix",
             "Circuit":
                {"circuitId":"albert_park",
                 "url":"http:\/\/en.wikipedia.org\/wiki\/Melbourne_Grand_Prix_Circuit",
                 "circuitName":"Albert Park Grand Prix Circuit",
                 "Location"
                    {"lat":"-37.8497",
                     "long":"144.968",
                     "locality":"Melbourne",
                     "country":"Australia"}
                },
             "date":"2016-03-20",
             "time":"05:00:00Z"
             },

             {"season":"2016",
              "round":"2", ….

             }, ….
               ]
        }
    }
}
```


Structure to get wanted items

- Declare the structure I actually want - it is in a record with lots of other fields, but the JSON decoder will just find the ones we want:

```
struct Race: Decodable {
    let round: String
    let url: URL?
    let raceName: String
}
```

Build outwards to match whole data file

- There is an array of Race records called Races within a RaceTable, within an MRData

```
struct Schedule: Decodable {
    struct MRDataStruct: Decodable {
        struct RaceTableStruct: Decodable {
            let Races: [Race]
        }
        let RaceTable: RaceTableStruct
    }
    let MRData: MRDataStruct
}
```

15

Step 2: Decode the data

- Having added the definition for a Race and a Schedule to our playground, we can replace the print statement after ***if let*** *goodData = data* with code to decode the JSON into a Schedule

```
let decoder = JSONDecoder()
let schedule = try! decoder.decode(Schedule.self, from: goodData)
let races = schedule.MRData.RaceTable.Races
for race in races {
      print( "Round \(race.round): \(race.raceName)")
}
```

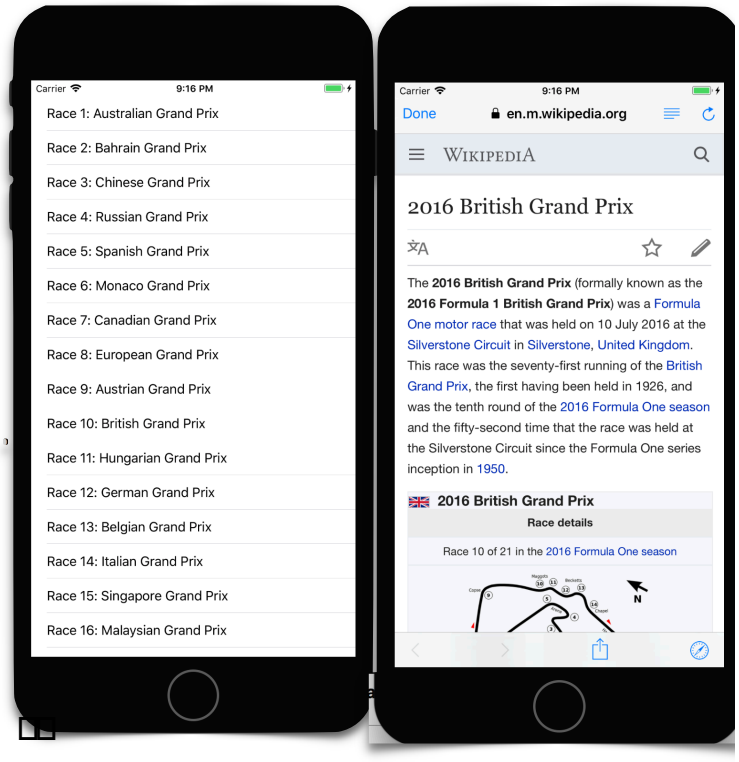

16

Several refinements are possible

- Error handling can be improved
- Explicit decoders can be written where you specify in detail how to interpret data
- Naming mappings can also be declared if you do not like the names of fields in the JSON

Lets make an app from this data

- We can add this code in viewDidLoad in a tableviewcontroller, and make an app which is a list of the 2016 Grand Prix circuits, and when we select one of them, it loads a web page from the URL associated with that circuit



What you need to do (main screen)

- New app, make initial screen a tableviewcontroller
- Declare an array of Races at the top level, and read data into the array in viewDidLoad (using code from our last playground)
- Use the array of Races in the usual way when populating tables

Problem!

- When we run the app, no data appears on screen
- Depending on how you have written it, the app may crash as well
- We can see better what the problem is if we initialise our array of Races as follows:
  - Var races = [Race(round: "", url: **nil**, raceName: "Waiting for data")]

The table is populated BEFORE we get the JSON data

- viewDidLoad runs before we populate the array, but the data is not obtained at that point
- viewDidLoad runs on the main thread, but the decoding is done asynchronously on a separate thread, AFTER viewDidLoad and numberOfSections and the other calls terminate

21

How it goes

Main thread

Other thread

Time

Initialise races

viewDidLoad

numberOfSections

numberOfRowsInSection

cellForRowAt Indexpath

(Stop)

Data received - do the code in the closure in URLSession

Need to update table now

22

We know how to update a table

- We call self.tableView.reloadData()
- However, this will fail (try it)
- All UI updating MUST be done on the main thread
- We can do this in the closure by explicitly telling the system to run reloadData on the main thread:

```
DispatchQueue.main.async {
   self.tableView.reloadData()
}
```


Finally we need to show a web browser with the URL if the user selects a table cell

- We can do this by overriding the tableView function didSelectRowAtIndexPath
- This lets us take an action when a specific cell is chosen

```
override func tableView(_ tableView: UITableView,
        didSelectRowAt indexPath: IndexPath) {
    if let url = races[indexPath.row].url {
        let myWebVC = SFSafariViewController(url: url)
        present(myWebVC, animated: true, completion: nil)
    }
}
```

We have successfully built an app that deals with asynchronous inputs

Next we will look at the more general issues of concurrency

25

Concurrency in Swift: why and how

While building our web app, we called:

```
DispatchQueue.main.async {
    self.tableView.reloadData()
}
```

- This was necessary because we were working on a background thread
- How did we get on a background thread in the first place?

DispatchQueue is part of more general concurrency provided by Grand Central Dispatch (GCD)

- URLSession is built on top of GCD for the special case of getting web data, and hides much of GCD
- Essentially URLSession calls GCD to set up code that works on a background thread to handle the case of receiving data from a web request
- We can see the need for this if we use a more explicit call to get the data that does not use GCD

Calling

- The example in folder 6 of Session 19 calls Data(contentsOf:) instead of URLSession - this call waits for a reply, so the app freezes until that happens - using GCD gets us around this

```
if let url = URL(string: "https://ergast.com/api/f1/2016.json") {
        if let data = try? Data(contentsOf: url) {
             let decoder = JSONDecoder()
             let schedule = try! decoder.decode(Schedule.self, from: data)
             self.races = schedule.MRData.RaceTable.Races
             self.tableView.reloadData()
}
```

What is Grand Central Dispatch

- GCD is Apple's scheme for prioritising tasks
- It allows you to spin off a task, and say how important it is
- It also allows iOS to make as much use as possible of multiple cores, as tasks can then be assigned to different processors

5

Levels of importance in GCD

- GCD allows you to specify the Quality of Service (QoS) you expect for a task
- There are five levels of service
  - User Interactive - this is the top priority, as you want the interface to be responsive
  - User Initiated - next level, for tasks that the user is waiting for, e.g. processing a photo where the user wants to see the results
  - Utility - for tasks the user is not actively waiting for, such as checking a remote database for updates
  - Background - for tidying up tasks
  - Default - where activities such as URLSession would run if nothing else was specified. It is between User Initiated and Utility
- You can think of the levels of service as being priority queues - user initiated tasks will run before default tasks, before Utility tasks, etc.

6

Example use in Conference app

- We have included all of the photos of speakers within our conference app
- What if we did not do that, but instead had to download the photos from the Internet?
- We would not want to do this on the main thread
- We could call Data(contentsOf:) on the Utility thread, once for each photo that is not presently stored
- Folder 7 gives an example of this
- In practice, we would probably use URLSession to do this, but what is happening behind the scenes is similar

7

Example use in Conference app

