แนวทางในการจัดกิจกรรมภาคปฏิบัติ

หัวข้อการเรียนรู้ :
การสร้างส่วนติดต่อกับผู้ใช้แบบ Imperative และแบบ Declarative ด้วย Xcode

ระยะเวลาในการทำกิจกรรม: 180 นาที

วัตถุประสงค์ในการทำกิจกรรม :

1. การพัฒนาทักษะการใช้ Xcode เพื่อการสร้างส่วนติดต่อกับผู้ใช้ (UI)
2. บอกความแตกต่างในการพัฒนาส่วนติดต่อกับผู้ใช้ (UI) แบบเชิงสั่งการ (Imperative) ด้วย UIKit และแบบเชิงประกาศ (Declarative) ด้วย SwiftUI ได้
3. เกิดความเข้าใจเกี่ยวกับแนวคิดการออกแบบส่วนติดต่อผู้ใช้แบบวนซ้ำ (Iterative UI Design) และสามารถนำมาประยุกต์ใช้กับการสร้างส่วนติดต่อกับผู้ใช้โดยใช้ Xcode และ SwiftUI ได้

ความคิดรวบยอด:
กิจกรรมการเรียนรู้ชุดนี้มุ่งพัฒนาความเข้าใจของผู้เรียนเกี่ยวกับความแตกต่างของการออกแบบส่วนติดต่อผู้ใช้ (User Interface: UI) ระหว่าง **แนวคิดการพัฒนาแบบเชิงสั่งการ (Imperative) ใน UIKit** และ **แนวคิดแบบเชิงประกาศ (Declarative) ใน SwiftUI** ตลอดจนส่งเสริมให้ผู้เรียนตระหนักถึงความสำคัญของ**กระบวนการออกแบบ UI แบบวนซ้ำ (Iterative UI Design)** และการใช้ศักยภาพของเครื่องมือใน Xcode เช่น Live Preview, Device Simulation และ Accessibility Tools เพื่อสร้างประสบการณ์ผู้ใช้ (User Experience Design) ที่ดี โดยเน้นให้ผู้เรียนตระหนักถึงบทบาทของนักพัฒนาในการออกแบบที่ต้องคำนึงถึงความต้องการและความหลากหลายของผู้ใช้ในทุกขั้นตอนของการพัฒนาแอปพลิเคชัน

บทความสำหรับอ่านประกอบการทำกิจกรรม
https://ajthiti.gitbook.io/develop-in-swift/getting-started/hello-swiftui

แนวทางในการจัดกิจกรรมการเรียนรู้

ขั้นนำเข้าสู่บทเรียน (Warm-up)

ในช่วงเริ่มต้นของบทเรียน ครูควรสร้างความสนใจและเชื่อมโยงบทเรียนกับประสบการณ์ใกล้ตัวของผู้เรียน โดยให้ผู้เรียนหยิบโทรศัพท์มือถือของตนขึ้นมาแล้วเปิดแอปที่ใช้งานเป็นประจำ เช่น LINE, TikTok หรือ YouTube จากนั้นชวนผู้เรียนสำรวจว่า

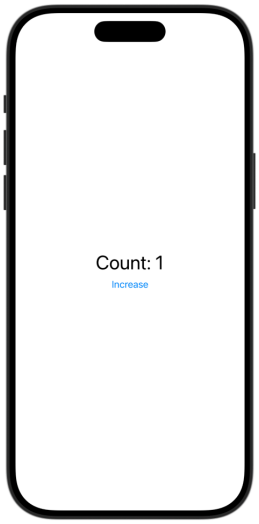
- องค์ประกอบใดบนหน้าจอที่ดึงดูดความสนใจเป็นอันดับแรก
- ปุ่มหรือข้อความใดที่ทำให้รู้ว่าจะต้องกดอะไรต่อไป
- อะไรเป็นปัจจัยที่ช่วยให้เราสามารถใช้งานแอปเหล่านั้นได้ โดยไม่จำเป็นต้องอ่านคู่มือ

ครูอาจตั้งคำถาม เช่น “ถ้าแอปนี้ไม่มี UI จะเกิดอะไรขึ้น?” เพื่อช่วยให้ผู้เรียนตระหนักว่า UI คือ เครื่องมือสื่อสารสำคัญระหว่างผู้ใช้และระบบ รวมทั้งเป็นปัจจัยสำคัญที่ทำให้แอปน่าใช้หรือไม่น่าใช้ จากนั้น ครูอาจให้ผู้เรียนลองวาด “หน้า Welcome Screen” แบบง่ายๆ บนกระดาษ เพื่อให้เห็นว่า การออกแบบ UI เป็นงานสร้างสรรค์ และเป็นจุดเริ่มต้นสู่การเรียนรู้เกี่ยวกับการใช้ UIKit และ SwiftUI เพื่อการออกแบบส่วนติดต่อกับผู้ใช้

ขั้นการเรียนรู้ (Learn)

เมื่อผู้เรียนมีความสนใจและเข้าใจบริบทเบื้องต้นแล้ว ครูสามารถเริ่มอธิบายแนวคิดสำคัญเกี่ยวกับการสร้างส่วนติดต่อผู้ใช้ โดยเน้นความแตกต่างระหว่าง **การพัฒนา UI แบบ Imperative ผ่าน UIKit** และ **การพัฒนา UI แบบDeclarative ผ่าน SwiftUI** ผ่านกิจกรรมการสร้าง Couter App ดังต่อไปนี้

กิจกรรมที่ 1 การพัฒนา Counter App ด้วย UIKit และ SwiftUI

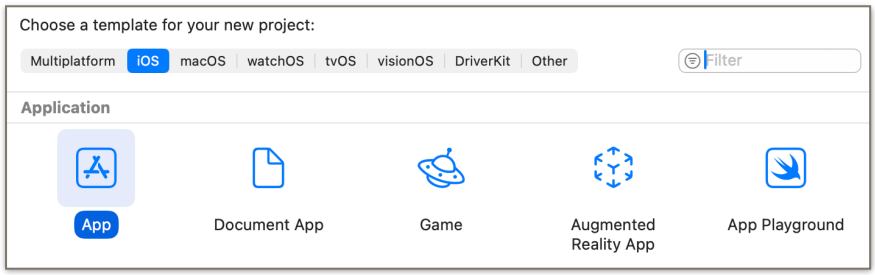


การพัฒนา UI แบบ Imperative ผ่าน UIKit คือ แนวทางการออกแบบและสร้างส่วนติดต่อผู้ใช้ที่อาศัยการ “สั่งงานเป็นลำดับขั้นตอน” (step-by-step instructions) ซึ่งนักพัฒนาต้องกำหนดรายละเอียดของทุกองค์ประกอบด้วยตนเอง ตั้งแต่การสร้างวัตถุ UI การกำหนดคุณสมบัติ การจัดวางตำแหน่ง ไปจนถึงการอัปเดตหน้าจอเมื่อข้อมูลเปลี่ยนแปลง

