แนวทางในการจัดกิจกรรมภาคปฏิบัติ

หัวข้อการเรียนรู้ :
View และ Modifier ใน SwiftUI

ระยะเวลาในการทำกิจกรรม: 180 - 360 นาที

วัตถุประสงค์ในการทำกิจกรรม :

1. อธิบายความหมายของ View ใน SwiftUI ในฐานะคำอธิบายเชิงโครงสร้างของส่วนติดต่อผู้ใช้ และแยกแยะความแตกต่างจากแนวคิด “หน้าจอ” หรือ “วิดเจ็ต” ในเฟรมเวิร์กเชิงสั่งการได้อย่างถูกต้อง
2. เขียนและปรับแก้ View พื้นฐานด้วย SwiftUI โดยใช้ Xcode เพื่อสังเกตความสัมพันธ์ระหว่างข้อมูลที่เปลี่ยนแปลงกับผลลัพธ์ของ UI ตามแนวคิด View as a Function of Data ได้
3. ใช้ Modifier เพื่อปรับแต่งลักษณะของ View และอธิบายผลของลำดับการใช้ Modifier ต่อผลลัพธ์ของ UI และโครงสร้าง View Hierarchy ได้อย่างมีเหตุผล
4. วิเคราะห์โครงสร้าง View Hierarchy จากโค้ด SwiftUI โดยสามารถระบุความสัมพันธ์ระหว่าง View ระดับบนสุดและ View ย่อย รวมถึงอธิบายบทบาทของ View แต่ละส่วนในโครงสร้างของหน้าจอได้

ความคิดรวบยอด:

กิจกรรมการเรียนรู้ชุดนี้มุ่งพัฒนาความรู้และความเข้าใจของผู้เรียนเกี่ยวกับ แนวคิดพื้นฐานของการออกแบบส่วนติดต่อผู้ใช้ด้วย SwiftUI ภายใต้กรอบการพัฒนาแบบเชิงประกาศ (Declarative UI) โดยเน้นให้ผู้เรียนเข้าใจว่า View ใน SwiftUI คือคำอธิบายเชิงโครงสร้างของส่วนติดต่อผู้ใช้ ไม่ใช่วัตถุหรือหน้าจอที่ถูกควบคุมโดยตรง และการเปลี่ยนแปลงของส่วนติดต่อผู้ใช้เกิดจากข้อมูลที่เปลี่ยนแปลง ไม่ใช่จากการสั่งอัปเดต UI ทีละขั้นตอน

ผู้เรียนจะได้เรียนรู้บทบาทของ Modifier ในฐานะกลไกการแปลง View เพื่อกำหนดลักษณะและพฤติกรรมของส่วนติดต่อผู้ใช้โดยไม่ทำลายโครงสร้างของ View รวมถึงการทำความเข้าใจว่า UI ใน SwiftUI เกิดจากการประกอบ View หลายตัวเป็น View Hierarchy ซึ่งทำหน้าที่เป็นแบบจำลองเชิงโครงสร้างของหน้าจอ การเรียนรู้ชุดนี้จึงช่วยพัฒนากรอบความคิดเชิงระบบในการออกแบบ UI ควบคู่ไปกับทักษะการเขียนโค้ด SwiftUI และการใช้เทคโนโลยี AI อย่างเหมาะสมในฐานะผู้ช่วยในการเรียนรู้ การออกแบบ และการแก้ปัญหา โดยไม่พึ่งพา AI ในการเขียนโค้ดแทนผู้เรียน

บทความสำหรับอ่านประกอบการทำกิจกรรม
https://ajthiti.gitbook.io/develop-in-swift/getting-started/view-and-modifier

แนวทางในการจัดกิจกรรมการเรียนรู้

ขั้นนำเข้าสู่บทเรียน (Warm-up)

การนำเข้าสู่บทเรียนในกิจกรรมการเรียนรู้เรื่อง View และ Modifier ใน SwiftUI มีเป้าหมายสำคัญเพื่อช่วยให้ผู้เรียนปรับกรอบความคิดเกี่ยวกับการออกแบบส่วนติดต่อผู้ใช้ (User Interface) จากแนวคิดแบบเดิมที่เน้นการสั่งงานและควบคุมขั้นตอนการแสดงผล ไปสู่แนวคิดการออกแบบแบบเชิงประกาศ (Declarative UI) ซึ่งเป็นพื้นฐานสำคัญของ SwiftUI

