INFO1111: Computing 1A Professionalism

2023 Semester 1

Self-Learning Report

Submission number: 1

Github link: Click here to visit my project

Student name	Oscar Moore
Student ID	510549799
Topic	*Swift*
Levels already achieved	??
Levels in this report	A and B

Contents

1. Level A: Initial Understanding

1.1. Level A Demonstration

-Install and prepare an environment that supports swift
-Program and run a simple script
-Create a new model view in a separate script

1.2. Learning Approach

My approach to learning Swift mimics closely my approach to learning all programming languages. The first thing I did was first identify why I wanted to learn Swift. Prior to starting to study advanced computing at university, I had a clear goal of developing my own app. We live in a world that is heavily dominated by the apple ecosystem, hence, the logical next step was to choose a programming language that would work seamlessly with Apple's frameworks and tools.

The next step of my learning approach was understanding the basics of Swift. Understanding the basics of the language also included researching different environments that would support my project. Given that I was designing an app, Xcode would be the most appropriate platform to use, a necessity in the world of IOS app development.

In order to further research the language itself, I heavily relied on the magic of Chatgpt. The system's ability to scrape important information quickly and concisely, made the learning process seamless.

1.3. Challenges and Difficulties

Given the intuitive nature of SWIFT, I rarely ran into any technical difficulties (Other than a few thousand syntax errors). Having programmed in java and python before, Swift did not present a massive learning curve. With this being said, I ran into numerous challenges with regard to setting up the environment. Using the Xcode environment differed greatly from any IDE I had used before.

My largest challenge was the lack of resources or materials available for learning in swift. While there are many resources available for Swift, they are less abundant than resources for more established languages like Java and Python. When I started programming my project, I ran into a bug that was unique to the new IOS 16 system, which was rarely covered on the apple developer website due to the novelty of the new apple software. This took the progress of my project to an abrupt halt until an anonymous respondee on the apple developer website answered my thread question.

1.4. Learning Sources

Learning Source - What source did you use? (Note: Include source details such as links to websites, videos etc.). Contribution to Learning - How did the source contribute to your learning (i.e. what did you use the source for)?

Learning Source - What source did you use? (Note: Include source details such as links to websites, videos etc.).	Contribution to Learning - How did the source contribute to your learning (i.e. what did you use the source for)?
Swift youtube video: click here	Extremely helpful with regard to styling my swift project, also had good tips with how to organise class hierarchy.
Chatgpt	Helpful with debugging and solving syntax errors. Also gave me key functions that could be used based on use case prompts.
Apple developer	Very helpful for debugging code with errors that were not syntax.
Swift documentation: click here	Website used to access the swift documentation

1.5. Application artifacts

My initial product proposal for this project was to create a messaging platform at a very basic level. The issue with this initial outlay was that in order to have users communicate through my app, I would have to integrate additional software tools such as Firebase (to store user information in a database). I deduced that it would be impossible to complete the project without collaboration with other tools. I will only cover the swift components of my project in this section of the report, however, I am aware that contributions from other languages/applications would be needed to complete the entirety of the project.

My first objective was to create a landing page for the user. I added the Zeplin package to my project to further aid the design process. After a week of rigorous trial and error, I programmed the following:

Please proceed to the next page...

(a) preview of landing page

```
NavigationView{
    VStack {
        //MUST DOWNLOAD FONT ASSETS
        HStack(spacing: -45){
            Text("the")
                .foregroundColor(Color(red: 0/255.0, green: 0/255.0, blue: 0/255.0, opacity: 1))
                .frame(width: 122, height: 32)
                .font(Font.custom("Inter-Bold", size: 24))
            Text("Bubble")
                .foregroundColor(color)
                .font(Font.custom("Inter-Bold", size: 24))
        }
        .offset(x:-20, y:-250)

        VStack(alignment: .leading,spacing: 0){
            Text("Welcome, ")
                .foregroundColor(Color(red: 9/255.0, green: 10/255.0, blue: 10/255.0, opacity: 1))
                .font(Font.custom("Inter-Bold", size: 30))

            HStack(spacing: -45){
                Text("my nickname is")
                    .foregroundColor(Color(red: 0/255.0, green: 0/255.0, blue: 0/255.0, opacity: 1))
                    .font(Font.custom("Inter-Bold", size: 28))
                TextField(
                    "",
                    text: $name,
                    onCommit:{
                        loginSequence(var: name)
                        shouldShowNextView = true
                    })
                .offset(x:55)
                .foregroundColor(color)
                .font(Font.custom("Inter-Bold", size: 28))

            }
```