```
func loadSpeakerPhotos() {
    let accessStem = "https://github.com/chrisinaber/iosdevforelm/raw/main/Speakers/"
    for speaker in allSpeakers {
        let fileName = speaker.id + ".jpg"
        if !fileExistsInDocuments(fileName) {
            let fileString = accessStem + fileName
            if let url = URL(string: fileString ) {
             DispatchQueue.global(qos: .utility).async {
                if let data = try? Data(contentsOf: url) {
                    let photoURL =  self.urlToFileInDocuments(fileName)

                    //Write the data to backing store.
                    try? data.write(to: photoURL, options: .noFileProtection)
                    print( "file is at \(photoURL)")
                  }
                }
            }
        }
    }
}
```

8

This is about to get easier

- Many calls to dispatch queue end up embedded in each other
- Example: if we were reading the json with speaker names in it, then we would embed the call to get the photos inside the successful call to get the json
- This will not be necessary with iOS 15, because there is new syntax for handling this kind of thing
- It will only work with the new operating system though

9

Session 20: Cocoa pods and cloud services

- Setting up Google Firebase
- Adding Cocoapods to your app
- Running a Cocoapods build
- Using Firebase for cloud storage
- Authorization and security
- Local and remote data
- *Lab 20: Building a Firebase app*
- This material is not covered in the Develop in Swift books

1

Google Firebase

- Google's main cloud service
- Works well with both iOS and Android
- One of the easiest ways to provide a web server for your app
- Free up to 50K writes per day

2

CocoaPods

- Was the most popular dependency manager for iOS
- Contains thousands of libraries that you can include in your app
- Has been the default way of including Firebase in your app
- Swift Package Manager is starting to replace it as it is more integrated with Xcode

3

Using Firebase for the Conference app

- Several ways we could use it
  - 1/ Separate records for each type of object (speakers, sessions, locations etc.). This would be the way to go with large numbers of objects.
  - 2/ Single object that holds our details in JSON format and read it all in. This works well for the number of records we have.

4

Advantages over in-app data

- We can make changes to the data without going through Apple's approval process
- We can serve different data to different people (probably not in a conference app!)
- We can perform database operations on the server

5

Disadvantages over in-app data

- If we don't have access to the server, then we don't have data (Firebase makes this less serious by caching data)
- We need more code to get the data in

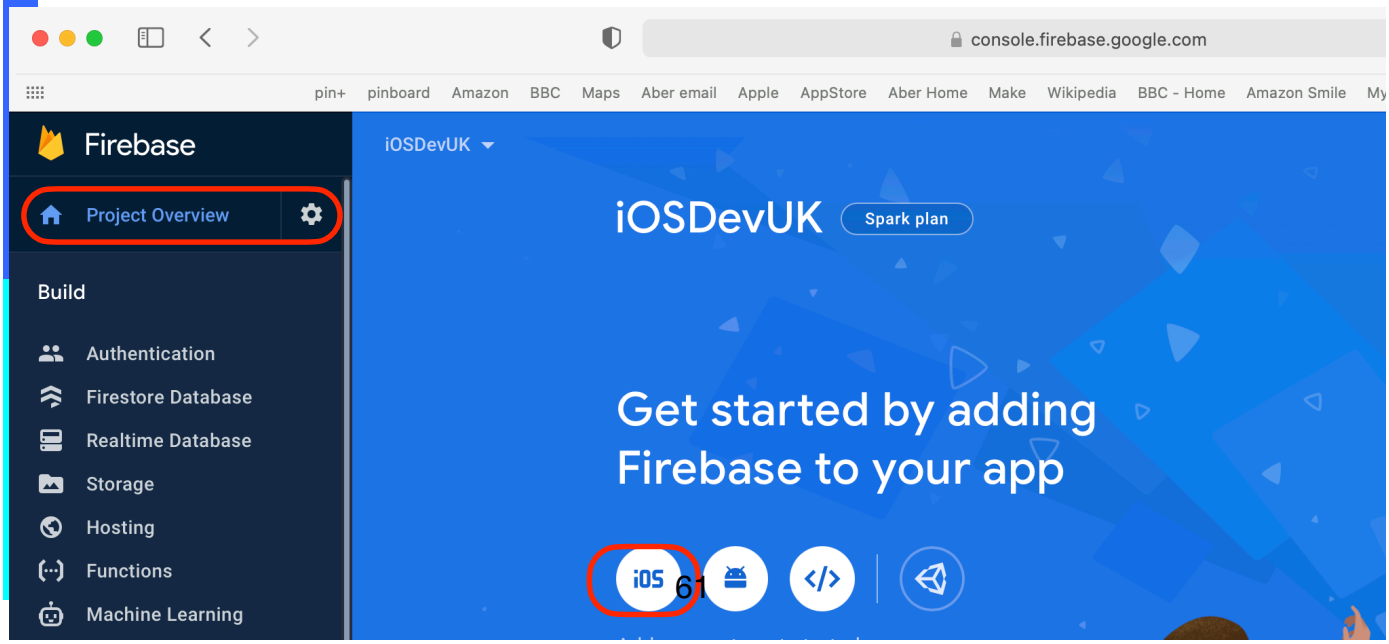
6

Setting up Firebase for the conference

- Step 1: We need a Firebase database with the data in (this has been done - the JSON data has been placed on a server)
- We will add this to the start project in Session 20 under "2 Conference app start"
- This project has a Speaker table, but needs to get the data from Firebase

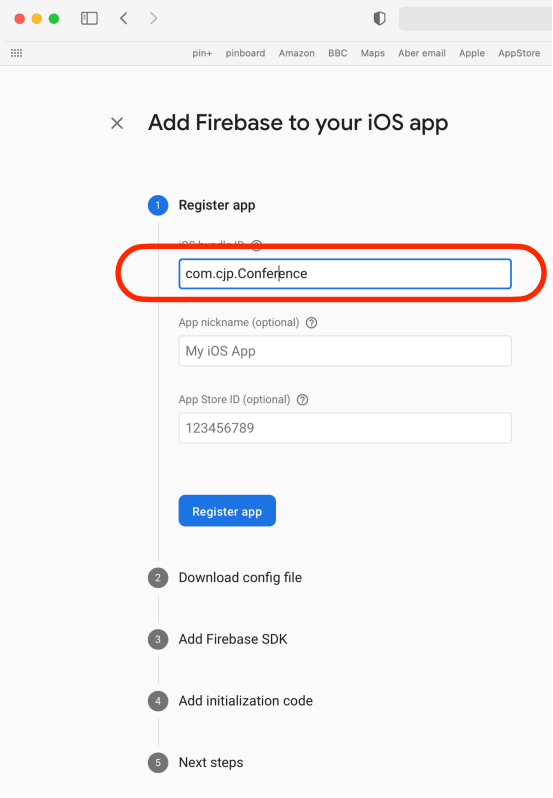
Step 2: Link Firebase to your app

- Choose add iOS app to Firebase in Project Overview



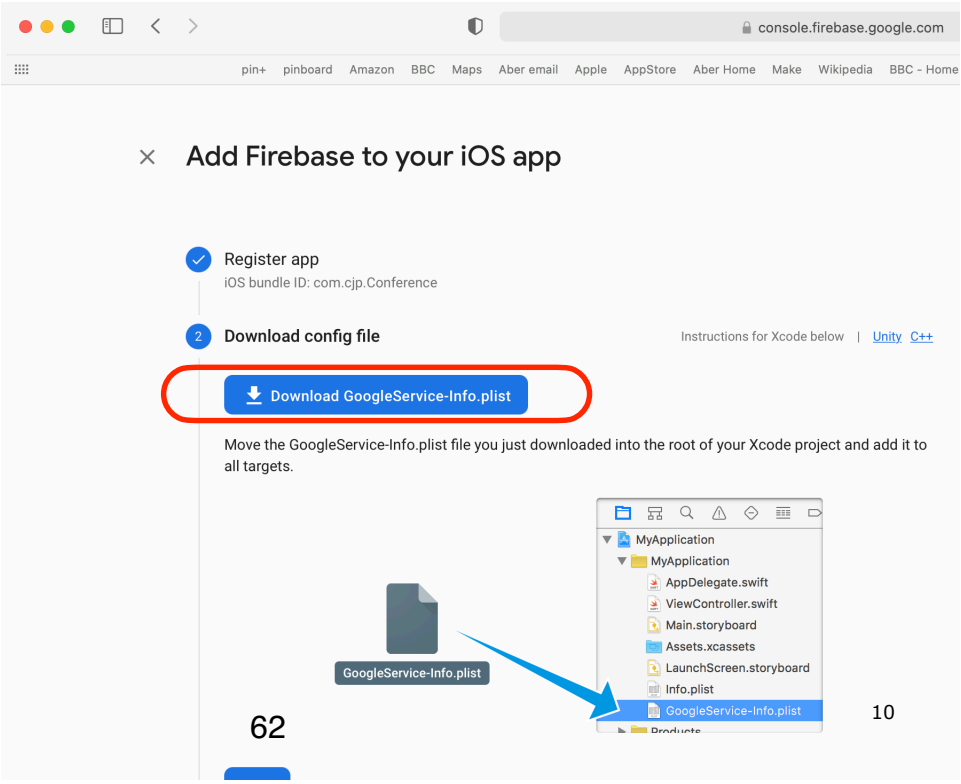
Step 2: Link Firebase to your app (continued)

- (If starting a new app) Register the app with the app ID you are using
- (You don't need to do this as it is already done in the Firebase database we are using)



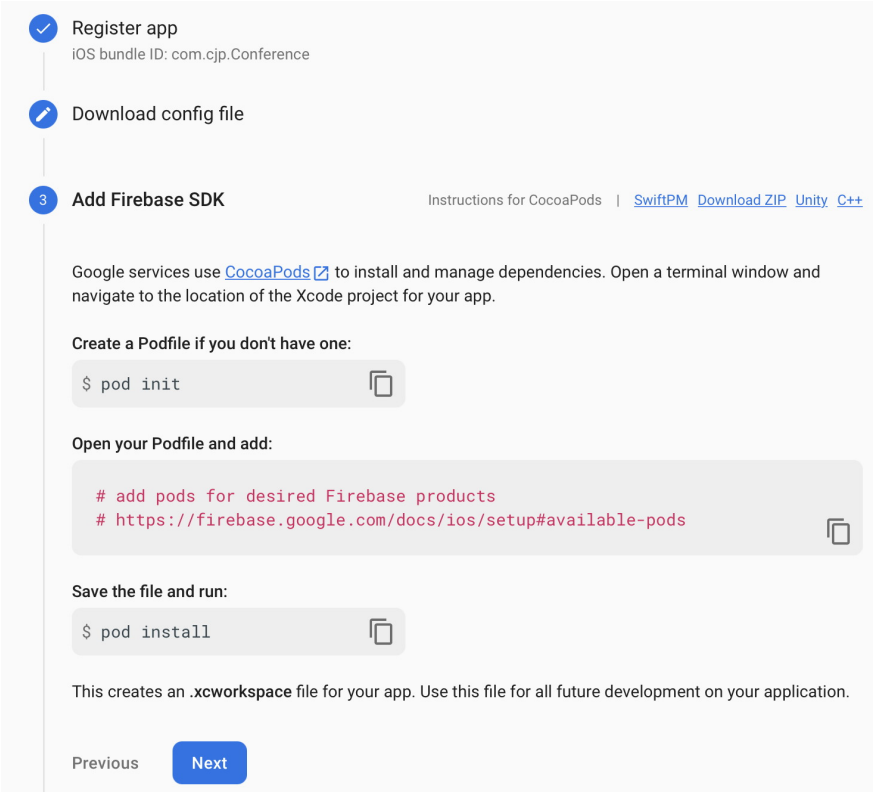
Step 2: Link Firebase to your app (continued)

- Download the resulting Google services file
- Add it to your iOS project
- (It is included in Session 20 under "3 Google Services")



Step 3: Include firebase libraries

- Standard way used to be to use CocoaPods



11

CocoaPods being replaced

- Gradually being replaced by Swift Package Manager (SPM)
  - much tidier way to include packages in your software
  - Better integrated with Xcode and Swift
- SPM was in beta for Firebase, recently become established

12

Adding Firebase via SPM

Via Xcode

Swift Package Manager support requires Xcode 12.5 or higher.

1. If migrating from a CocoaPods-based project, run `pod deintegrate` to remove CocoaPods from your Xcode project. The CocoaPods-generated `.xcworkspace` file can safely be deleted afterward. If you're adding Firebase to a project for the first time, this step can be ignored.
2. In Xcode, install the Firebase libraries by navigating to **File > Swift Packages > Add Package Dependency...**
3. In the prompt that appears, select the Firebase GitHub repository:

```
https://github.com/firebase/firebase-ios-sdk.git
```

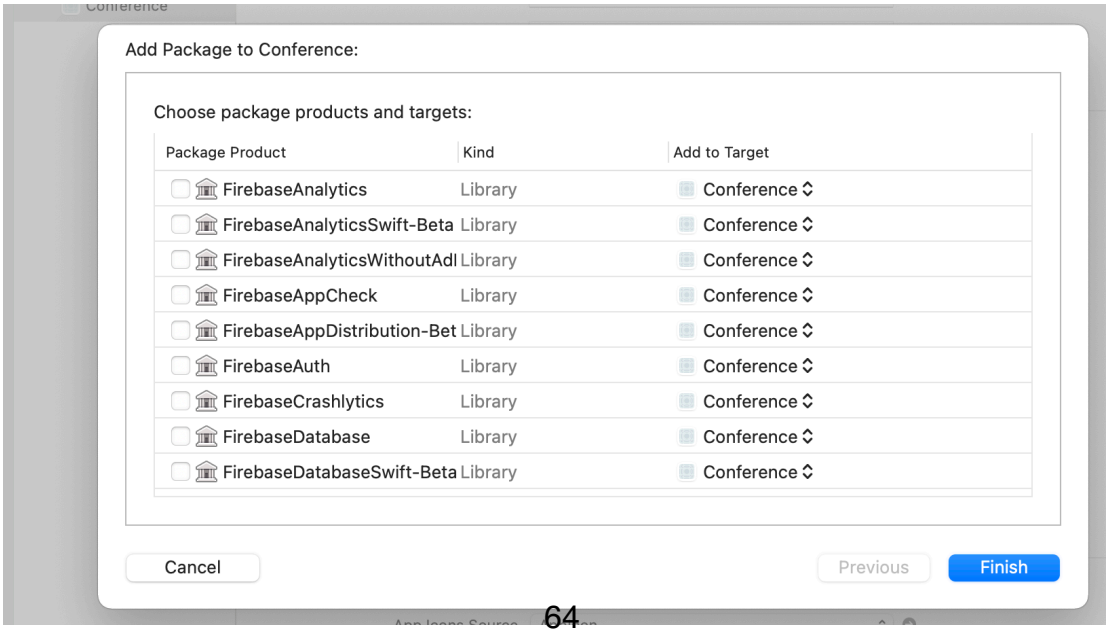
4. Select the version of Firebase you want to use. For new projects, we recommend using the newest version of Firebase.
5. Choose the Firebase products you want to include in your app.

Once you're finished, Xcode will begin resolving your package dependencies and downloading them in the background.

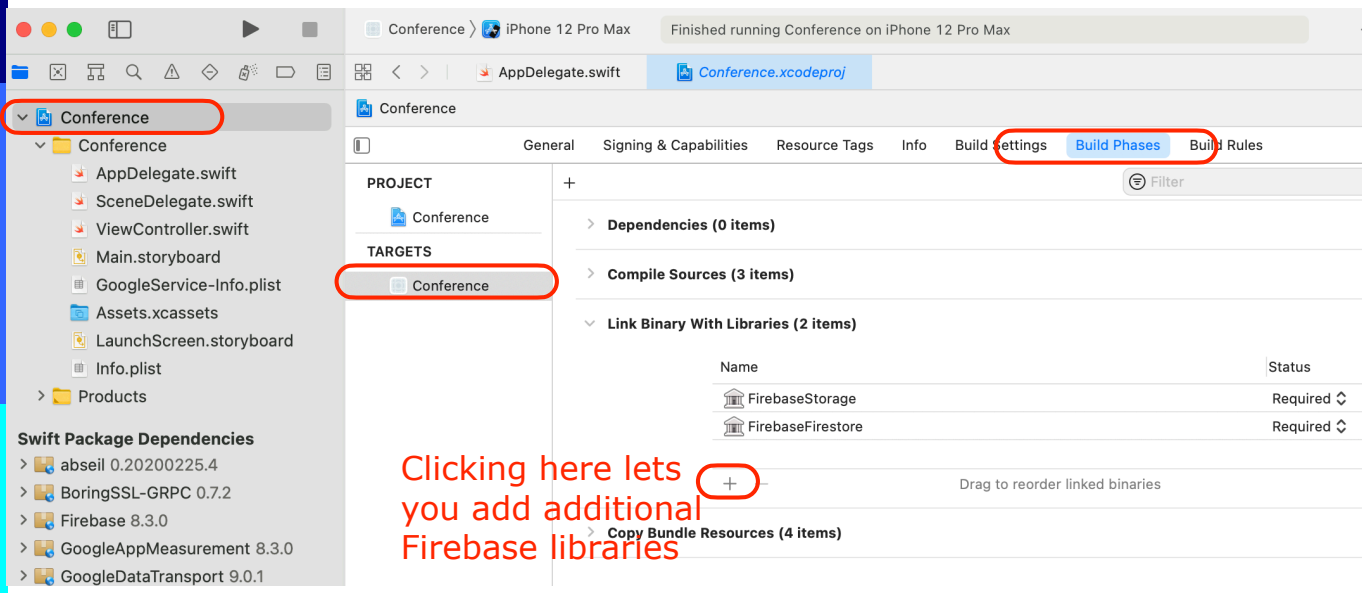
- Copyable link: https://github.com/firebase/firebase-ios-sdk.git

Make choice of Firebase products

- Choose FirebaseStorage and FirebaseFirestore



How to update libraries used

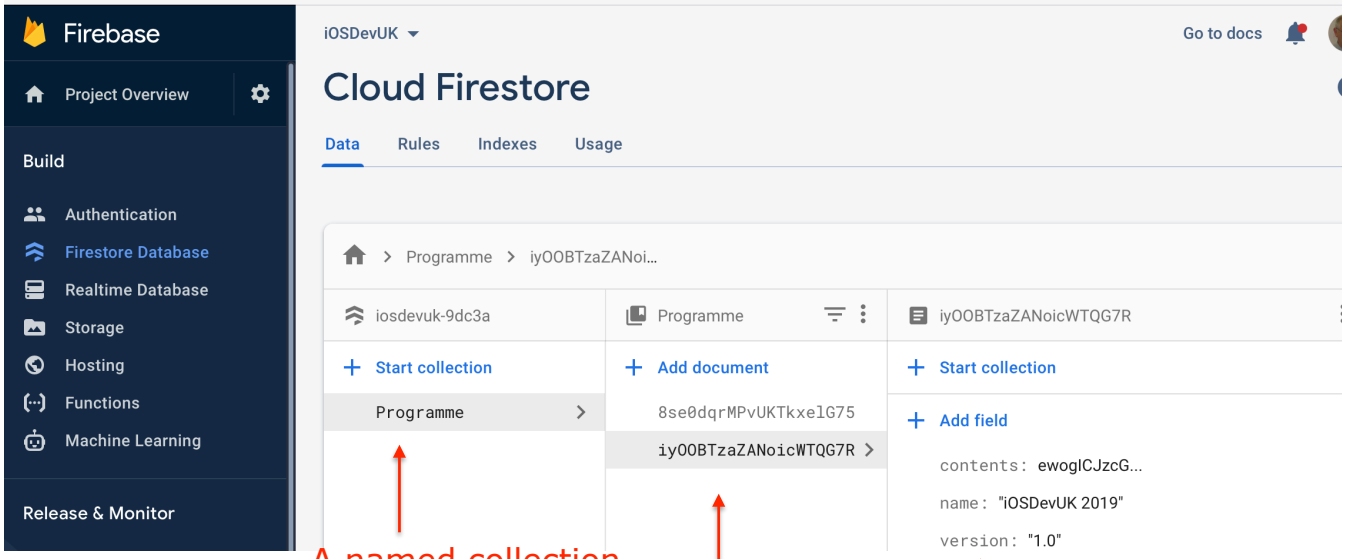


Step 4: Start firebase link in our app

- In AppDelegate.swift, update didFinishLaunchingwithOptions to start Firebase