แนวคิดนี้สะท้อนรูปแบบการเขียนโปรแกรมเชิงสั่งการ (Imperative Programming) ที่ให้ความสำคัญกับ “วิธีการทำงาน” (How to do) มากกว่า “ผลลัพธ์ที่ต้องการ” (What it should look like)

การสร้างส่วนติดต่อกับผู้ใช้ด้วยการเขียนโปรแกรมด้วยตนเอง (UIKit - Programmatic)

1. เปิด Xcode และสร้าง Project ใหม่ โดยเลือก Template เป็น **iOS > App**

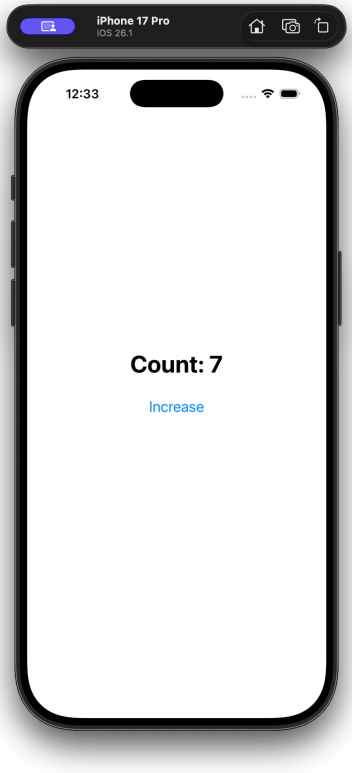


2. ตั้งชื่อ Project เป็น **UIKitDemo01**
   - กำหนด Organization Identifier เพื่อสร้าง Bundle ID ตามที่ต้องการ
   - กำหนดค่า Interface เป็น **Storyboard**
   - กำหนดค่า Language เป็น **Swift**
   - กำหนดค่า Testing System และ Storage เป็น **None**

3. เขียนคำสั่งที่ไฟล์ **ViewController.swift** ดังนี้

```swift
import UIKit

class ViewController: UIViewController {

    private var count = 0
    // การสร้าง UI ด้วย UILabel เพื่อใช้ในการแสดงข้อความ Count: 0
    private let countLabel: UILabel = {
        let label = UILabel()
        label.text = "Count: 0"
        label.font = UIFont.systemFont(ofSize: 32, weight: .bold)
        label.translatesAutoresizingMaskIntoConstraints = false
        return label
    }()

    // การสร้างปุ่ม Increase
    private let increaseButton: UIButton = {
```

```
        let button = UIButton(type: .system)
        button.setTitle("Increase", for: .normal)
        button.titleLabel?.font = UIFont.systemFont(ofSize: 20)
        button.translatesAutoresizingMaskIntoConstraints = false
        return button
    }()


    override func viewDidLoad() {
        super.viewDidLoad()
        view.backgroundColor = .systemBackground

        //เพิ่ม subviews
        view.addSubview(countLabel)
        view.addSubview(increaseButton)

        //จัดการ Auto Layout
        NSLayoutConstraint.activate([
            countLabel.centerXAnchor.constraint(equalTo: view.centerXAnchor),
            countLabel.centerYAnchor.constraint(equalTo: view.centerYAnchor, constant: -40),
            increaseButton.centerXAnchor.constraint(equalTo: view.centerXAnchor),
            increaseButton.topAnchor.constraint(equalTo: countLabel.bottomAnchor, constant: 20)
        ])


        //กำหนด event ให้ปุ่ม
        increaseButton.addTarget(self, action: #selector(increaseTapped),
for: .touchUpInside)
    }


    @objc private func increaseTapped() {
        //ปรับค่าตัวแปร state
        count += 1

        //อัปเดต UI ด้วยตนเอง (Imperative)
        countLabel.text = "Count: \(count)"
    }

}
```

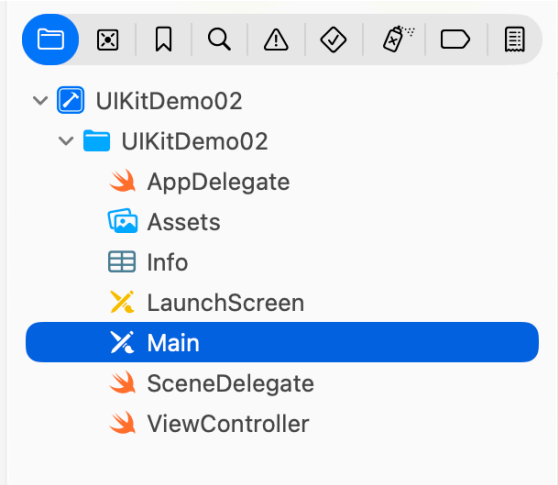


4. ทดลองรัน โดยกำหนด Simulator เป็นอุปกรณ์ที่ต้องการ

ผู้สอนอธิบายเพิ่มเติม เพื่อให้ผู้เรียนเกิดความเข้าใจเกี่ยวกับ
- การสร้าง Label และ Button เพื่อใช้ในการโต้ตอบกับผู้ใช้
- การเพิ่ม Label และ Button ลงใน View และกำหนดตำแหน่งในการจัดวาง ด้วย Auto Layout
- การกำหนด Event เพื่อเรียกใช้ฟังก์ชัน increaseTapped() และการอัพเดต ค่าที่แสดงบน Label เมื่อผู้ใช้กดปุ่ม Increase

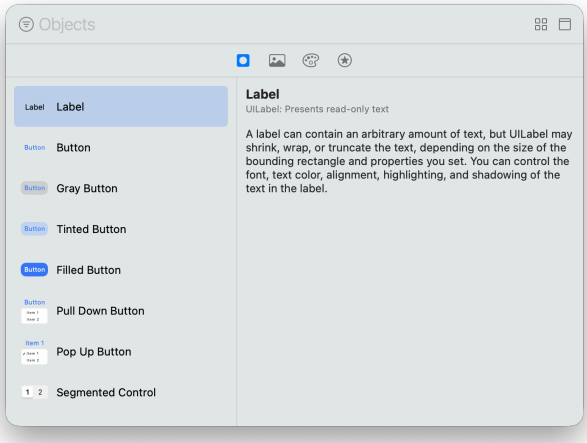
การสร้างส่วนติดต่อกับผู้ใช้ด้วย Interface Builder และการใช้ @IBOutlet/@IBAction

