# iOS development

Tiberiu Vilcu
Prepared for EECS 411
Sugih Jamin
11 September 2017

## Outline

1. Set up Xcode
2. Create sample app
3. Add UI to see how the builder works
4. Learn Swift basics to understand the code
5. Add some code to give app functionality
6. Run and test to verify and debug

## Xcode

- Modern IDE
- Swift/Objective-C/C(++) compiler (LLVM)
- LLDB debugger
- Apple product simulator (iOS, watchOS, tvOS)
- Capable of version control (git)
- Only on macOS

## Installing Xcode

1. Open App Store
2. Search + install Xcode
3. Open and install all other components it asks for

## Warnings about Swift / iOS

- Things change quickly in Swift!
- A lot of code online is outdated
- SDK's online also often outdated or use Objective-C
  You may need to build them yourselves
- Don't update to new macOS / iOS without entire team
  (Updates are scheduled for late September)

## Apple Developer account

You will eventually need an account for the class

- $99 for a one-year subscription
- Required to publish to App Store
- Gives access to betas
- Gives ability to run/save your app on actual phones

Consider splitting accounts with other teams to save costs

## Outline

1. Set up Xcode
2. Create sample app
3. Add UI to see how the builder works
4. Learn Swift basics to understand the code
5. Add some code to give app functionality
6. Run and test to verify and debug

## Sample project: Chatter

- Has view to write a post and submit to server
- Has timeline view to see posts
- Is cheaply inspired by Twitter
- Has a live web API already:
  http://159.203.172.151/getchatts/
  http://159.203.172.151/addchatt/
  (we will create this in a later lecture as well)

## Creating an iOS project

1. “New Xcode project” in startup screen
2. “iOS Single View Application”
3. Pick product name
4. Pick organization name and identifier (com.organization)
5. Language: Swift
6. Devices: iPhone (not both)
7. (Don’t need to include tests)
8. Create Git repository (later setup GitHub remote)

## Xcode window

Left: navigator pane (file explorer, warnings/errors, much more)

Middle: editor pane

- Standard editor shows one file
- Assistant editor shows two related files
- Version editor shows history

Right: utility pane (component attributes and libraries)

Bottom: debug pane (shows console output)

## General app settings

Display name: Chatter

Signing: add Apple Developer ID once created

Deployment target: latest iOS (that whole team has)

Main interface: Main (Main.storyboard)

Orientation: Portrait only recommended

Status bar style: default (black text on white background)

## Adding GitHub remote

1. Source control —> Chatter —> Configure...
2. Remotes tab, click plus, Add Remote
3. Paste GitHub remote link
4. Source control —> Commit
5. Add comment and can also push to remote
6. Add credentials when asked
7. Pull before working!

## Outline

1. Set up Xcode
2. Create sample app
3. Add UI to see how the builder works
4. Learn Swift basics to understand the code
5. Add some code to give app functionality
6. Run and test to verify and debug

## Storyboards

- Define user interfaces
- Are added to by drag-and-drop
- Main.storyboard: the default UI view
- LaunchScreen.storyboard: shows when app opens, until loaded
  Should look like the Main board but with unpopulated fields

## View Controllers

- Frame controllers that “manage” views
- Have different types: Table Views, Collection Views, Page Views...
- Add UI elements inside a view controller

## Navigation Controllers

- Manages a hierarchy of views
- Useful when using tab bar navigation (“forward” and back”)
- View controllers can be embedded in navigation controllers

(Note: different from Tab Bar Controllers; those are at bottom)

## Starting our timeline view

1. Replace start controller with new TableViewController, class ChattTableViewControler
2. Editor —> Embed In —> Navigation Controller
3. Add Bar Button Item to View Controller's Navigation Item
4. Edit button to say "Post" for now (Attributes, Title)
5. Add new View Controller, make class ComposeViewController
6. Control+drag from Post button to controller; "present modally segue"
7. Editor —> Embed In —> Navigation Controller (new controller)

## Adding to compose view

1. Add Bar Button Item to navigation to Submit
2. Add Label to top center and type a username
3. Add Text View to middle center
4. Give it some short sample text

## Adding to table view

1. Rename class to ChattTableCell and add an Identifer
2. Drag to resize vertically to visually see changes
3. Add label for username to top left
4. Add label for timestamp to top right
5. Add label for message to bottom left
   Make Lines: 0 and Line Break: Word Wrap

## Adding constraints

- Constraints define how elements are placed in the UI
- Auto Layout is on by default, leave that set and just use constraints
- (Leading and trailing mean left and right)
- Changed through buttons in bottom right of Storyboard editor

1. Reset to suggested constraints for username label
2. Reset for timestamp label as well (top=8, trailing=0)
3. Make message label bottom = bottom margin + 8
4. Make message label trailing = right margin + 0
5. Make message label top = username label bottom + 10