```
func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
    // Override point for customization after application launch.
    FirebaseApp.configure()
    return true
}
```

What our Firebase data looks like

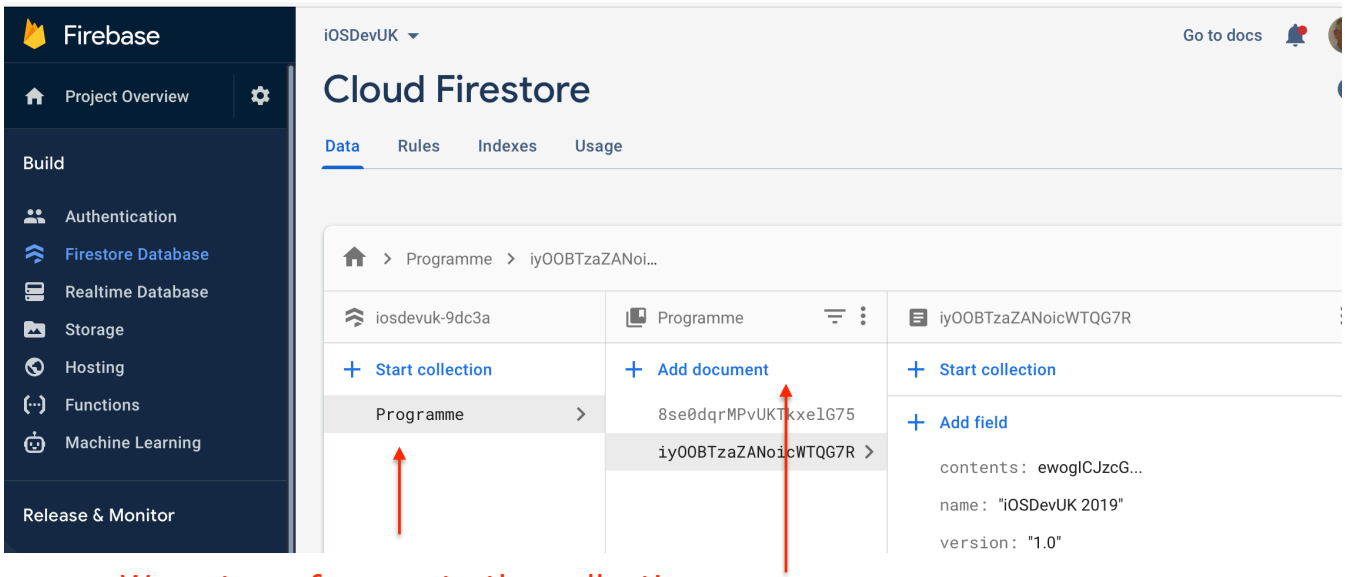


A named collection

has documents

which have fields

What we do with it



We get a reference to the collection (Programme in our case)

We ask for the document(s) with name "iOSDevUK 2019"

Step 5: Read data from Firebase

- Our singleton, ConferenceData.swift, has an empty method loadAllConfInfo. Previous version used to read it from a local JSON file.
- We now need to read it from the Firebase we are linked to.

Code for loadAllCodeInfo