1. เปิด Xcode และสร้าง Project ใหม่ โดยเลือก Template เป็น **iOS > App**
2. ตั้งชื่อ Project เป็น **UIKitDemo02**
   - กำหนด Organization Identifier เพื่อสร้าง Bundle ID ตามที่ต้องการ
   - กำหนดค่า Interface เป็น **Storyboard**
   - กำหนดค่า Language เป็น **Swift**
   - กำหนดค่า Testing System และ Storage เป็น **None**

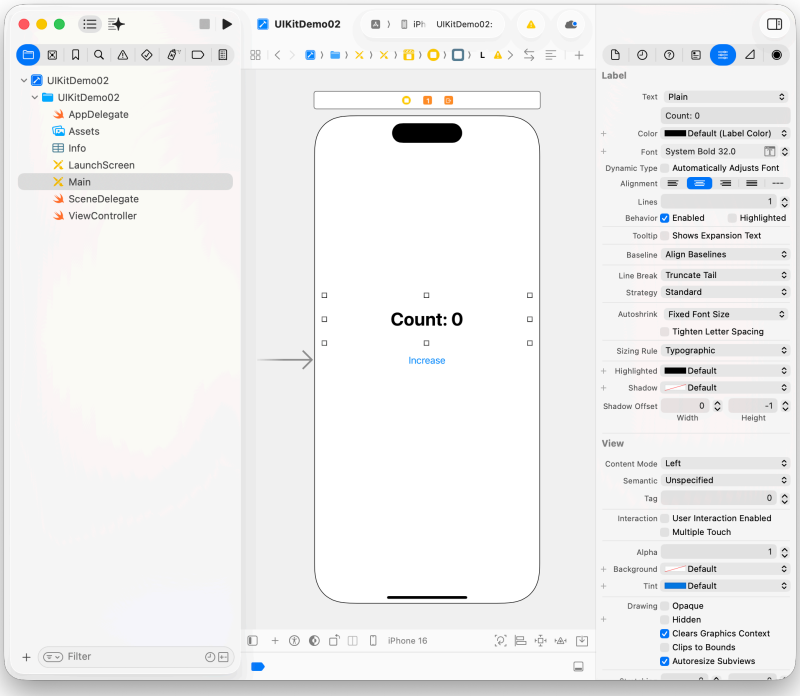


3. คลิกเลือกที่ไฟล์ **main.storyboard** ใน Navigator Area เพื่อออกแบบหน้าจอด้วย Interface Builder