## Running on a simulator

1. Select the iPhone type on the top left (iPhone 7 is good)
2. (May need to download simulators separately to use them)
3. Click play to build and run
4. Simulator will boot iOS and load the app
5. Click to interact or use Hardware menu for physical buttons
6. Disable Hardware —> Keyboard —> Connect Hardware Keyboard to display the on-screen keyboard
7. Click stop button to close the app; Cmd+Q to close the simulator separately

## Outline

1. Set up Xcode
2. Create sample app
3. Add UI to see how the builder works
4. Learn Swift basics to understand the code
5. Add some code to give app functionality
6. Run and test to verify and debug

## Swift variables

- `let` defines a constant
- `var` defines a variable
- data types are inferred but can also be specified

Examples:

- `var myVar1 = 13`
- `let myConst1 = 20`
- `let myVar2: Double = 9`

## Data types

They have `Int, Double, Float, Bool, Strings`

Arrays and Dictionaries use brackets

```
var myDict = [“Hello”: “World”]

var myList: [Int] = [1, 2, 3]
```

## Control flow

The basics: `if, for, while, repeat-while`

Without parentheses (like Python)

```
for item in myDict { }

for i in 0…3 {// 0, 1, 2, 3}

for i in 0..<3 {// 0, 1, 2}
```

## Functions and closures

Functions: "first class objects" that can be passed around

Closures: refers to a function and the variables it "closes"

```
func sum(x: Int, y: Int) -> Int {
   return x + y
}

var sum: (Int, Int) -> (Int) = {return x + y}
```

## Optionals

- Optional types either have a value or are nil
- Types include `Int?, Double?, Bool?, String?`

```
var myVariable: Int? = 29
```

## Unwrapping optionals

```
var myVariable: Int? = 29
var myVariable2: Int
myVariable2 = myVariable!
```

! is dangerous! Fatal error if it finds nil

```
if let myVariable2 = myVariable {
// Runs here if it was not nil
}
```

## Outline

1. Set up Xcode
2. Create sample app
3. Add UI to see how the builder works
4. Learn Swift basics to understand the code
5. Add some code to give app functionality
6. Run and test to verify and debug

## App lifecycle

- Not running: not launched or terminated by system
- Inactive: in foreground, running code, not receiving events
- Active: in foreground, receiving events
- Background: not in foreground, running code, before suspension
- Suspended: not in foreground, not running code, still in memory

## UIViewController lifecycle

1. viewDidLoad(): called when controller is created
2. viewDidLayoutSubviews(): called whenever a subview needs to layout
3. viewWillAppear(): called when view is about to show, used to customize view settings
4. viewDidAppear(): called after the view shows
5. viewWillDisappear(): called before the view is removed
6. viewDidDisappear(): called after the view is in a disappeared state

## Calling function on button click

1. Copy ViewController class code for ComposeViewController
2. Use assistant editor view
3. Go to Submit button and Control+drag to inside class code
4. Give Action for Connection and a relevant name then connect
5. (Can set type to UIBarButtonItem)
6. `dismiss(animated: true, completion: nil)`

## Adding UI items to code

1. Control+drag item to Swift code (use assistant editor)
2. Leave outlet connection, give a good name and leave the type
3. Reference it by variable name in code

Be wary of renaming things. References can get broken.

## Packing data into JSON

```
let json: [String: Any] = ["username": self.usernameLabel.text ?? "NULL",
                           "message": self.messageTextView.text ??
                               "I wrote a blank message, oops!"]

let jsonData = try? JSONSerialization.data(withJSONObject: json)
```

`try?` returns the value or nil if it failed

## Creating an HTTP POST request

```
var request = URLRequest(url:
    URL(string: "http://159.203.172.151/addchatt/")!)
request.httpMethod = "POST"
request.httpBody = jsonData
```

## Running an async task

```
let task = URLSession.shared.dataTask(with: request)
{ data, response, error in
    guard let _ = data, error == nil else {
        print("NETWORKING ERROR")
        return
    }
    if let httpStatus = response as? HTTPURLResponse,
httpStatus.statusCode != 200 {
        print("HTTP STATUS: \(httpStatus.statusCode)")
        return
    }
}
task.resume()
```

## Making a class for a chatt

In a new Swift file, Chatt.swift

```
import UIKit

class Chatt {
    var username: String
    var message: String
    var timestamp: String

    init(username: String, message: String, timestamp: String) {
        self.username = username
        self.message = message
        self.timestamp = timestamp
    }
}
```

## Making a class for ChattTableCell

```
import UIKit

class ChattTableCell: UITableViewCell {

    @IBOutlet weak var messageLabel: UILabel!
    @IBOutlet weak var usernameLabel: UILabel!
    @IBOutlet weak var timestampLabel: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
    }
    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)
        // Configure the view for the selected state
    }
}
```