(b) Sinpet of code for landing page

Figure 1: The brunt of the information on the landing screen was included in a VStack as you can see above. This organized all my information vertically down the screen. In terms of the logic of the above code, there is very little. It is mostly formatting code so that it looks visually pleasing.

The only challenging part above was managing the login sequence of the user. When the user writes in their username and presses the 'enter' key, the loginsequence() function is called.

```
private func loginSequence(var nickname: String){
    let url = URL(string: "http://localhost:3000/login/hsis")!
    Auth.auth().signInAnonymously { authResult, error in
        guard let result = authResult, error == nil else {
            print("Failed to log in")
            return
        }
        let user = result.user
        print("Logged in user")
    }
}
```

Figure 2: Function code

This function basically registers the user into the database and authorises it to move to the next page. This is the first place that my program interacts with firebase, the user is given a unique identifier that is stored on the database. I decided to use the anonymous user authentication function that firebase provides since I wanted to avoid a long and tedious user registration process (The create account button on the display above was used in case the user wanted to save message history). If the login sequence is successful, the user is directed to the next swift page.

2. Level B: Basic Application

Whilst level A is about doing something simple with the topic to just show that you have started to be able to use the tool or technology, level B is about doing something practical that might actually be useful.

2.1. Level B Demonstration

In order to make the project described above actually useful, I would have to draw up an appropriate use case. I decided to slightly change the initial project outlay to include additional features that would make it a useful piece of software.

Although a messaging app by itself would be highly useful, there are hundreds of much better alternatives already accessible. I decided to add an extra component to my app:
-The app would consider the location of the user
-I would have pre-defined group chats that could only
be accessed when the user was physically within the pre-defined radius.
-The app would suggest group chats that fall within the physical radius of the user.

With the above requirements, the app would qualify as something that I would find useful to communicate with friends in my vicinity.

2.2. Application artifacts

Similar to the previous section, I will only provide artefacts that include code that I wrote in Swift since that is the topic I am self-learning.
The first step in providing the user with unique group chats based on location was by finding a way to pass information from a file into the Swift program. I decided to pass the parameters of the group chat (which I called bubbles in my app) using a JSON file, which my swift program read. The following screenshots include the JSON file with the necessary parameters for the 'bubble' and the config reader program I made to process this information.

Please proceed to following page...

```json
{
    "bubbles":[
        {
            "id": 1,
            "name": "St johns",
            "Range": 100,
            "Private": false
        },
        {
            "id": 2,
            "name": "Sydney Uni",
            "Range": 500,
            "Private": false
        },
        {
            "id": 2,
            "name": "The Grose",
            "Range": 300,
            "Private": false
        }
        ]
}
```

(a) json example

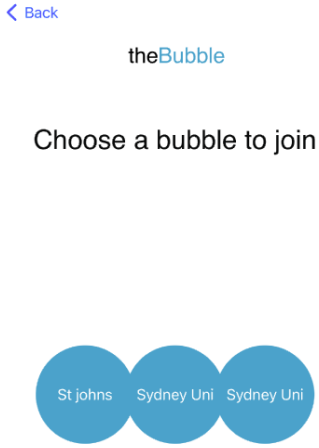
```swift
import SwiftUI

struct ResponseData: Decodable {
    var bubbles: [BubbleObject]
}
struct BubbleObject: Identifiable, Decodable {
        var id: Int
        var name: String
        var Range: Int
        var Private: Bool
}

func loadJson(filename fileName: String) -> [BubbleObject]? {
    if let url = Bundle.main.url(forResource: fileName, withExtension: "json") {
        do {
            let data = try Data(contentsOf: url)
            let decoder = JSONDecoder()
            let jsonData = try decoder.decode(ResponseData.self, from: data)
            return jsonData.bubbles
        } catch {
            print("error:\(error)")
        }
    }
    return nil
}
```