4. คลิกที่ + หรือกด Shift + Command + L เพื่อเปิด **Objects Library**

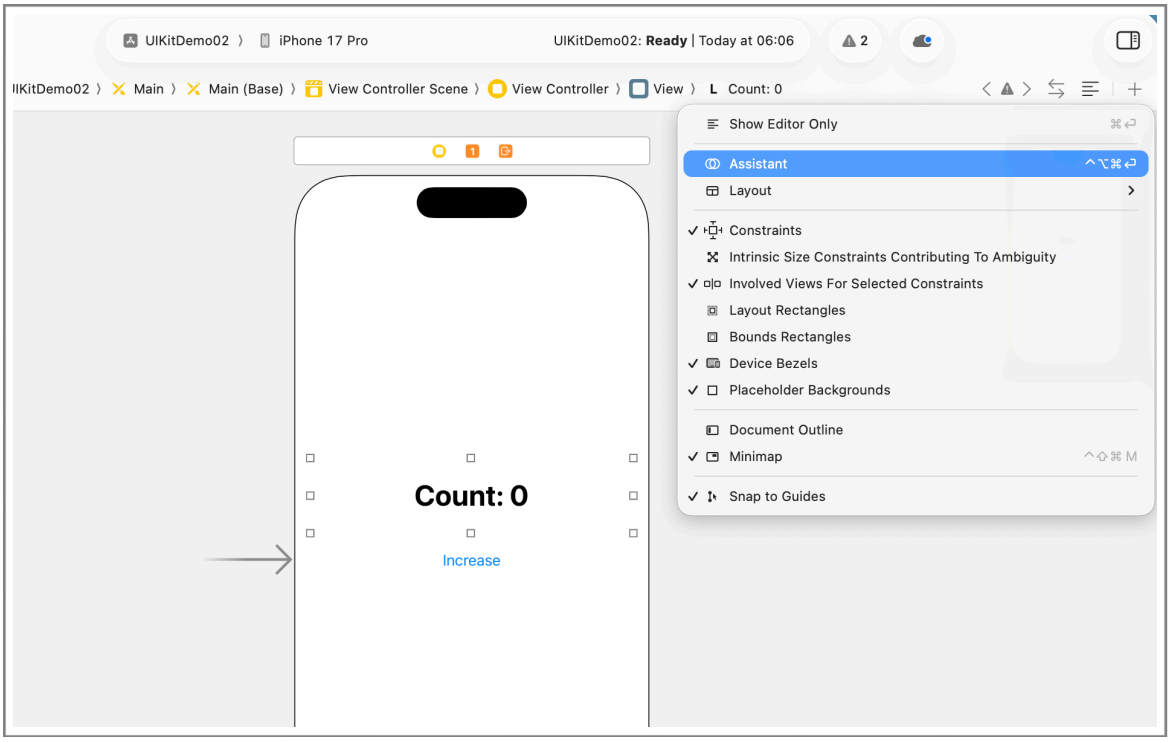


5. นำ Label และ Button มาวางบนจอภาพ



6. กำหนดรูปแบบการแสดงผลของ Label และ Button ด้วย Attribute Inspector

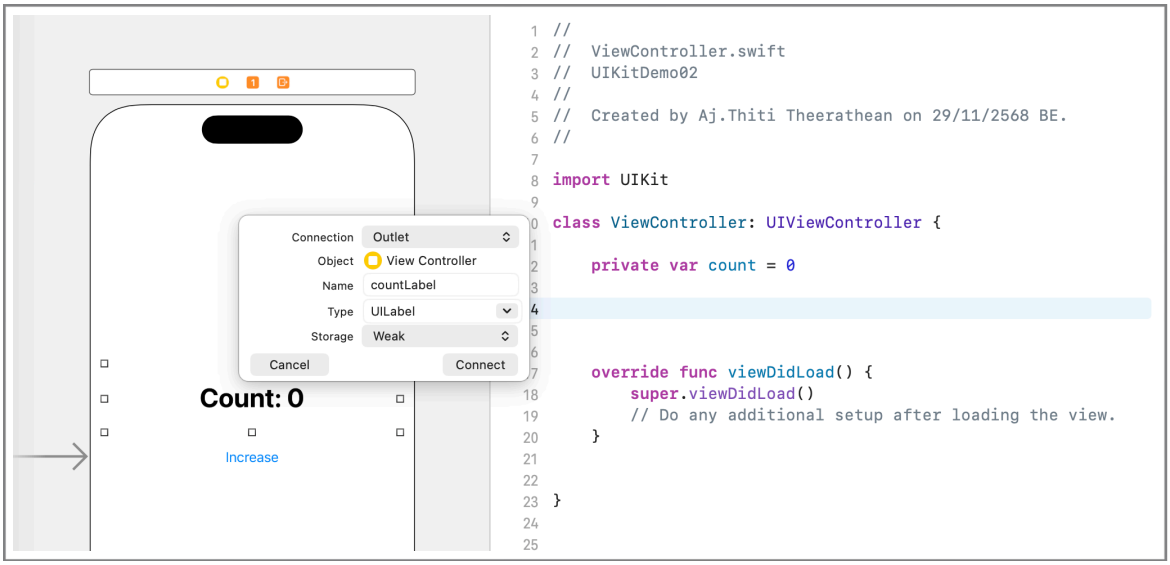
7. เปิด **Assistant Editor** เพื่อเปิดไฟล์ ViewController.swift ขึ้นมาคู่กับ Interface Builder



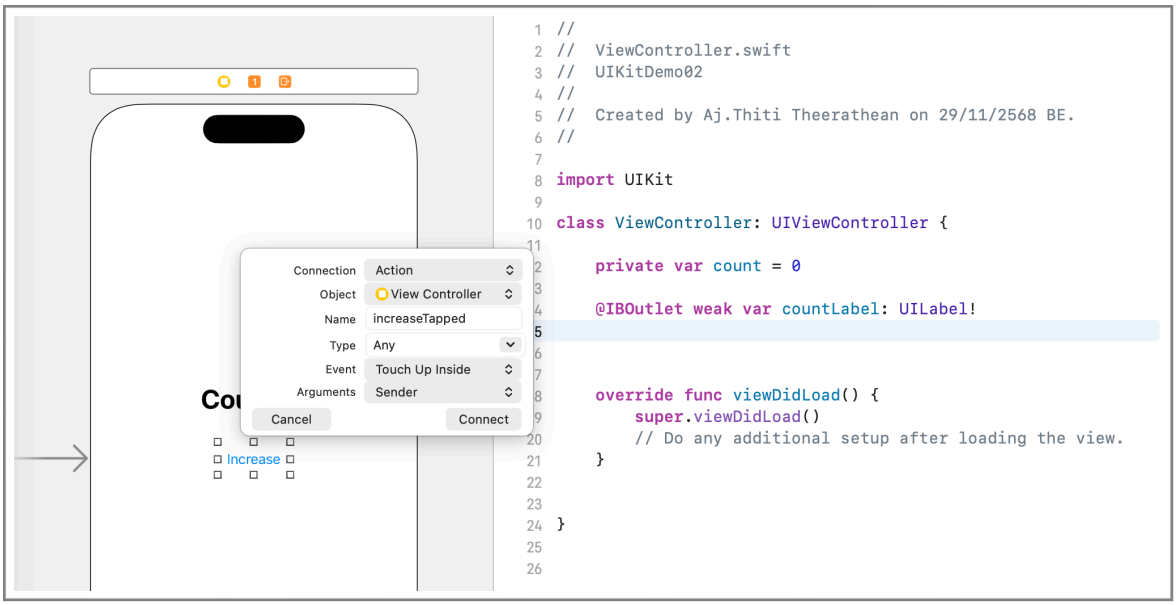
8. สร้างตัวแปรสำหรับเก็บค่าการนับ

```
private var count = 0
```

9. สร้าง **@IBOutlet** ที่ Label โดยตั้งชื่อว่า **countLabel**



10. สร้าง **@IBAction** ที่ ปุ่ม Increase โดยตั้งชื่อว่า **increaseTapped**




```swift
//
//  ViewController.swift
//  UIKitDemo02
//
//  Created by Aj.Thiti Theerathean on 29/11/2568 BE.
//

import UIKit

class ViewController: UIViewController {

    private var count = 0

    @IBOutlet weak var countLabel: UILabel!



    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
    }


}
```

11. สร้าง **Helper Methods** เพื่อใช้ในการอัพเดตข้อมูลใน countLabel

```swift
private func updateUI() {
     countLabel.text = "Count: \(count)"
}
```

12. สร้างคำสั่งใน **increaseTapped( )** ดังนี้

```swift
@IBAction func increaseTapped(_ sender: Any) {
    count += 1
    updateUI()
}
```

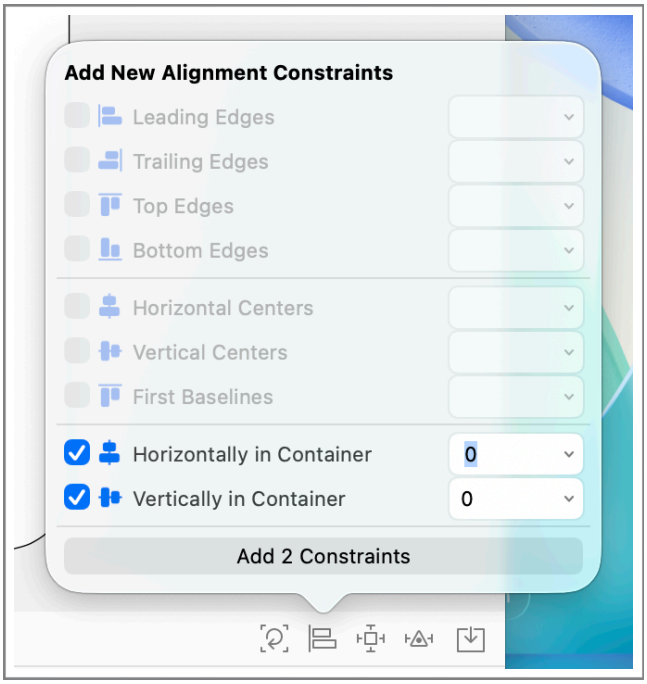
13. กำหนดค่าเริ่มต้นเมื่อโหลด View ในฟังก์ชัน **viewDidLoad( )**
    โดย กำหนดค่า view.backgroundColor เป็น .systemBackground
    ซึ่งเป็น Dynamic System Color ที่ออกแบบมาเพื่อให้ UI ปรับสีอัตโนมัติตามสภาพแวดล้อม
    ของระบบ เช่น Light Mode หรือ Dark Mode