```
func loadAllConfInfo() {
    // Get Firebase document we want
    let db = Firestore.firestore()
    let docRef = db.collection("Programme").document("iOSDevUK2019")
    docRef.getDocument() { (document, error) in
       // Code executed asynchronously on return from network call
       guard let document = document else {
           print("Document does not exist")
           return
       }
       guard let docData = document.data() else {
           print("Document has no data")
           return
       }
       if let encodedData = docData["contents"] as? Data {
           let decoder = JSONDecoder()
           if let confInfo = try? decoder.decode(ConfInfo.self, from: encodedData) {
               self.allTalks = confInfo.talks
               self.allSpeakers = confInfo.speakers
               self.allLocations = confInfo.locations
           }
       }
   }

}
```

Add update parameter to loadAllConfInfo

- func loadAllConfInfo(update: @escaping () -> Void)
- When loadAllConfInfo has successfully decoded the data, we can call update() to tell the caller that the data is available
- We need to call loadAllConfInfo from SpeakerTableVC instead of implicitly, as we need it to specify what we want to do when the call completes (we want to update the speaker table, but we need to do it on the main thread)

21

Step 6: Call loadAllConfInfo from SpeakerTableVC

- Delete call to loadAllConfInfo in the init of ConferenceData singleton
- Add call to loadAllConfInfo to viewDidLoad in SpeakerTableVC as follows:

```
override func viewDidLoad() {
    super.viewDidLoad()
    ConferenceData.sharedInstance.loadAllConfInfo(update: {
        self.speakers = ConferenceData.sharedInstance.allSpeakers
        DispatchQueue.main.async {
            self.tableView.reloadData()
       }
    })
}
```

22

What we have achieved

- We have shown how to load the conference data and we have used it for the Speaker Table
- As the photos are not in the Speaker Table, we would need to load them separately

What we have not done

- Reacting to updates to the database
  - We just need to load the data in this case
  - Where it is continually changing, we can make a listener, and update when Firebase changes
- Keep data private to the user
  - if we were saving the user's data in Firebase, we would want them to log in to access their personal records
  - There are a range of libraries to log in with email, Apple mail, Google, Facebook etc.

Session 21: Notifications

- What are notifications?
- Working with notifications
- Creating and Issuing local notifications
- Issuing and catching a push notification
- *Lab 21: Adding notifications to your app*

- Local notifications are covered in lesson 3.5 of Development in Swift Data Collections
- Push notifications are not covered in the books

What are notifications?

- Messages to your phone either generated from the app itself (Local notifications) or sent to the app from a server (Push notifications)
- If your app is not active, they cause a message to the phone screen
- If your app is active, they can prompt your app to carry out some activity

Working with notifications

- Like use of location, the user needs to give permission to allow notifications
- They are so misused by apps that many users say no to notifications by default
- If you have a good reason for sending notifications to the user, you need to make that clear to them BEFORE asking them to allow notifications
- Excellent article on how to handle users to get permission: https://blog.hurree.co/blog/ios-push-notification-permissions-best-practises

3

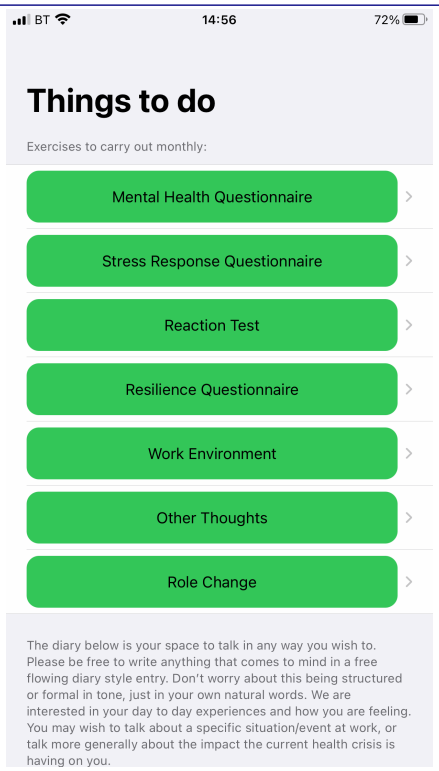
Local notifications

- Local notifications can be generated by:
  - Setting a specific time and date - "It is time for the weekly staff meeting"
  - Setting a time interval - "10 minutes gone"
  - Detecting a location - "You have arrived at your destination"

4

Example use of local notification

- App for Health Service Staff to record stress levels
- They are reminded at the start of each month to complete questionnaires for previous month



5

Example code

```
func addNotificationNextMonth() {
    // Put in a notification for 1st of next month to say do monthlies
    UNUserNotificationCenter.current().requestAuthorization(
       options : [.alert, .badge, .sound]) { success, error in
    if success {
       let content = UNMutableNotificationContent()
        content.title = "Start of the month"
        content.subtitle = "Please fill in the questionnaires"
        content.sound = UNNotificationSound.default
        // show this notification on startdate
        var dateComponents = calculateStartDate()
        let trigger = UNCalendarNotificationTrigger(dateMatching:
                                    dateComponents, repeats: false)
        // choose a random identifier
        let request = UNNotificationRequest(identifier: UUID().uuidString,
                                    content: content, trigger: trigger)
        // add our notification request
        UNUserNotificationCenter.current().add(request)
    }
}
```

6

Making local notification - step 1

- Make a new simple app and put a button and a stepper and a label on the screen
- Link them up to a button action doTimer, a stepper action changeValue, a stepper output minuteValue and a label output minuteCount
- Set minuteCount to 1, and make that the stepper minimum
- When stepper pressed, change minuteCount
- When the button is pressed, set the timer to give an alert after the amount of minutes on the stepper (see code on next screen)

Making simple notification - step 2