(b) snipet of code used to read JSON

Figure 3: The way this works: the Swift file finds the JSON file in its local repository and passes it through a LoadJson() function. This function returns the bubble object in the correct format as defined by the structure in the code.
The following code includes the logic used in order to output the bubbles onto the screen. I have also included a screenshot of what the preview of the app looked like for the respective page.



(a) preview of page

```swift
var body: some View {
    VStack{
        //MUST DOWNLOAD FONT ASSETS
        HStack(spacing: -45){
            Text("the")
                .foregroundColor(Color(red: 0/255.0, green: 0/255.0, blue: 0/255.0, opacity: 1))
                .frame(width: 122, height: 32)
                .font(Font.custom("Inter-Bold", size: 24))
                .onAppear{
                    bubbles = loadJson(filename: "bubble_data") ?? []
                }
            Text("Bubble")
                .foregroundColor(color)
                .font(Font.custom("Inter-Bold", size: 24))
        }
        .offset(x:-20, y:-250)

        Text("Choose a bubble to join")
            .foregroundColor(Color(red: 9/255.0, green: 10/255.0, blue: 10/255.0, opacity: 1))
            .font(Font.custom("Inter-Bold", size: 30))
            .offset(x:0, y:-200)
              .onAppear{
                  Render(BubbleList: bubbles)
              }
        HStack(spacing: -10){
              let render = Render(NumberOfCircles: CGFloat(bubbles.count))
            ForEach(bubbles) { bubble in
                BubbleCircle(name: bubble.name, size: 110)
            }
        }
```

(b) Sinpet of code for page

Figure 4: The way this works: the Swift file finds the JSON file in its local repository and passes it through a LoadJson() function. This function returns the bubble object in the correct format as defined by the structure in the code. The BubbleCircle function is called and returns the bubble circle view that you can see on the left hand side. It takes the name and size parameters and outputs the circles accordingly

3. Level C: Deeper Understanding

Level C focuses on showing that you have actually understood the tool or technology at a relatively advanced level. You will need to compare it to alternatives, identifying key strengths and weaknesses, and the areas where this tool is most effective.

3.1. Strengths

What are the key strengths of the item you have learnt? (50-100 words)

3.2. Weaknesses

What are the key weaknesses of the item you have learnt? (50-100 words)

3.3. Usefulness

Describe one scenario under which you believe the topic you have learnt could be useful. (50-100 words)

3.4. Key Question 1

Note: This question is in the table in the 'Self Learning: List of Topics' page on Canvas. (50-100 words)

3.5. Key Question 2

Note: This question is in the table in the 'Self Learning: List of Topics' page on Canvas. (50-100 words)

4. Level D: Evolution of skills

4.1. Level D Demonstration

This is a short description of the application that you have developed. (50-100 words). **IMPORTANT:** *You might wish to submit this as part of an earlier submission in order to obtain feedback as to whether this is likely to be acceptable for level D.*

4.2. Application artifacts

Include here a description of what you actually created (what does it do? How does it work? How did you create it?). Include any code or other related artefacts that you created (these should also be included in your github repository).

If you do include screengrabs to show what you have done then these should be annotated to explain what it is showing and what the application does.

4.3. Alternative tools/technologies

Identify 2 alternative tools/technologies that can be used instead of the one you studied for your topic. (e.g. if your topic was Python, then you might identify Java and Golang)

4.4. Comparative Analysis

Describe situations in which both your topic and each of the identified alternatives would be preferred over the others (100-200 words).