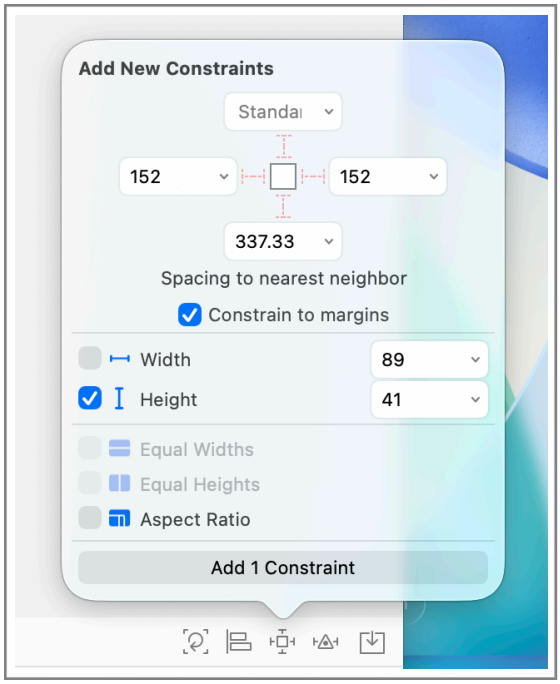
```swift
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view.
    view.backgroundColor = .systemBackground
    updateUI()
}
```

สรุปคำสั่งทั้งหมดในไฟล์ ViewController.swift เป็นดังนี้

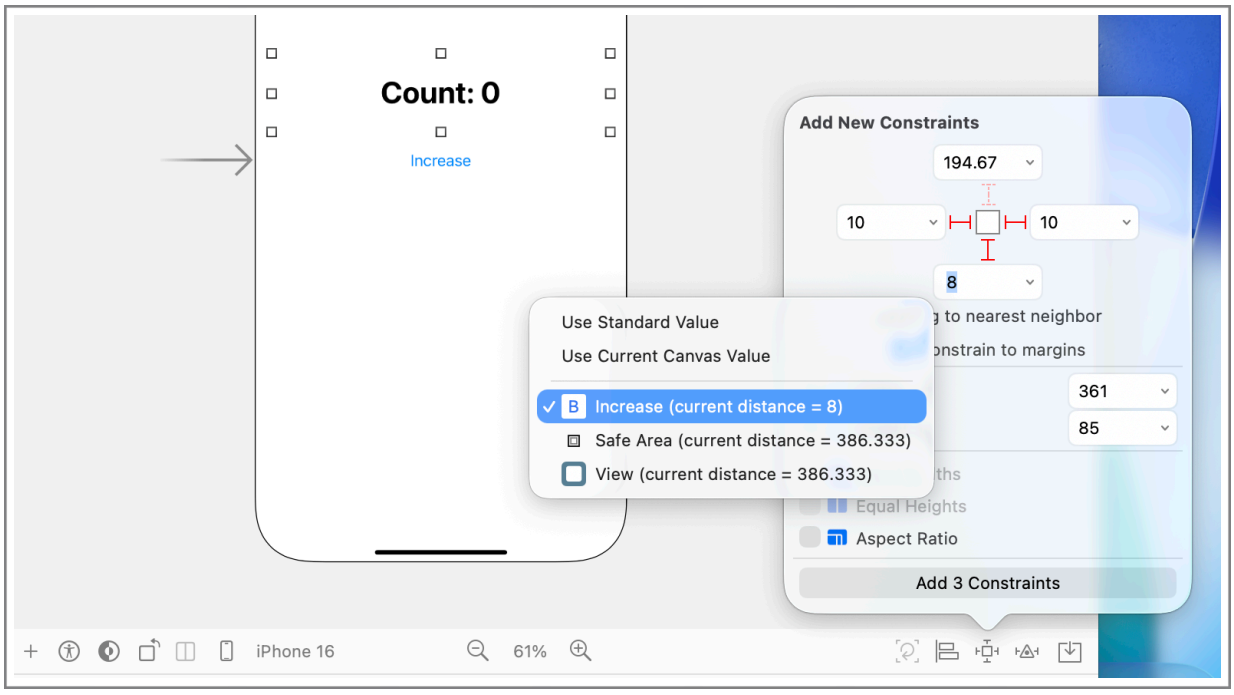
```
import UIKit

class ViewController: UIViewController {

    private var count = 0

    @IBOutlet weak var countLabel: UILabel!

    @IBAction func increaseTapped(_ sender: Any) {
        count += 1
        updateUI()
    }


    // MARK: - Lifecycle
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
        view.backgroundColor = .systemBackground
        updateUI()
    }


    // MARK: - Helper Methods
    private func updateUI() {
        countLabel.text = "Count: \(count)"
    }

}
```

14. เลือก **ปุ่ม Increase** เพื่อกำหนดขนาดและตำแหน่งการจัดวางด้วย AutoLaout



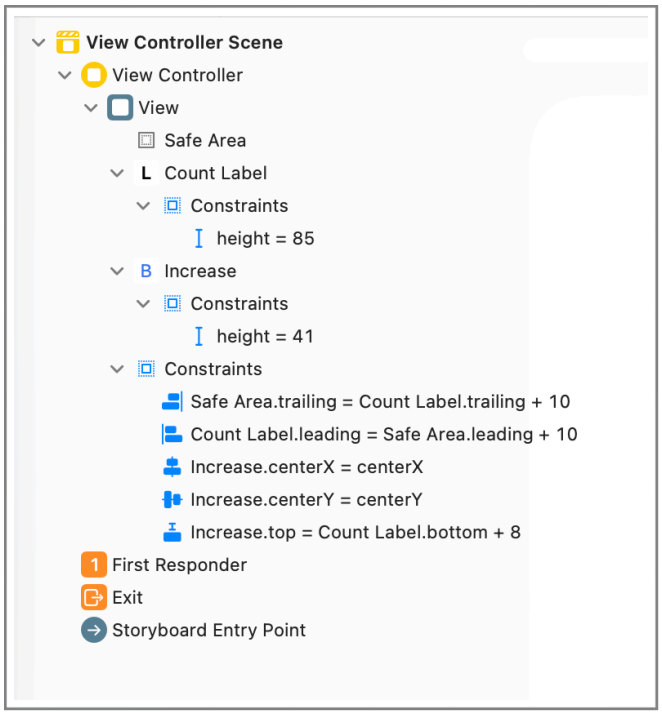


15. เลือก **Label** เพื่อกำหนดขนาดและตำแหน่งการจัดวางด้วย AutoLaout



โดยกำหนดค่าต่างๆ ดังนี้

- ค่า Leading และ Trailing ให้ห่างจากขอบ safe area เท่ากับ 10
- ค่า Bottom ให้ห่างจากปุ่ม Increase เท่ากับ 8
- ค่า Height ของปุ่มเท่ากับ 85