ผู้สอนควรเริ่มต้นด้วยการชวนผู้เรียนย้อนคิดถึงประสบการณ์เดิมในการใช้งานแอปพลิเคชันบนโทรศัพท์มือถือ โดยตั้งคำถามกระตุ้นความคิด เช่น **“เมื่อเราเปิดแอปหนึ่งขึ้นมา หน้าจอที่เราเห็นนั้นถูกสร้างขึ้นมาอย่างไร”** หรือ **“เมื่อข้อมูลบนหน้าจอเปลี่ยนไป เช่น คะแนนเพิ่มขึ้น หรือข้อความเปลี่ยน หน้าจอรู้ได้อย่างไรว่าต้องเปลี่ยน”** คำถามลักษณะนี้ไม่ได้มุ่งหวังคำตอบที่ถูกต้องในเชิงเทคนิค แต่มีเป้าหมายเพื่อเปิดพื้นที่ให้ผู้เรียนได้แสดงความคิดเห็น และตระหนักว่า การเปลี่ยนแปลงของหน้าจอแอปไม่ได้เกิดขึ้นแบบสุ่ม แต่เกิดจากแนวคิดและโครงสร้างที่นักพัฒนาออกแบบไว้ จากนั้น ผู้สอนอาจอธิบายเชื่อมโยงให้เห็นว่า ในการพัฒนาแอปแบบดั้งเดิม นักพัฒนามักต้องเขียนโค้ดเพื่อสั่งให้หน้าจอเปลี่ยนแปลงโดยตรง เช่น สั่งให้เปลี่ยนข้อความ เปลี่ยนสี หรือปรับตำแหน่งขององค์ประกอบบนหน้าจอ แต่แนวทางดังกล่าวอาจมีความซับซ้อนเมื่อแอปมีขนาดใหญ่หรือข้อมูลเปลี่ยนแปลงบ่อย ดังนั้น **SwiftUI จึงนำเสนอแนวคิดใหม่ที่เปลี่ยนบทบาทของนักพัฒนาจากผู้ควบคุมหน้าจอ ไปสู่ผู้ออกแบบ “คำอธิบายของหน้าจอ”**

ในช่วงท้ายของการนำเข้าสู่บทเรียน ผู้สอนอาจตั้งคำถามนำเพื่อเชื่อมโยงไปสู่กิจกรรมภาคปฏิบัติ เช่น **“ถ้าเราไม่สั่งให้หน้าจออัปเดต แล้วหน้าจอจะเปลี่ยนตามข้อมูลได้อย่างไร”** หรือ **“ถ้าเรามองหน้าจอเป็นคำอธิบาย หน้าจอหนึ่งหน้าจะถูกอธิบายอย่างไร”** คำถามเหล่านี้จะทำหน้าที่เป็นสะพานเชื่อมจากการเรียนรู้เชิงแนวคิดไปสู่การทดลองเขียนโค้ดจริงใน Xcode ซึ่งเป็นขั้นตอนถัดไปของบทเรียน โดยช่วยให้ผู้เรียนมีกรอบความคิดที่พร้อมสำหรับการทำความเข้าใจ View และ Modifier ใน SwiftUI อย่างเป็นระบบ

ขั้นการเรียนรู้ (Learn)

ในขั้นการเรียนรู้ในบทเรียนนี้ออกแบบให้ผู้เรียนค่อย ๆ สร้างความเข้าใจเกี่ยวกับ **View และ Modifier ใน SwiftUI** ผ่านการสลับบทบาทระหว่าง การเรียนรู้จากผู้สอน การลงมือทดลองเขียนโค้ดด้วย Xcode และการใช้ AI เป็นผู้ช่วยสะท้อนความคิด โดยกิจกรรมทั้งหมดเรียงลำดับจากแนวคิดพื้นฐานไปสู่การประยุกต์ใช้เชิงโครงสร้าง

กิจกรรมที่ 1 : ทำความเข้าใจ View ในฐานะคำอธิบายของ UI

กิจกรรมนี้มีเป้าหมายเพื่อช่วยให้ผู้เรียนสร้างกรอบความคิดที่ถูกต้องเกี่ยวกับ View ใน SwiftUI ตั้งแต่เริ่มต้น โดยทำให้ผู้เรียนเข้าใจว่า View ไม่ใช่หน้าจอหรือวัตถุที่ถูกควบคุมโดยตรง แต่เป็น คำอธิบายเชิงโครงสร้าง ของส่วนติดต่อผู้ใช้ การวางรากฐานแนวคิดในกิจกรรมนี้มีความสำคัญอย่างยิ่ง เพราะจะส่งผลต่อความเข้าใจในกิจกรรมถัดไปทั้งหมด

ผู้สอนเริ่มกิจกรรมด้วยการเปิด SwiftUI Project ใหม่ใน Xcode และแสดงโค้ดเริ่มต้นของ ContentView บนหน้าจอ โดยยังไม่รีบอธิบายไวยากรณ์หรือโครงสร้างของภาษา Swift แต่ควรชวนผู้เรียนมองโค้ดในเชิงความหมายก่อน

```
struct ContentView: View {
    var body: some View {
        Text("Hello, SwiftUI")
    }
}
```

ผู้สอนอธิบายให้ผู้เรียนเข้าใจว่า โค้ดชุดนี้ **ไม่ได้สร้างหน้าจอจริง ๆ ขึ้นมาในทันที** และไม่ได้สั่งให้ระบบวาดข้อความบนหน้าจอ แต่เป็นการอธิบายว่า หากต้องแสดง ContentView หน้าจอควรมี Text ที่แสดงข้อความว่า “Hello, SwiftUI” เท่านั้น ผู้สอนอาจใช้ถ้อยคำเปรียบเทียบ เช่น “เป็นเหมือนพิมพ์เขียวของหน้าจอ” เพื่อช่วยให้ผู้เรียนเห็นภาพ จากนั้น ผู้สอนควรสาธิตการเปลี่ยนข้อความภายใน Text เป็นค่าอื่น เช่น “SwiftUI View” พร้อมให้ผู้เรียนสังเกต Preview ที่เปลี่ยนแปลงทันที และตั้งคำถามนำ เช่น **“ในโค้ดนี้ เราได้เรียกคำสั่งให้หน้าจออัปเดตตรงไหนหรือไม่”** เพื่อชี้ให้เห็นว่าการเปลี่ยนแปลงของ UI เกิดจากการเปลี่ยนคำอธิบายของ View ไม่ใช่จากการสั่งงาน