```
@IBAction func setTimer(_ sender: UIButton) {
    let minutes = minuteValue.value
    UNUserNotificationCenter.current().requestAuthorization(
              options : [.alert, .badge, .sound]) { success, error in
        if success {
           let content = UNMutableNotificationContent()
           content.title = "Timer"
           content.subtitle = "Your time is up!"
           content.sound = UNNotificationSound.default
            // show this notification after several minutes
            let trigger = UNTimeIntervalNotificationTrigger(
                              timeInterval: minutes*60.0, repeats: false)
            // choose a random identifier
            let request = UNNotificationRequest(identifier: UUID().uuidString,
                                        content: content, trigger: trigger)
            // add our notification request
            UNUserNotificationCenter.current().add(request) { (error: Error?) in
                if let error = error {
                    print(error.localizedDescription)
                }
            }

        }
    }
}
```

Push notifications

- Generated by a server
- Wider set of reasons are likely than with local notifications
  - To let a user know their parcel has been delivered
  - To tell a user about a great new marketing offer
  - To remind the user to do something
  - To tell the user that some status has changed

9

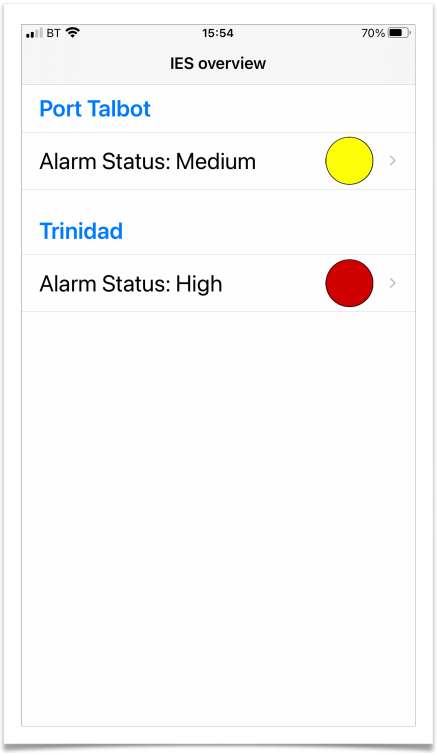
Why don't I get any notifications?

- Notifications don't appear if you are running the app - you need to set the notifications, then choose to exit the app on the simulator and notifications will show
- You can catch the notification in your app and do something (details in the Apple book) if you want to react when running the app

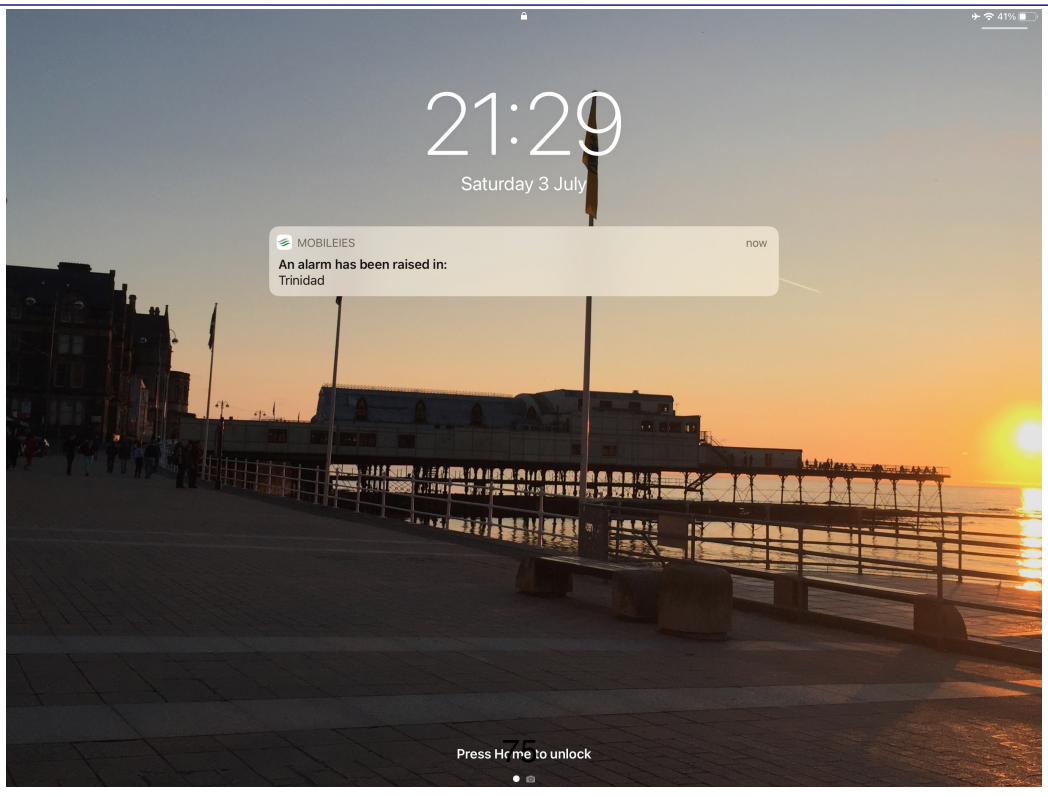
10

Example of push notification

- App for monitoring solar power plants
- Users want to know when a serious problem happens - even in the middle of the night
- Power plant data is in Firebase
- Firebase Functions generate a push notification sent to all registered devices whenever a new high alarm happens



Device gets message and plays sound when notification arrives



Session 22: Reactive apps

- Overview of SwiftUI
- MVVM
- Basics of building screens in SwiftUI
- Specific example of making Forms in SwiftUI
- Overview of Combine
- *Lab 23: Example of SwiftUI*

- This material is not covered in the Develop in Swift books

1

What is SwiftUI?

- SwiftUI is a User Interface Framework which replaces UIKit
- Instead of building Storyboards to show what screens should look like, you declare what the features you want to show are, and SwiftUI arranges them
- UI description can be conditional on the state of the app and when the state changes, the UI changes

2

Advantages of SwiftUI

- A lot of a complex app is about dealing with the state of your app (acting in different ways in different states) - a lot of that becomes easier with SwiftUI
- Many issues that are hard work in UIKit (resizing views, dynamic type size, light and dark views) are dealt with automatically in SwiftUI
- Apps will work across platforms (iOS, Mac, TVOS, Watch)
- You end up with significantly less code on average

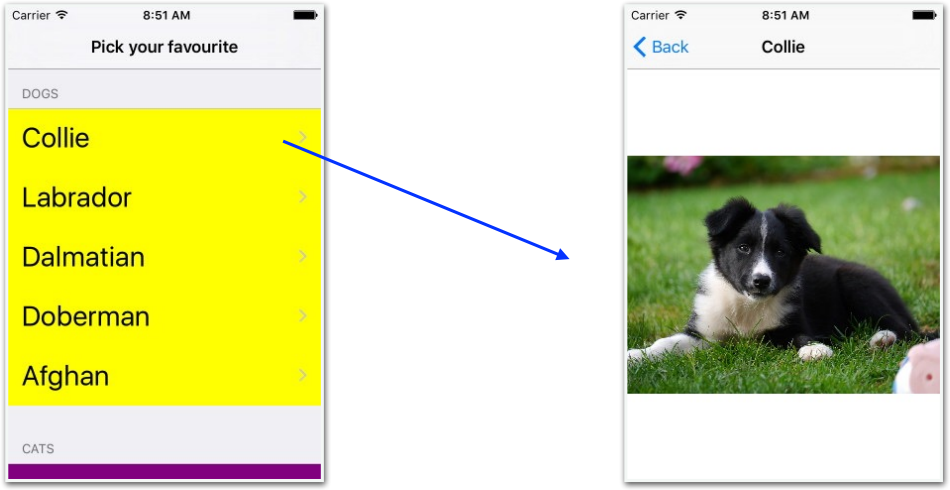
3

Disadvantages of SwiftUI

- It is a different way of thinking about your app - if you build it as you would have in UIKit, it will be clumsy
- Views can get very big if you are not careful - you need to build them up from smaller subviews
- Some screens and objects are not implemented in SwiftUI - this was a big issue in 2019, less so now
- Some of the best improvements need iOS15 (but you can build good apps in the 2020 release)
- Error messages are not always helpful

4

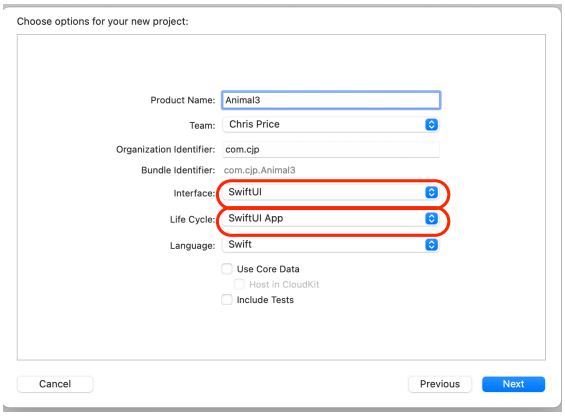
Example App - Animal3



5

Step 1

- Make a new Project called Animal3
- Make it a Single View project
- Choose SwiftUI for Interface and Life Cycle



- When project is created - see different files produced

6

Step 2

•Add animal photos to Assets as before
•Add animal data below to file ContentView after the import (same data as before, but better format)