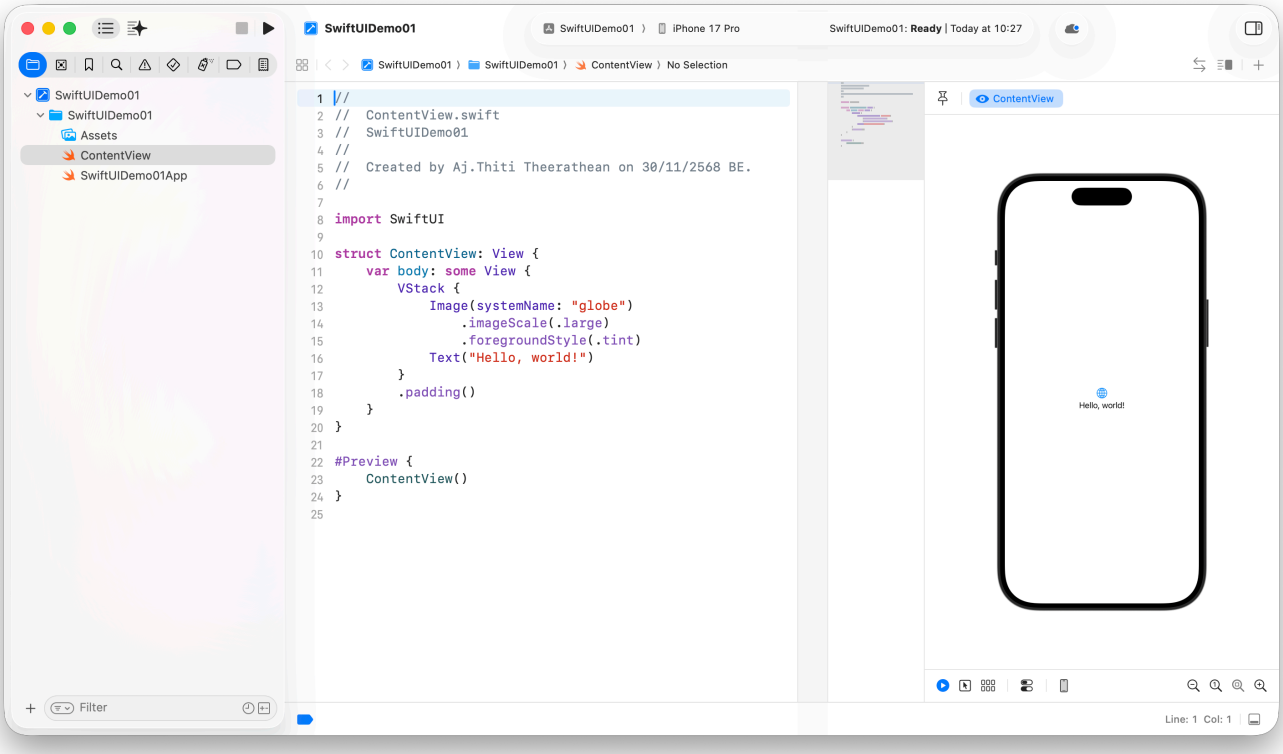


ตรวจสอบการกำหนดค่าต่างๆ ใน Document Outline

15. ทดลองรัน โดยกำหนด Simulator เป็นอุปกรณ์ที่ต้องการ

การสร้างส่วนติดต่อกับผู้ใช้ด้วย SwiftUI

1. เปิด Xcode และสร้าง Project ใหม่ โดยเลือก Template เป็น **iOS > App**
2. ตั้งชื่อ Project เป็น **SwiftUIDemo01**
   - กำหนด Organization Identifier เพื่อสร้าง Bundle ID ตามที่ต้องการ
   - กำหนดค่า Interface เป็น **SwiftUI**
   - กำหนดค่า Language เป็น **Swift**
   - กำหนดค่า Testing System และ Storage เป็น **None**



3. สร้างชุดคำสั่ง ดังนี้

```
import SwiftUI

struct ContentView: View {

    @State private var count = 0

    var body: some View {
        VStack {
            Text("Count: \(count)")
                .font(.largeTitle)

            Button("Increase") {
                count += 1
            }
        }
        .padding()
    }
}
```

กิจกรรมที่ 2 การออกแบบ Welcome Screen

การออกแบบ Welcome Screen สำหรับแอปพลิเคชันเป็นกระบวนการสำคัญที่กำหนดความประทับใจแรกของผู้ใช้ และเป็นจุดเริ่มต้นของการสื่อสารว่า แอปนั้นต้องการให้ผู้ใช้ทำอะไรต่อ การนำแนวคิด **Iterative UI Design** มาประยุกต์ใช้ร่วมกับ **SwiftUI** ช่วยให้กระบวนการสร้างหน้าจอนี้มีความยืดหยุ่น มีประสิทธิภาพ และสอดคล้องกับหลักการออกแบบสมัยใหม่ที่เน้นการปรับปรุงอย่างต่อเนื่องเพื่อให้ตอบโจทย์ผู้ใช้ได้มากที่สุด

การออกแบบ Welcome Screen ด้วย Iterative UI Design และ SwiftUI

1. เปิด Xcode และสร้าง Project ใหม่ โดยเลือก Template เป็น **iOS > App**
2. ตั้งชื่อ Project เป็น **WelcomeScreen**
   - กำหนด Organization Identifier เพื่อสร้าง Bundle ID ตามที่ต้องการ
   - กำหนดค่า Interface เป็น **SwiftUI**
   - กำหนดค่า Language เป็น **Swift**
   - กำหนดค่า Testing System และ Storage เป็น **None**