โดยตรง ในขั้นนี้ ผู้สอนควรหลีกเลี่ยงการใช้ศัพท์เทคนิคมากเกินไป และมุ่งเน้นให้ผู้เรียนเข้าใจ แนวคิด มากกว่ารายละเอียดของภาษา

หลังจากการสาธิต ผู้สอนควรให้ผู้เรียนได้ทดลองเขียนคำสั่งโดยเปิด Xcode และสร้าง SwiftUI Project ด้วยตนเอง จากนั้นให้ผู้เรียนทดลองปรับแก้ข้อความภายใน Text ใน ContentView ตามตัวอย่างที่ผู้สอนสาธิต ผู้สอนควรเดินสำรวจการทำงานของผู้เรียน และตั้งคำถามเชิงชี้นำ เช่น “ถ้าเปลี่ยนข้อความนี้ หน้าจอจะเปลี่ยนหรือไม่ เพราะเหตุใด” จุดสำคัญที่ผู้สอนควรเน้นคือ ผู้เรียนไม่จำเป็นต้องเขียนโค้ดเพิ่มเติมในขั้นนี้ แต่ควรใช้เวลาในการสังเกตความสัมพันธ์ระหว่างโค้ดกับผลลัพธ์ของ UI เพื่อสร้างความเข้าใจว่า View ทำหน้าที่เป็นคำอธิบายของหน้าจอ ไม่ใช่ตัวควบคุมหน้าจอ ผู้สอนควรย้ำให้ผู้เรียนสังเกตว่า Preview เปลี่ยนทันทีเมื่อแก้ไขโค้ด และชี้ให้เห็นว่านี่คือคุณลักษณะสำคัญของ SwiftUI ภายใต้แนวคิด Declarative UI

เมื่อผู้เรียนได้ทดลองด้วยตนเองแล้ว ผู้สอนควรแนะนำให้ผู้เรียนใช้ LM Studio ในบทบาทของ AI Tutor เพื่อช่วยสะท้อนความเข้าใจและเรียบเรียงความคิดของตนเอง โดยอาจให้ผู้เรียนใช้ Prompt ต่อไปนี้ใน LM Studio

Prompt: ในการออกแบบส่วนติดต่อกับผู้ใช้ด้วย SwiftUI นั้น เหตุใดการเปลี่ยนข้อความใน Text จึงทำให้หน้าจอเปลี่ยน โดยไม่ต้องเรียกคำสั่งอัปเดต UI

หลังจากผู้เรียนได้รับคำตอบจาก AI ผู้สอนควรชวนผู้เรียนพิจารณาคำตอบอย่างมีวิจารณญาณ โดยอาจถามต่อว่า “คำอธิบายนี้ตรงกับสิ่งที่เราเห็นจากการทดลองหรือไม่” หรือ “มีส่วนใดที่ยังไม่เข้าใจ หรืออยากอธิบายเพิ่มเติมในแบบของตนเอง”

กิจกรรมที่ 2 : View as a Function of Data

กิจกรรมนี้มีเป้าหมายเพื่อช่วยให้ผู้เรียนเข้าใจว่า **View ใน SwiftUI ไม่ได้เป็นเพียงคำอธิบายแบบตายตัว** แต่เป็นคำอธิบายของ UI ที่ ขึ้นอยู่กับข้อมูล ที่ View ได้รับ แนวคิดนี้เป็นหัวใจของ Declarative UI และเป็นรากฐานสำคัญก่อนการเรียนรู้เรื่อง Properties และ State ในบทถัดไป ผู้เรียนควรเริ่มมอง View ว่า “รับข้อมูล → สร้างหน้าจอ” มากกว่าการมองว่า View เป็นหน้าจอที่มีค่าคงที่

ผู้สอนควรเริ่มต้นด้วยการทบทวนสั้น ๆ จากกิจกรรมที่ 1 โดยย้ำว่า View เป็นคำอธิบายของ UI จากนั้นชวนผู้เรียนตั้งคำถามต่อว่า “ถ้าคำอธิบายนี้อ้างอิงข้อมูล คำอธิบายจะเปลี่ยนตามข้อมูลได้หรือไม่” จากนั้น ผู้สอนสาธิตการเพิ่มตัวแปรข้อมูลเข้าไปใน ContentView โดยเขียนโค้ดดังนี้

```
struct ContentView: View {
    let message = "Hello, SwiftUI"

    var body: some View {
        Text(message)
    }
}
```

ผู้สอนควรอธิบายให้ผู้เรียนเข้าใจว่า **ตัวแปร message** ในที่นี้ไม่ได้เป็นคำสั่งควบคุมหน้าจอ แต่เป็น **ข้อมูล** ที่ถูกนำมาใช้สร้างคำอธิบายของ UI ผู้สอนอาจใช้ถ้อยคำเปรียบเทียบ เช่น “View กำลังอ่านข้อมูล แล้วอธิบายหน้าจอตามข้อมูลนั้น” เมื่อเปลี่ยนค่าของ message ผู้สอนควรชี้ให้ผู้เรียนสังเกตว่า **UI เปลี่ยนตามข้อมูลทันที โดยที่โค้ดใน body ไม่ได้เปลี่ยนแปลง** ซึ่งเป็นจุดสำคัญที่สะท้อนแนวคิด View as a Function of Data