```
var categories = [
    [["Dogs"], [ "Collie", "Labrador", "Dalmation", "Doberman", "Afghan"]],
    [["Cats"], ["Siamese", "Manx", "Persian"]],
    [["Rabbits"], ["Dutch", "Chinchilla", “Lionhead”]]]
```

•Change Text(“Hello world”) to Text(categories[0][0][0]) and run the app
•It should show *Collie* on screen
•You can also see what the screen should look like in the Preview screen - the Preview often stalls, but if you click Resume or press Option-Command-P, it should start again

Step 3

•We can make a list of the dogs with the following code

```
List{
     Text( categories[0][0][0])
     ForEach(categories[0][1], id: \.self) {
         Text($0)
    }
}
```

Step 4

•If we loop over the categories, we can list all the animals in sections

```
List{
    ForEach((0...2), id: \.self) { index in
        Text( categories[index][0][0])
        ForEach(categories[index][1], id: \.self) {
            Text($0)
        }
    }
}
```

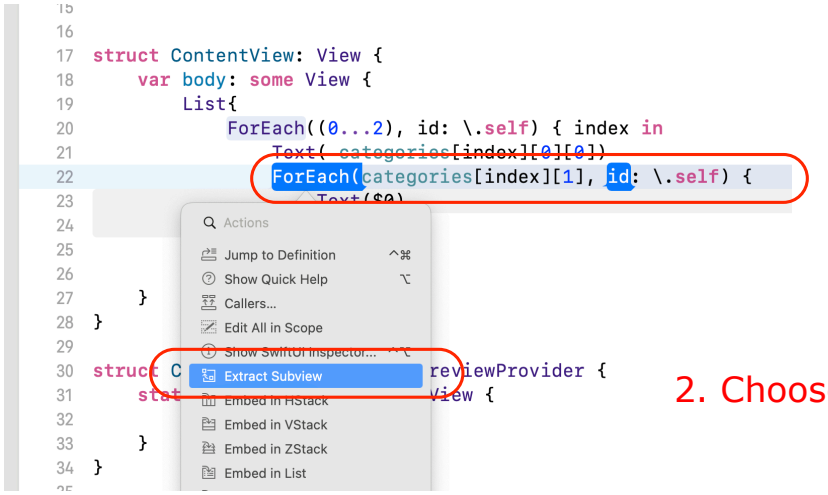
New code

New code

•You spend a lot of time reindenting SwiftUI code - you can do this by pressing COMMAND-A (to select the text) and then CONTROL-I

Step 5

•SwiftUI is made up of views, and we can keep them simple by extracting subivews.
•Command-click on the inner ForEach, and choose “Extract Subview”



1. Command-click here

2. Choose Extract Subview

Result of Step 5

```
17  struct ContentView: View {
18      var body: some View {
19          List{
20              ForEach((0...2), id: \.self) { index in
21                  Text( categories[index][0][0])
22                  ExtractedView()
23              }
24          }
25      }
26  }
27
28  struct ContentView_Previews: PreviewProvider {
29      static var previews: some View {
30          ContentView()
31      }
32  }
33
34  struct ExtractedView: View {
35      var body: some View {
36          ForEach(categories[index][1], id: \.self) {  Cannot c
37              Text($0)
38          }
39      }
40  }
41
```

A call to the subview is created, and selected so that we can change the name

The selected code is extracted into a subview

Step 6

- While ExtractedView is selected, type AnimalListView to rename it
- We need to pass a parameter of categories[index][1] to it
- Change AnimalListView as follows:

```
struct AnimalListView: View {
  let animals: [String]
   var body: some View {
      ForEach(animals, id: \.self) {animal in
         Text(animal)
      }....
```

Change this to animals

Change call to AnimalListView in ContentView to add a parameter:

```
AnimalListView(animals: categories[index][1])
```

Step 7

•We have made a table with very little code compared with what we do in UIKit
•We want to format the text, but first we will add the second screen
•We start by putting ContentView in a Navigation View

```
struct ContentView: View {
    var body: some View {
        NavigationView {
            List{
                ForEach((0...2), id: \.self) { index in
                    Text( categories[index][0][0])
                    AnimalListView(animals: categories[index][1])
                }
        } .navigationTitle( "Animals")
    }
```

13

Step 8

•We change AnimalListView so that when the list item is clicked, it brings up a new NavigationView. That could be a completely different custom view, but it is just an image, so we will call it inline

```
struct AnimalListView: View {
    let animals: [String]
    var body: some View {
        ForEach(animals, id: \.self) {animal in
            NavigationLink(
                destination: Image(animal),
                label: {Text(animal) })
        }
    }
}
```

14

Step 9

•We have all of the functionality now, we need to format things
•We will change AnimalListView to format the table cells
•We will put the table text into a HStack, and format it and add a photo. So Command-click on Text(animal) and choose Embed in HStack, then add the extra statements below:

```
HStack {
    Image(animal)
      .resizable()
      .frame(width: 70, height: 70)
    Text(animal)
      .font(.title)
      .padding(.horizontal)
    Spacer()
}
```

15

Step 10

•The photo on the second screen was of variable size - depending on the size of the actual photo. We can resize it to be consistent and fit within the screen (in fact we already did something similar in step 11 for the thumbnail we added to each cell.
•Do the following for the destination image:

```
NavigationLink(
      destination: Image(animal)
      .resizable()
      .frame(width: 300, height: 300),
```

16

Step 11

•We can preview for different modes (dark and light) and different screen sizes
•When we try that for light and dark modes, we will see that we have created a problem:
•Change the preview code as follows to see both modes:

```
struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ForEach(ColorScheme.allCases, id: \.self) {
                ContentView().preferredColorScheme($0)
        }
    }
}
```

17

Final Step

•We can fix the unreadable text in dark mode by specifying that the text is always black.