3. **Iteration 1** — สร้าง “โครงสร้างพื้นฐาน” ของหน้าจอ (Core Structure)

```
struct ContentView: View {
    var body: some View {
        VStack() {
            Text("Welcome")
            Text("Start your journey with our app.")
        }
        .padding()
    }
}

#Preview {
    ContentView()
}
```

4. **Iteration 2** — ปรับปรุงด้านการมองเห็น (Visual Enhancement)

```
  VStack(spacing: 20) {
      Text("Welcome")
          .font(.largeTitle)
          .bold()

      Text("Start your journey with our app.")
          .font(.subheadline)
          .foregroundStyle(.secondary)
          .multilineTextAlignment(.center)
          .padding(.horizontal, 32)

  }
  .padding()
```

5. **Iteration 3** — การเพิ่มการตอบสนองกับผู้ใช้งาน (Interactivity) ด้วยปุ่มกด

```
struct ContentView: View {

    @State private var showMessage = false

    var body: some View {
        VStack(spacing: 20) {
            Text("Welcome")
                .font(.largeTitle)
                .bold()

            Text("Start your journey with our app.")
                .font(.subheadline)
                .foregroundStyle(.secondary)
                .multilineTextAlignment(.center)
                .padding(.horizontal, 32)

            Button {
                showMessage = true      // เปลี่ยน state เมื่อกดปุ่ม
            } label: {
                Text("Get Started")
                    .padding(.vertical, 12)
                    .padding(.horizontal, 40)
                    .background(.blue)
                    .foregroundStyle(.white)
                    .cornerRadius(22)
            }
            .buttonStyle(.plain)
        }
        .padding()
        .alert("You tapped Get Started", isPresented: $showMessage) {
            Button("OK", role: .cancel) { }
        }
    }
}
```

6. **Iteration 4** — ทดสอบการใช้งาน (Usability Testing) โดยทดสอบการทำงานในส่วน Dark Mode และการแสดงผลบนอุปกรณ์ต่างๆ ที่ขนาดของจอภาพที่แตกต่างกัน

```
#Preview("iPhone 17 Pro (Dark)") {
     ContentView()
          .preferredColorScheme(.dark)
}
```

7. **Iteration 5** — การปรับแต่งเพื่อความสมบูรณ์ (Refinement) และเพิ่มการรองรับการทำงานผ่านระบบ VoiceOver สำหรับผู้ใช้มีความบกพร่องทางสายตา

```
import SwiftUI

struct ContentView: View {

    @State private var showMessage = false

    var body: some View {
```

```
            ZStack {
                // พื้นหลังแบบ gradient ดูมีมิติขึ้น
                LinearGradient(
                    colors: [.blue.opacity(0.8), .purple.opacity(0.8)],
                    startPoint: .topLeading,
                    endPoint: .bottomTrailing
                )
                .ignoresSafeArea()

                VStack(spacing: 24) {
                    Spacer()

                    VStack(spacing: 20) {
                        Text("Welcome")
                            .font(.largeTitle)
                            .fontWeight(.bold)
                            .foregroundStyle(.white)
                            .multilineTextAlignment(.center)
                            .accessibilityAddTraits(.isHeader)

                        Text("Start your journey with our app.\n
                              Let’s build something great together.")
                            .font(.body)
                            .foregroundStyle(.white.opacity(0.9))
                            .multilineTextAlignment(.center)
                            .padding(.horizontal, 32)
                    }

                    Spacer()

                    Button {
                        showMessage = true
                    } label: {
                        Text("Get Started")
                            .font(.headline)
                            .frame(maxWidth: .infinity)
                            .padding(.vertical, 12)
                            .background(.white)
                            .foregroundStyle(.blue)
                            .cornerRadius(22)
                            .shadow(radius: 4)
                    }
                    .padding(.horizontal, 32)
                    .padding(.bottom, 40)
                    .accessibilityLabel("Get Started")
                    .accessibilityHint("Double tap to begin using the app.")
                }
            }
            .alert("You tapped Get Started", isPresented: $showMessage) {
                Button("OK", role: .cancel) { }
            }
        }
    }
```

ขั้นสรุปบทเรียน (Conclusion)

ภายหลังการทำกิจกรรมเกี่ยวกับการออกแบบส่วนติดต่อกับผู้ใช้และการสร้างแอปด้วย UIKit, SwiftUI และ Iterative UI Design ผู้สอนควรสรุปให้ผู้เรียนเข้าใจว่า การพัฒนา UI ไม่ได้เป็นเพียงการเขียนโค้ดเพื่อให้แอปแสดงผลเท่านั้น แต่เป็นกระบวนการคิดเชิงออกแบบที่ต้องคำนึงถึงความต้องการของผู้ใช้เป็นหลัก ในบทเรียนนี้ นักเรียนได้เห็นความแตกต่างระหว่างการสร้าง UI แบบ Imperative ผ่าน UIKit

ซึ่งต้องกำหนดขั้นตอนการสร้างและอัปเดตหน้าจอด้วยตนเองทุกส่วน กับ แนวคิดแบบ Declarative ของ SwiftUI ที่ให้นักพัฒนาระบุว่า UI ควรมีหน้าตาอย่างไร แล้วระบบจะจัดการรายละเอียดภายในให้โดยอัตโนมัติ นอกจากนี้ ผู้เรียนยังได้เรียนรู้เกี่ยวกับกระบวนการ Iterative UI Design และตะหนักว่า UI ที่ดีต้องเกิดจากการทดลอง การทดสอบ และการปรับปรุงซ้ำหลายรอบ

คุณสมบัติ Live Preview และ Simulator ใน Xcode เป็นเครื่องมือสำคัญในกระบวนการ Iterative UI Design และกิจกรรม Welcome Screen ที่ผู้เรียนได้ลงมือทำ ยังช่วยให้เข้าใจบริบทจริงของการออกแบบ UI ตั้งแต่การวางโครงสร้างหน้าจอ การเลือกสื่อ การจัดลำดับข้อมูล ไปจนถึงการคำนึงถึง Accessibility เพื่อให้แอปสามารถถูกใช้งานได้กับผู้ใช้ทุกกลุ่ม

ท้ายที่สุด ผู้สอนควรเน้นย้ำว่าทักษะสำคัญที่ผู้เรียนได้รับ ไม่จำกัดแค่ความสามารถในการเขียนโค้ด แต่รวมถึงการคิดอย่างมีระบบ การออกแบบอย่างมีเหตุผล การมอง UI จากมุมมองผู้ใช้ และการใช้แนวคิดเชิงวนซ้ำในการพัฒนาผลงาน ซึ่งเป็นพื้นฐานสำคัญของการพัฒนาซอฟต์แวร์สมัยใหม่ และสามารถต่อยอดไปสู่การสร้างแอปที่มีคุณภาพและตอบโจทย์การใช้งานได้จริงในอนาคต