หลังจากการสาธิต ผู้สอนให้ผู้เรียนเปิด Xcode และปรับโค้ดใน ContentView ตามตัวอย่าง จากนั้นให้ผู้เรียนทดลองเปลี่ยนค่าของ message เป็นข้อความอื่น ๆ และสังเกตผลลัพธ์ใน Preview ในระหว่างกาทำกิจกรรม ผู้สอนควรเดินสำรวจและตั้งคำถามเชิงชี้นำ เช่น

- “ถ้าโค้ด View เหมือนเดิม แต่ข้อมูลเปลี่ยน หน้าจอควรเปลี่ยนหรือไม่”
- “ถ้าเปลี่ยนข้อมูลกลับเป็นค่าเดิม หน้าจอจะกลับมาเหมือนเดิมหรือไม่ เพราะเหตุใด”

เมื่อผู้เรียนเริ่มเข้าใจแล้ว ผู้สอนอาจชวนให้ขยายการทดลองโดยสร้าง View ใหม่ที่รับข้อมูลจากภายนอก เช่น MessageView

```
struct MessageView: View {
    let message: String

    var body: some View {
        Text(message)
    }
}
```

และนำไปใช้ใน ContentView

```
struct ContentView: View {
    var body: some View {
        VStack(spacing: 16) {
            MessageView(message: "Hello")
            MessageView(message: "SwiftUI")
            MessageView(message: "View as Function")
        }
    }
}
```

ในขั้นนี้ ผู้สอนควรชี้ให้เห็นว่า MessageView ถูกเขียนเพียงครั้งเดียว แต่สามารถสร้าง UI ที่แตกต่างกันได้ตามข้อมูลที่รับเข้ามา ซึ่งเป็นตัวอย่างที่ชัดเจนของการมอง View เป็นฟังก์ชันของข้อมูล

เมื่อผู้เรียนได้ทดลองเขียนโค้ดและสังเกตผลลัพธ์แล้ว ผู้สอนควรให้ผู้เรียนใช้ LM Studio ในบทบาทของ AI Tutor เพื่อช่วยเรียบเรียงและสะท้อนความเข้าใจของตนเอง โดยผู้เรียนอาจใช้ Prompt ต่อไปนี้

Prompt: จากตัวอย่าง MessageView ช่วยอธิบายให้ฉันเข้าใจว่า View ใน SwiftUI ทำหน้าที่เหมือนฟังก์ชันของข้อมูลอย่างไร โดยไม่กล่าวถึงไวยากรณ์ของโค้ด

หลังจากผู้เรียนได้รับคำตอบจาก AI ผู้สอนควรชวนผู้เรียนพิจารณาคำอธิบายดังกล่าว และตั้งคำถามต่อ เช่น “คำอธิบายนี้สอดคล้องกับสิ่งที่เราเห็นจาก Xcode หรือไม่” หรือ “ถ้าเปลี่ยนข้อมูล หน้าจอเปลี่ยนเพราะอะไร ไม่เปลี่ยนเพราะอะไร” ขั้นตอนนี้ช่วยให้ผู้เรียนใช้ AI เป็นเครื่องมือในการจัดระเบียบความคิด และเชื่อมโยงการลงมือทำกับแนวคิดเชิงนามธรรมได้อย่างมีความหมาย

กิจกรรมที่ 3 : การทำความเข้าใจ View Hierarchy ใน SwiftUI

กิจกรรมนี้มีเป้าหมายเพื่อช่วยให้ผู้เรียนเข้าใจว่า **UI ใน SwiftUI ไม่ได้เป็นผลรวมของ View ทีละตัว** แต่เป็นผลลัพธ์ของ **โครงสร้างลำดับชั้นของ View (View Hierarchy)** ซึ่ง SwiftUI ใช้เป็นแบบจำลองเชิงโครงสร้างในการประเมินและแสดงผล UI ผู้เรียนควรเริ่มมองโค้ด SwiftUI ในเชิงโครงสร้างแบบต้นไม้ (tree) และสามารถอธิบายความสัมพันธ์ระหว่าง View ระดับบนสุดและ View ย่อยได้อย่างชัดเจน

ผู้สอนควรเริ่มกิจกรรมด้วยการทบทวนแนวคิดจากกิจกรรมที่ 2 แบบสั้น ๆ โดยย้ำว่า View เป็นฟังก์ชันของข้อมูล จากนั้นชวนผู้เรียนตั้งคำถามต่อว่า “ถ้าเรามี View หลายตัว หน้าจอหนึ่งหน้าจะถูกประกอบขึ้นมาอย่างไร”

ผู้สอนสาธิตด้วยโค้ดตัวอย่างที่มีโครงสร้างชัดเจน เช่น

```
struct ContentView: View {
    var body: some View {
        VStack {
            Text("Profile")
            HStack {
                Image(systemName: "person.circle")
                Text("Student")
            }
        }
    }
}
```