## Loading Chatts into memory

```
var chatts = [Chatt]()

func refreshChatts() {
        let requestURL = "http://159.203.172.151/getchatts/"
        var request = URLRequest(url: URL(string: requestURL)!)
        request.httpMethod = "GET"

        let task = URLSession.shared.dataTask(with: request) { data, response, error in
            guard let _ = data, error == nil else {
                print("NETWORKING ERROR")
                return
            }
            if let httpStatus = response as? HTTPURLResponse, httpStatus.statusCode != 200 {
                print("HTTP STATUS: \(httpStatus.statusCode)")
                return
            }

            do {
                var newChatts = [Chatt]()
                let json = try JSONSerialization.jsonObject(with: data!) as! [String:Any]
                let chattsReceived = json["chatts"] as? [[String]] ?? []
                for chattEntry in chattsReceived {
                    let chatt = Chatt(username: chattEntry[0], message: chattEntry[1], timestamp: chattEntry[2])
                    newChatts += [chatt]
                }
                self.chatts = newChatts
                self.tableView.estimatedRowHeight = 140
                self.tableView.rowHeight = UITableViewAutomaticDimension
                self.tableView.reloadData()
            }
            catch let error as NSError {
                print(error)
            }
        }
        task.resume()
    }
```

## Setting up the table view controller

numberOfSections: return how many sections are in data

numberOfRowsInSection: return how many rows (how many chatts)

didSelectRowAt: add behavior when a cell is tapped

cellForRowAtIndexPath: configure a single cell

prepareForSegue: where objects can be passed to next controller

# Adding table functions

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
    tableView.deselectRow(at: indexPath as IndexPath, animated: true)
}

override func numberOfSections(in tableView: UITableView) -> Int {
    return 1
}

override func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
    return chatts.count
}
```

# Adding table functions

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
        let cellIdentifier = "ChattTableCell"
        guard let cell = tableView.dequeueReusableCell(withIdentifier:
cellIdentifier, for: indexPath) as? ChattTableCell else {
            fatalError("The dequeued cell is not an instance of
ChattTableCell")
        }

        let chatt = chatts[indexPath.row]
        cell.usernameLabel.text = chatt.username
        cell.usernameLabel.sizeToFit()
        cell.messageLabel.text = chatt.message
        cell.messageLabel.sizeToFit()
        cell.timestampLabel.text = chatt.timestamp
        cell.timestampLabel.sizeToFit()
        return cell
    }
```

# Refreshing the table

Enable refreshing in TableViewController in Storyboard

```
self.refreshControl?.addTarget(self, action:
#selector(ChattTableViewController.handleRefresh(_:)), for:
UIControlEvents.valueChanged)
(In viewDidLoad() to allow the refresh control)
```

```
func handleRefresh(_ refreshControl: UIRefreshControl) {
    self.refreshChatts()
}
(In the class)

self.refreshControl?.endRefreshing()
(Whenever refreshChatts() returns)
```

# Outline

1. Set up Xcode
2. Create sample app
3. Add UI to see how the builder works
4. Learn Swift basics to understand the code
5. Add some code to give app functionality
6. Run and test to verify and debug

## Running on a simulator

1. Select the iPhone type on the top left (iPhone 7 is good)
2. (May need to download simulators separately to use them)
3. Click play to build and run
4. Simulator will boot iOS and load the app
5. Click to interact or use Hardware menu for physical buttons
6. Disable Hardware —> Keyboard —> Connect Hardware Keyboard to display the on-screen keyboard
7. Click stop button to close the app; Cmd+Q to close the simulator separately

## Possible improvements

- Error handling
- Carefully spaced layouts
- Ability to cancel posting
- Automatically add new post to table view through segue
- Refresh only new Chatts, don’t load everything again

## Things to consider

- Caching and storing data locally for long term use
- Efficient asynchronous requests to APIs
- Following Apple Human Interface UI guidelines
- Content restrictions on App Store

## Tips before you begin

- You can learn Swift as you go
- Plan your storyboard ahead of time; they are difficult to change/fix (but small aesthetic UI changes are fine)
- You will find bugs in Xcode; save/commit often and google problems
- Xcode can automatically convert old Swift code into new code (however it's not guaranteed to work or be correct)
- Don’t waste too much time on an app icon

# Resources

- Project sample repo: https://github.com/UM-EECS-441/ios-project-sample-f17
- Swift guide: https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/index.html#//apple_ref/doc/uid/TP40014097-CH3-ID0
- Apple Human Interface guidelines: https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/
- App life cycle: https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html
- View controllers: https://developer.apple.com/library/content/referencelibrary/GettingStarted/DevelopiOSAppsSwift/WorkWithViewControllers.html
- Apple developer enrollment: https://developer.apple.com/programs/enroll/