```
Text(animal)
   .font(.title)
   .foregroundColor(.black)
   .padding(.horizontal)
```

18

What we have seen

- Much easier to set up tables in SwiftUI
- All UI structure is in code
- Views quickly get big and complex - making subviews gives modularity and reuse
- Previews allow you to see what the screen looks like in different conditions without running the app
- As interaction happens, the SwiftUI changes what the screen looks like

19

What we haven't seen yet

- Doing MVVM in SwiftUI
- Reacting to change in state where the data changes
- Making Forms

We will see all of that by building a second example app

20

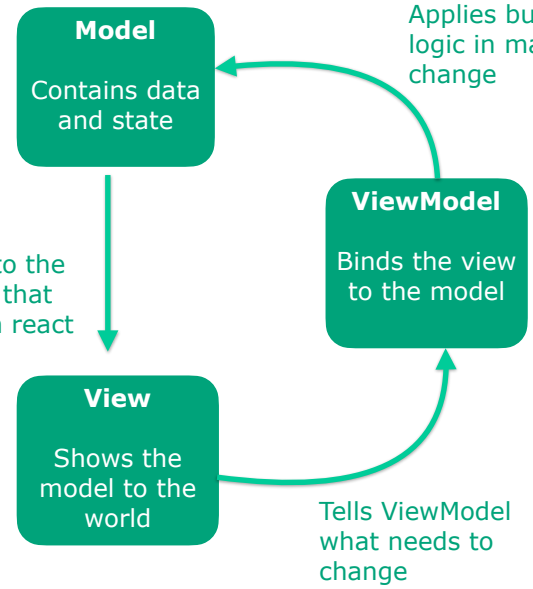
What is MVVM?

- In MVC (Model-View-Controller), the View is passive - all the effort of changing it is in the Controller
- This can lead to very large view controllers
- In MVVM (Model-View-ViewModel), you ideally have an active View - as changes happen to the Model, they are reflected in the View
- The View Model then has much less to do, and it can concentrate on the business logic of the app - checking that things are all done correctly, and producing model data in the form the View needs

21

Model-View-ViewModel example - getting list of conference speakers

Fairly simple for conference speakers app - details of the speakers, but for more complex apps, might include a lot more state



View is stateless - it says what the screen should look like in each state of the model. It is reactive - as the model changes, what the view shows will change

Deals with business logic of getting data (e.g. pulling it from database, or making network calls), and of making changes to the model when the user interacts with the view (e.g. storing a new speaker's details)
View does not change model directly.

22

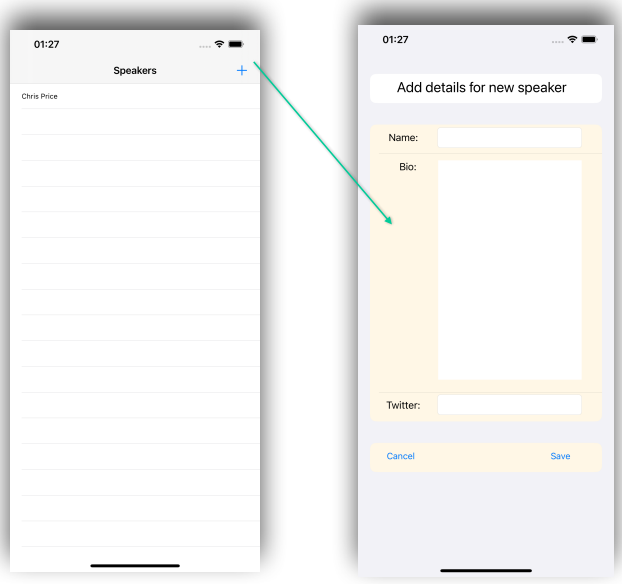
Where does SwiftUI come in?

- It makes the View reactive - it propagates data changes through the View
- We have seen this happen when the UI state changes (e.g. selecting a row in a table)
- It can also handle changes to data state
- Before Apple made SwiftUI, this could be done through a framework called RXSwift, but there are obvious benefits to doing it in Apple's own system

23

Second example SwiftUI app - make speaker data for Conference app

- This repeats the app we created in session 15, using SwiftUI, and applying MVVM
- Model concentrates on data, SwiftUI on View, and data manipulation goes into ViewModel



Speaker creator is provided in folder for session

- Model consists of Speaker and SpeakerList - they are mostly just the data and simple operations on it
- ViewModel does a lot of what we had in our singleton previously, including the SpeakerList that is made available to the View - it handles all the storing in JSON and the reading from JSON
- ContentView looks very similar to our last example, but it navigates to a Form where we can fill in the details of new speakers.
- When we add a speaker, main list is automatically updated

25

SwiftUI adoption - when should we do it?

- Depends on your updating strategy (how many years of past operating systems do you need to support):
  - iOS 13 SwiftUI is not usable for most apps
  - iOS 14 is much improved and fills in a lot of the gaps - most things can be done there and you can link to UIKit modules for the rest
  - iOS 15 looks great
- There is quite a learning curve on top of UIKit - we are preparing a three day upgrade course for people who use UIKit

26

Session 23: Final App

You have three choices to make:

1. What app will you build?
   - Provided app problem / Your own choice
2. What technology will you use?
   - UIKit / SwiftUI
3. Who will you build it with?
   - On your own / in a team

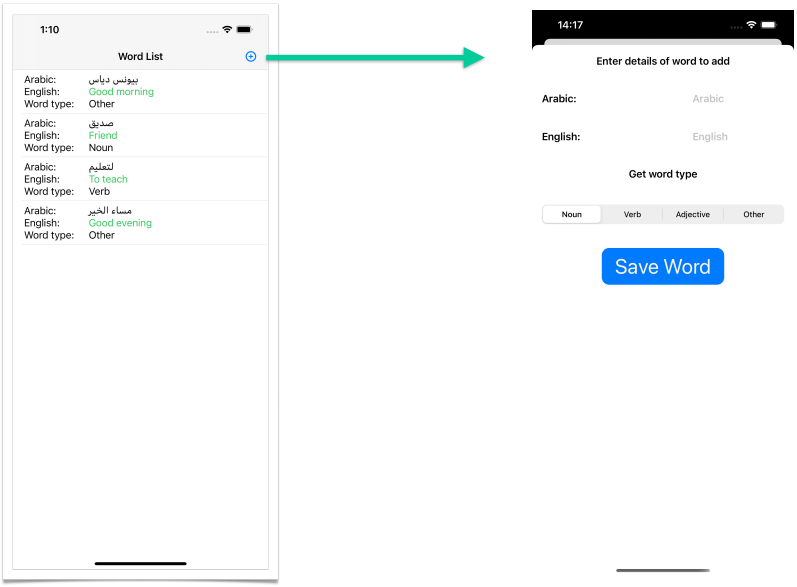
1

Q1: What app will you build?

- Arabic-English phrasebook (will explain next slide)
- Conference app from scratch
- Another app appropriate to your future work
- A specific technology that you want to master (maps, gestures, core data etc.)

2

New app problem: Arabic /English phrasebook



Characteristics of phrase book app

- You maintain an ordered list (alphabetic by Arabic) of words in Arabic and English, along with what kind of word they are.
- It starts with no words, and you add them as you learn new ones.
- They are stored persistently.
- Possible extensions:
  - You can order by Arabic or English
  - You can test yourself on words by showing the Arabic and asking for the English