ในขั้นนี้ ผู้สอนไม่ควรอธิบายเรื่อง layout หรือการจัดตำแหน่งเชิงลึก แต่ควรชี้นำให้ผู้เรียน “อ่านโครงสร้าง” ของโค้ด โดยตั้งคำถามนำ เช่น

- View ใดเป็น View ระดับบนสุด
- View ใดทำหน้าที่จัดโครงสร้าง
- View ใดเป็น View ที่แสดงข้อมูล

ผู้สอนควรอธิบายว่า VStack และ HStack ไม่ได้เป็นเพียงเครื่องมือจัดตำแหน่ง แต่เป็น View ที่ทำหน้าที่กำหนดความสัมพันธ์เชิงโครงสร้างระหว่าง View ย่อยทั้งหมด

หลังจากการสาธิต ผู้สอนให้ผู้เรียนเปิด Xcode และเขียนโค้ดตามตัวอย่าง จากนั้นให้ผู้เรียนลองวิเคราะห์ View Hierarchy ด้วยตนเอง โดยอาจให้วาดผังโครงสร้างอย่างง่าย เช่น เขียนเป็นรายการลำดับชั้นหรือผังต้นไม้บนกระดาษหรือในสมุด เมื่อผู้เรียนเริ่มเข้าใจแล้ว ผู้สอนชวนให้ทดลองแยก View บางส่วนออกมาเป็น View ใหม่ เพื่อเน้นแนวคิดการออกแบบแบบองค์ประกอบ เช่น

```
struct ProfileInfoView: View {
    var body: some View {
        HStack {
            Image(systemName: "person.circle")
            Text("Student")
        }
    }
}
```

และนำไปใช้ใน ContentView ดังนี้

```
struct ContentView: View {
    var body: some View {
        VStack {
            Text("Profile")
            ProfileInfoView()
        }
    }
}
```

ผู้สอนควรชี้ให้ผู้เรียนสังเกตว่า แม้จะมีการแยก View ออกเป็นส่วนย่อย แต่ผลลัพธ์ของ UI ยังคงเหมือนเดิม สิ่งที่เปลี่ยนไปคือ **โครงสร้างของโค้ด** ซึ่งอ่านง่ายขึ้นและจัดการได้ง่ายขึ้น นี่คือประโยชน์สำคัญของการเข้าใจ View Hierarchy

เมื่อผู้เรียนได้ทดลองและเห็นโครงสร้างของ View แล้ว ผู้สอนให้ผู้เรียนใช้ **LM Studio** เพื่อช่วยสะท้อนและจัดระเบียบความเข้าใจ โดยอาจให้ใช้ AI ในบทบาท AI Tutor หรือ Design Advisor เพื่ออธิบายโครงสร้างที่ตนเองสร้างขึ้น

Prompt: จากโค้ด SwiftUI ที่ฉันเขียน อธิบาย View Hierarchy ในรูปแบบข้อความ โดยระบุว่า View ใดเป็น parent และ child และการเขียนโค้ดในลักษณะนี้มีข้อดีอย่างไร

หลังจากผู้เรียนได้รับคำอธิบายจาก AI ผู้สอนควรชวนผู้เรียนพิจารณาคำตอบ และเชื่อมโยงกลับไปยังโค้ดจริงใน Xcode เช่น “คำอธิบายนี้ตรงกับโครงสร้างที่เราเห็นในโค้ดหรือไม่” และ “ถ้าเพิ่ม View ใหม่เข้าไป View Hierarchy จะเปลี่ยนอย่างไร”

กิจกรรมที่ 4 : Modifier และลำดับของ Modifier ใน SwiftUI

กิจกรรมนี้มีเป้าหมายเพื่อช่วยให้ผู้เรียนเข้าใจว่า Modifier ใน SwiftUI ไม่ได้ทำหน้าที่แก้ไข View เดิม แต่เป็นกลไกที่ใช้ สร้าง View ใหม่ จาก View เดิมโดยเพิ่มคุณลักษณะเข้าไป และลำดับของ Modifier ที่ถูกเรียกใช้มีผลโดยตรงต่อโครงสร้างของ View Hierarchy และผลลัพธ์ของ UI ความเข้าใจในประเด็นนี้จะช่วยให้ผู้เรียนสามารถวิเคราะห์และแก้ปัญหา UI ได้อย่างมีเหตุผล แทนการลองผิดลองถูก

ผู้สอนควรเริ่มกิจกรรมด้วยการทบทวนแนวคิดจากกิจกรรมที่ 3 โดยย้ำว่า UI ใน SwiftUI เกิดจาก View Hierarchy จากนั้นชวนผู้เรียนตั้งคำถามต่อว่า “**ถ้าเราอยากเปลี่ยนลักษณะของ View เช่น ขนาด สี หรือระยะห่าง เรากำลัง ‘เปลี่ยน View เดิม’ หรือ ‘สร้าง View ใหม่’**” จากนั้น ผู้สอนสาธิตการใช้ Modifier ง่าย ๆ เช่น

```
Text("Hello, SwiftUI")
    .font(.title)
```