คำถามเพื่อการสะท้อนผลการเรียนรู้จากการทำกิจกรรม

1. นักเรียนพบความท้าทายหรือข้อจำกัดใดบ้าง ในกิจกรรมการสร้าง UI ด้วย UIKit แบบ Programmatic ?
2. การใช้ Interface Builder (IB) ร่วมกับ @IBOutlet และ @IBAction ช่วยลดภาระการเขียนโค้ดได้อย่างไร และยังมีข้อจำกัดใดอยู่บ้าง เมื่อเทียบกับการใช้ SwiftUI ?
3. การพัฒนาส่วนติดต่อกับผู้ใช้ (UI) ในรูปแบบ Imperative (UIKit) และแบบ Declarative (SwiftUI) มีความแตกต่างกันอย่างไร ?
4. การออกแบบ UI แบบ Iterative แตกต่างจากการออกแบบครั้งเดียวจบ (One-shot Design) อย่างไร และเหตุใดแนวคิดแบบ Iterative จึงให้ผลลัพธ์ต่อการสร้างประสบการณ์ของผู้ใช้ (UX) ที่ดีกว่าในระยะยาว ?
5. การใช้ Live Preview ใน SwiftUI ช่วยให้นักพัฒนาสามารถออกแบบ UI ด้วยแนวคิด Iterative UI Design ได้อย่างไร ?
6. นักเรียนคิดว่า แนวคิด Iterative UI Design มีความเกี่ยวข้องอย่างไร กับแนวคิดเชิงวิศวกรรมซอฟต์แวร์อื่นๆ เช่น Agile Development, Design Thinking หรือ Prototyping ?

คำศัพท์สำคัญ

User Interface (UI) – ส่วนติดต่อผู้ใช้
องค์ประกอบที่ผู้ใช้มองเห็นและมีปฏิสัมพันธ์กับระบบ เช่น ข้อความ ปุ่ม ไอคอน รูปภาพ และการจัดวางหน้าจอ UI เป็นตัวกลางสื่อสารระหว่างผู้ใช้และแอปพลิเคชัน

UIKit – แนวคิดการพัฒนา UI แบบ Imperative
เฟรมเวิร์กสำหรับการสร้าง UI บน iOS ที่ใช้แนวคิดแบบเชิงสั่งการ (Imperative Programming) นักพัฒนาต้องกำหนดขั้นตอนการสร้าง UI ทีละลำดับ เช่น สร้างวัตถุ จัดวางตำแหน่ง และอัปเดต UI ด้วยตนเองเมื่อข้อมูลเปลี่ยนแปลง

SwiftUI – แนวคิดการพัฒนา UI แบบ Declarative
เฟรมเวิร์กสมัยใหม่สำหรับการสร้าง UI ด้วยรูปแบบเชิงประกาศ (Declarative Programming) นักพัฒนาระบุเพียงว่า UI ควรมีลักษณะเช่นใด แล้วระบบจะจัดการรายละเอียดภายในให้ UI ถูกขับเคลื่อนด้วย state และอัปเดตอัตโนมัติเมื่อข้อมูลเปลี่ยน

Imperative UI – การออกแบบ UI แบบเชิงสั่งการ
รูปแบบการพัฒนา UI ที่นักพัฒนาต้องควบคุมขั้นตอนทุกส่วนของการทำงานตั้งแต่การสร้าง อัปเดต และจัดการความสัมพันธ์ขององค์ประกอบ UI ทำให้เกิดความซับซ้อนเมื่อหน้าจอมีความหลากหลายหรือเปลี่ยนแปลงบ่อย

Declarative UI – การออกแบบ UI แบบเชิงประกาศ
แนวคิดการพัฒนา UI โดยกำหนด “ผลลัพธ์ที่ต้องการ” มากกว่าลำดับขั้นตอนการทำงาน ระบบจะจัดการการแสดงผล การอัปเดตตาม state และการวาดหน้าจอให้โดยอัตโนมัติ ส่งผลให้โค้ดกระชับและเข้าใจง่ายกว่าแบบ Imperative

State – สถานะของข้อมูลที่ส่งผลต่อ UI
ตัวแปรที่กำหนดลักษณะของ UI ณ ขณะใดขณะหนึ่ง เมื่อ state เปลี่ยน UI ก็จะเปลี่ยนตาม ตัวอย่างเช่น ค่า count ใน Counter App หรือ Boolean ที่กำหนดการแสดงผลปุ่มหรือข้อความ

Iterative UI Design – การออกแบบ UI แบบวนซ้ำ
กระบวนการออกแบบ UI ที่ประกอบด้วยการสร้างต้นแบบ (Prototype) การทดสอบ การรับข้อเสนอแนะ และการปรับปรุงหลายรอบ เพื่อให้ UI มีความชัดเจน ใช้งานง่าย และตอบสนองต่อความต้องการผู้ใช้ได้อย่างมีประสิทธิภาพ

Accessibility – การออกแบบเพื่อผู้ใช้ทุกกลุ่ม
แนวคิดในการพัฒนา UI ที่รองรับผู้ใช้ที่มีความต้องการพิเศษ เช่น ผู้มีความบกพร่องทางการมองเห็น ผ่านเครื่องมืออย่าง .accessibilityLabel , .accessibilityHint และ .accessibilityAddTraits(.isHeader) เพื่อให้แอปสามารถเข้าถึงและใช้งานได้อย่างเท่าเทียม

Layout Hierarchy – ลำดับชั้นของการจัดวาง UI
หลักการจัดเรียงองค์ประกอบ UI ตามลำดับความสำคัญ เพื่อให้ผู้ใช้เข้าใจข้อมูลก่อน–หลังได้ง่าย เช่น ข้อความหัวเรื่องอยู่ด้านบน ปุ่มการกระทำหลักอยู่ด้านล่าง

รายละเอียดสำหรับการอ้างอิงเอกสารชุดนี้
- **ผู้เขียน** ธิติ ธีระเธียร
- **วันที่เผยแพร่** วันที่ 1 ธันวาคม 2568
- **เข้าถึงได้จาก** https://ajthiti.gitbook.io/develop-in-swift/getting-started/hello-swiftui