ผู้สอนควรอธิบายให้ผู้เรียนเข้าใจว่า โค้ดนี้ **ไม่ได้เปลี่ยนฟอนต์ของ Text เดิม** แต่หมายถึง **“มี View ใหม่ที่เกิดจาก Text เดิม โดยมีคุณลักษณะด้านฟอนต์เพิ่มเข้ามา”** ผู้สอนอาจอธิบายเชิงเปรียบเทียบว่า Modifier ทำหน้าที่ “ห่อ” View เดิมไว้ ในขั้นนี้ ผู้สอนควรเน้นแนวคิด **immutability ของ View** โดยอธิบายว่า View ไม่ถูกแก้ไขค่าภายใน แต่ SwiftUI จะสร้างคำอธิบายใหม่ของ UI ทุกครั้งที่มีการแปลง View

หลังจากการสาธิต ผู้สอนให้ผู้เรียนเปิด Xcode และทดลองใช้ Modifier หลายตัวกับ View เดียวกัน เช่น

```
Text("SwiftUI")
    .font(.largeTitle)
    .foregroundStyle(.blue)
    .padding()
```

ผู้สอนควรชวนผู้เรียนสังเกตว่า Modifier ถูกเรียงลำดับจากบนลงล่าง และแต่ละ Modifier รับ View จากขั้นก่อนหน้าเสมอ จากนั้นให้ผู้เรียนทดลอง **สลับลำดับของ Modifier** เพื่อเปรียบเทียบผลลัพธ์ เช่น

```
Text("Hello")
    .padding()
    .background(.yellow)
```

และ

```
Text("Hello")
    .background(.yellow)
    .padding()
```

ผู้สอนควรชี้ให้ผู้เรียนสังเกตว่า แม้จะใช้ Modifier ชุดเดียวกัน แต่ UI ที่ได้แตกต่างกันอย่างชัดเจน และตั้งคำถามนำ เช่น

- “View ใดถูกห่อก่อน”
- “พื้นหลังถูกเพิ่มให้กับ View ใด”

ในขั้นนี้ ผู้สอนควรเชื่อมโยงกลับไปยังแนวคิด View Hierarchy เพื่อให้ผู้เรียนเห็นว่า ลำดับของ Modifier ทำให้โครงสร้างลำดับชั้นของ View เปลี่ยนไป

เมื่อผู้เรียนได้ทดลองและสังเกตผลลัพธ์ด้วยตนเองแล้ว ผู้สอนให้ผู้เรียนใช้ **LM Studio** เพื่อช่วยสะท้อนความเข้าใจและอธิบายเหตุผลเชิงโครงสร้าง โดยสามารถกำหนดบทบาทของ AI เป็น AI Tutor หรือ Debugging Assistant ตัวอย่าง Prompt ที่แนะนำ ได้แก่

Prompt: อธิบายว่าทำไมการสลับลำดับของ Modifier จึงทำให้ผลลัพธ์ของ UI แตกต่างกัน อธิบายเชิงแนวคิด ไม่กล่าวถึงไวยากรณ์ของโค้ด

ในกรณีที่ผู้เรียนยังสับสนอาจให้ถามต่อดังนี้
Prompt: จากโค้ด SwiftUI นี้ อธิบายว่า Modifier แต่ละตัว กำลังห่อ View ใดใน View Hierarchy

หลังจากผู้เรียนได้รับคำอธิบายจาก AI ผู้สอนควรชวนผู้เรียนตรวจสอบคำตอบนั้นกับโค้ดจริงใน Xcode เช่น “คำอธิบายนี้สอดคล้องกับสิ่งที่เราเห็นใน Preview หรือไม่” “ถ้าเพิ่ม Modifier ใหม่เข้าไป View Hierarchy จะเปลี่ยนอย่างไร” ขั้นตอนนี้ช่วยให้ผู้เรียนใช้ AI เป็นเครื่องมือช่วยวิเคราะห์และอธิบาย ไม่ใช่เป็นผู้ให้คำตอบสำเร็จรูป

กิจกรรมที่ 5 : การออกแบบ View อย่างเป็นระบบและการสังเคราะห์แนวคิด

กิจกรรมนี้มีเป้าหมายเพื่อช่วยให้ผู้เรียน สังเคราะห์แนวคิดทั้งหมดที่ได้เรียนรู้จากกิจกรรมที่ 1–4 โดยส่งเสริมให้ผู้เรียนสามารถนำแนวคิดเหล่านี้มาใช้ร่วมกันในการออกแบบหน้าจออย่างง่าย โดยเน้นโครงสร้าง ความชัดเจน และเหตุผลในการออกแบบ มากกว่าความสวยงามของ UI กิจกรรมนี้ทำหน้าที่เป็นจุดเปลี่ยนจาก “การทดลองตามตัวอย่าง” ไปสู่ “การออกแบบอย่างมีกรอบคิด”

ผู้สอนควรเริ่มกิจกรรมด้วยการทบทวนแนวคิดสำคัญทั้งหมดร่วมกับผู้เรียน โดยตั้งคำถามสรุป เช่น

- “ถ้าต้องอธิบายหน้าจอหนึ่งหน้าด้วย SwiftUI เราควรเริ่มคิดจากอะไร”
- “View, Modifier และข้อมูล ทำงานร่วมกันอย่างไร”

จากนั้น ผู้สอนสาธิตการออกแบบหน้าจออย่างง่าย โดยเน้นการคิดเชิงโครงสร้างก่อนการเขียนโค้ด ผู้สอนอาจอธิบายแนวคิดในลักษณะลำดับความคิด เช่น

- หน้าจอประกอบด้วย View อะไรบ้าง
- View ใดควรแยกออกมาเป็นองค์ประกอบย่อย
- ส่วนใดเป็นโครงสร้าง และส่วนใดเป็นการตกแต่งด้วย Modifier

ผู้สอนอาจสาธิตด้วยโค้ดตัวอย่าง เช่น หน้าจอข้อมูลผู้ใช้แบบง่าย

```
struct ContentView: View {
    var body: some View {
        VStack(spacing: 16) {
            HeaderView(title: "Profile")
            InfoView(label: "Name", value: "Student")
            InfoView(label: "Role", value: "Developer")
        }
        .padding()
    }
}
```

พร้อมอธิบายว่าโค้ดนี้สะท้อนแนวคิดการแยก View ออกเป็นองค์ประกอบย่อย และใช้ Modifier เพื่อปรับลักษณะโดยไม่ทำลายโครงสร้าง

หลังจากการสาธิต ผู้สอนให้ผู้เรียนออกแบบหน้าจออย่างง่ายด้วยตนเอง โดยกำหนดกรอบชัดเจน เช่น “ออกแบบหน้าจอที่มีอย่างน้อย 2–3 View ย่อย ใช้ข้อมูลเป็นตัวกำหนด UI และมีการใช้ Modifier อย่างเหมาะสม” ผู้สอนไม่ควรกำหนดหน้าตาของ UI ตายตัว แต่ควรเน้นให้ผู้เรียน

- แยก View อย่างมีเหตุผล
- ตั้งชื่อ View ให้สื่อความหมาย
- ใช้ Modifier เพื่อปรับลักษณะ ไม่ใช่แก้โครงสร้าง

ในระหว่างที่ผู้เรียนทำงาน ผู้สอนควรเดินดูและตั้งคำถามเชิงชี้นำ เช่น “View นี้ควรแยกออกมาเป็น component หรือไม่ เพราะเหตุใด” หรือ “Modifier ตัวนี้ควรอยู่ตำแหน่งใดใน View Hierarchy” กิจกรรมนี้ไม่ควรเร่งเวลา ผู้เรียนควรมีโอกาสคิดและปรับแก้โครงสร้างของตนเอง กิจกรรมที่ 5 นี้เป็นกิจกรรมสังเคราะห์ที่ช่วยให้ผู้เรียนเชื่อมโยงความรู้ทั้งหมดเข้าด้วยกัน และตระหนักว่า SwiftUI ไม่ใช่เพียงการเขียนโค้ด แต่เป็นการออกแบบโครงสร้างของ UI อย่างมีระบบ กรอบความคิดที่ได้จากกิจกรรมนี้จะช่วยให้ผู้เรียนพร้อมสำหรับการเรียนรู้หัวข้อถัดไปเรื่อง Properties และ State ซึ่งเป็นการนำข้อมูลที่เปลี่ยนแปลงได้มาใช้กับ View อย่างแท้จริง

ขั้นสรุปบทเรียน (Conclusion)

ภายในบทเรียนนี้ ผู้เรียนได้เรียนรู้แนวคิดพื้นฐานที่สำคัญของการออกแบบส่วนติดต่อผู้ใช้ด้วย SwiftUI ภายใต้กรอบการพัฒนาแบบเชิงประกาศ (Declarative UI) โดยเปลี่ยนมุมมองจากการมองหน้าจอเป็นวัตถุที่ต้องถูกควบคุม ไปสู่การมองหน้าจอเป็น **คำอธิบายเชิงโครงสร้างของ UI** ที่ถูกกำหนดจากข้อมูลและโครงสร้างของ View

ผู้เรียนได้ทำความเข้าใจว่า View ใน SwiftUI ไม่ใช่หน้าจอหรือวิดเจ็ตที่มีตัวตนถาวร แต่เป็นโครงสร้างข้อมูลที่อธิบายว่าหน้าจอควรมีลักษณะอย่างไรในช่วงเวลาหนึ่ง การเปลี่ยนแปลงของ UI จึงเกิดจากการเปลี่ยนแปลงของข้อมูล ไม่ใช่จากการสั่งอัปเดตหน้าจอโดยตรง แนวคิดนี้ถูกขยายต่อไปสู่การมอง View ในฐานะฟังก์ชันของข้อมูล ซึ่งช่วยให้ผู้เรียนเข้าใจว่าทำไม View เดียวกันจึงสามารถให้ผลลัพธ์ของ UI ที่แตกต่างกันได้เมื่อข้อมูลที่รับเข้ามาเปลี่ยนไป นอกจากนี้ ผู้เรียนยังได้เรียนรู้ว่า UI ใน SwiftUI เกิดจากการประกอบ View หลายตัวเข้าด้วยกันเป็น View Hierarchy ซึ่งทำหน้าที่เป็นแบบจำลองเชิงโครงสร้างของหน้าจอทั้งหมด และได้ทำความเข้าใจบทบาทของ Modifier ในฐานะกลไกการแปลง View ที่ช่วยเพิ่มคุณลักษณะให้กับ UI โดยไม่ทำลายโครงสร้างของ View เดิม ลำดับของ Modifier ที่ถูกเรียกใช้มีผลโดยตรงต่อโครงสร้างของ View Hierarchy และผลลัพธ์ของ UI

โดยสรุป บทเรียนนี้ไม่ได้มุ่งเน้นเพียงการเขียนโค้ด SwiftUI ให้ได้ผลลัพธ์บนหน้าจอ แต่เน้นให้ผู้เรียนพัฒนา กรอบความคิดเชิงโครงสร้างในการออกแบบ UI ซึ่งเป็นพื้นฐานสำคัญสำหรับการเรียนรู้หัวข้อถัดไปเกี่ยวกับการปรับแต่ง View ด้วย Properties และการจัดการข้อมูลและสถานะของแอปพลิเคชันใน SwiftUI

หลังจบบทเรียนนี้ ผู้สอนอาจใช้สไลด์เรื่อง View and Modifier ซึ่งสามารถดาวน์โหลดได้จากส่วนท้ายของบทความเพื่อให้ผู้เรียนได้ลงมือปฏิบัติในการใช้ Text, Image, Shape, VStack, HStack และ ZStack ในการสร้างส่วนติดต่อกับผู้ใช้ และต่อยอดการเรียนรู้ด้วยการทดลองสร้างโปรเจกต์ตามบทความเรื่อง Customize views with properties ซึ่งเข้าถึงได้จาก https://developer.apple.com/tutorials/develop-in-swift/customize-views-with-properties

คำถามเพื่อการสะท้อนผลการเรียนรู้จากการทำกิจกรรม

1. อธิบายคำว่า **View** ใน SwiftUI แตกต่างจากความหมายของ “หน้าจอ” หรือ “วิดเจ็ต” ในเฟรมเวิร์กแบบเดิมอย่างไร และเหตุใดจึงกล่าวได้ว่า View ใน SwiftUI เป็น คำอธิบายเชิงโครงสร้าง ของ UI ไม่ใช่วัตถุที่ถูกควบคุมโดยตรง ?
2. การออกแบบ View ให้รับข้อมูลจากภายนอกช่วยให้โค้ดมีความยืดหยุ่นและนำกลับมาใช้ซ้ำได้อย่างไร ?
3. การแยกบทบาทระหว่าง View (โครงสร้าง) และ Modifier (คุณลักษณะ) ช่วยให้การออกแบบ UI ง่ายขึ้นอย่างไร ?
4. อธิบายความหมายของ **View Hierarchy** ใน SwiftUI ได้อย่างไร ?
5. หากต้องออกแบบหน้าจอที่มีความซับซ้อนมากขึ้น ผู้เรียนคิดว่าควรเริ่มต้นจากการออกแบบโครงสร้าง View อย่างไร ?

คำศัพท์สำคัญ

View (ใน SwiftUI)
View ใน SwiftUI คือ คำอธิบายเชิงโครงสร้างของส่วนติดต่อผู้ใช้ ไม่ใช่หน้าจอหรือวิดเจ็ตที่มีตัวตนถาวรบนหน้าจอ View ทำหน้าที่บอกระบบว่า “ถ้าต้องแสดง UI หน้าจอนี้ ควรมีองค์ประกอบอะไรบ้าง”

View มีลักษณะเป็นค่า (value) ที่สอดคล้องกับ View protocol และไม่ทำหน้าที่ควบคุมการแสดงผลโดยตรง จุดเด่นคือช่วยให้ UI ถูกออกแบบในเชิงโครงสร้าง และสามารถประเมินใหม่ได้ทุกครั้งเมื่อข้อมูลเปลี่ยน

View Hierarchy
View Hierarchy คือโครงสร้างลำดับชั้นของ View ทั้งหมดที่ประกอบกันเป็นหน้าจอหนึ่งหน้าจอ
ในเชิงแนวคิด View Hierarchy สามารถมองเป็นโครงสร้างข้อมูลแบบต้นไม้ (tree structure) โดยมี View ระดับบนสุดเป็นโหนดราก และมี View ย่อยเป็นโหนดลูก

จุดเด่นของ View Hierarchy คือช่วยให้ SwiftUI เข้าใจความสัมพันธ์เชิงโครงสร้างของ UI และสามารถอัปเดตเฉพาะส่วนที่จำเป็นได้อย่างมีประสิทธิภาพ ผู้เรียนที่เข้าใจ View Hierarchy จะสามารถวิเคราะห์และแก้ปัญหา UI ได้อย่างเป็นระบบ

Modifier
Modifier คือกลไกใน SwiftUI ที่ใช้ แปลง View หนึ่งไปสู่อีก View หนึ่ง โดยเพิ่มคุณลักษณะด้านการแสดงผลหรือพฤติกรรมเข้าไป Modifier ไม่ได้แก้ไข View เดิม แต่สร้าง View ใหม่ที่ห่อ (wrap) View เดิมไว้

จุดเด่นของ Modifier คือช่วยแยกโครงสร้างของ UI ออกจากลักษณะการแสดงผล และสามารถนำมาใช้ต่อเนื่องกันได้ในลักษณะของการประกอบฟังก์ชัน (function composition)

รายละเอียดสำหรับการอ้างอิงเอกสารชุดนี้

- **ผู้เขียน** ธิติ ธีระเธียร
- **วันที่เผยแพร่** วันที่ 15 มกราคม 2569
- **เข้าถึงได้จาก** https://ajthiti.gitbook.io/develop-in-swift/getting-started/view-and-modifier