

FocusFlow 團隊整合指南

整合架構概覽

核心概念

我們的整合架構基於 **模組協調器 (Module Coordinator)** 模式，每個功能（跑步、專注、遊戲）都是獨立模組，透過統一的協調器管理。



整合步驟

Step 1: 實作模組協定

每個同學的模組都需要實作 `FlowModule` 協定：

```
swift
```

```

struct RunningModule: FlowModule {
    @ObservedObject var coordinator = ModuleCoordinator.shared

    var configuration = RunningConfiguration()

    var body: some View {
        // 你的 UI 程式碼
    }

    func onStart() {
        // 模組啟動時執行
        coordinator.sharedTimer.start()
        coordinator.sharedAudio.playBackgroundMusic("running_music")
    }

    func onStop() {
        // 模組停止時執行
        let duration = coordinator.sharedTimer.currentTime
        coordinator.sendEvent(.runCompleted(minutes: Int(duration / 60)))
    }

    func onPause() {
        coordinator.sharedTimer.pause()
    }

    func onResume() {
        coordinator.sharedTimer.resume()
    }

    func getProgress() -> ModuleProgress {
        // 回傳目前進度
        return ModuleProgress(
            completed: todayRuns,
            total: goalRuns,
            percentage: completionRate,
            todayMinutes: todayMinutes,
            weekMinutes: weekMinutes,
            streakDays: streakDays
        )
    }
}

```

Step 2: 使用共用服務

 共用計時器 (SharedTimerService)

swift

```
// 開始計時
coordinator.sharedTimer.start()

// 開始倒數計時
coordinator.sharedTimer.startCountdown(from: 25 * 60) {
    // 倒數完成後的動作
    completePomodoro()
}

// 取得格式化時間
Text(coordinator.sharedTimer.formattedTime)
```

共用音效 (SharedAudioService)

swift

```
// 播放背景音樂
coordinator.sharedAudio.playBackgroundMusic("focus_music")

// 播放音效
coordinator.sharedAudio.playSound("success")

// 開始節拍器
coordinator.sharedAudio.startMetronome()
```

共用震動 (SharedHapticService)

swift

```
// 輕度震動
coordinator.sharedHaptic.impact(.light)

// 成功通知震動
coordinator.sharedHaptic.notification(.success)
```

共用資料 (SharedDataService)

swift

```
// 儲存跑步記錄
coordinator.sharedData.saveRunningRecord(
    duration: 1200, // 20分鐘
    distance: 3.5, // 3.5公里
    bpm: 120
)

// 儲存番茄記錄
coordinator.sharedData.savePomodoroRecord(
    focusDuration: 25 * 60,
    breakDuration: 5 * 60,
    type: .focus
)

// 取得今日統計
let todayStats = coordinator.sharedData.getTodayStats()
```

Step 3: 事件通訊

發送事件

```
swift

// 跑步完成
coordinator.sendEvent(.runCompleted(minutes: 20))

// 番茄完成
coordinator.sendEvent(.pomodoroCompleted(focus: 25, rest: 5))

// 遊戲完成
coordinator.sendEvent(.gameCompleted(score: 2048))
```

訂閱事件

```
swift
```

```

coordinator.subscribeToEvents()
    .sink { event in
        switch event {
        case .energyEarned(let amount, let source):
            print("獲得 \(amount) 能量來自 \(source)")
        case .dailyGoalAchieved:
            showCelebration()
        default:
            break
        }
    }
    .store(in: &cancellables)

```

Step 4: 能量系統整合

賺取能量

```

swift

// 自動計算並給予能量
coordinator.earnEnergy(2, source: .running)

```

消耗能量

```

swift

// 檢查並消耗能量
if coordinator.spendEnergy(1) {
    // 能量足夠，開始遊戲
    startGame()
} else {
    // 能量不足，顯示提示
    showEnergyInsufficient()
}

```

團隊分工整合點

成員 A - 跑步模組

```

swift

```

```

// RunningModuleView.swift
struct RunningModuleView: View {
    @EnvironmentObject var coordinator: ModuleCoordinator

    var body: some View {
        VStack {
            // 使用共用計時器
            Text(coordinator.sharedTimer.formattedTime)
                .font(Theme.Typography.timerDisplay)

            PrimaryButton("開始跑步") {
                startRunning()
            }
        }
    }

    func startRunning() {
        // 1. 通知協調器
        coordinator.startModule(.running)

        // 2. 使用共用服務
        coordinator.sharedTimer.start()
        coordinator.sharedAudio.playBackgroundMusic("running")

        // 3. 節拍器 (如果啟用)
        if coordinator.sharedSettings.metronomeEnabled {
            coordinator.sharedAudio.startMetronome()
        }
    }

    func stopRunning() {
        // 1. 儲存記錄
        let duration = coordinator.sharedTimer.currentTime
        coordinator.sharedData.saveRunningRecord(duration: duration)

        // 2. 發送完成事件 (自動獲得能量)
        coordinator.sendEvent(.runCompleted(minutes: Int(duration/60)))

        // 3. 停止服務
        coordinator.stopModule(.running)
    }
}

```

成員 B - 專注模組

```
// FocusModuleView.swift
```

```
struct FocusModuleView: View {
```

```
    @EnvironmentObject var coordinator: ModuleCoordinator
```

```
    @State private var currentPhase: PomodoroType = .focus
```

```
    var body: some View {
```

```
        VStack {
```

```
            // 階段指示器
```

```
            Text(phaseText)
```

```
                .padding(.horizontal)
```

```
                .background(Capsule().fill(Theme.Colors.secondaryGradient))
```

```
            // 倒數計時
```

```
            Text(coordinator.sharedTimer.formattedTime)
```

```
                .font(Theme.Typography.timerDisplay)
```

```
            PrimaryButton("開始專注") {
```

```
                startPomodoro()
```

```
            }
```

```
        }
```

```
    }
```

```
func startPomodoro() {
```

```
    coordinator.startModule(.focus)
```

```
    // 使用共用設定的時間
```

```
    let minutes = coordinator.sharedSettings.pomodoroFocusMinutes
```

```
    // 倒數計時
```

```
    coordinator.sharedTimer.startCountdown(from: Double(minutes * 60)) {
```

```
        completePomodoro()
```

```
    }
```

```
}
```

```
func completePomodoro() {
```

```
    // 播放完成音效
```

```
    coordinator.sharedAudio.playSound("complete")
```

```
    // 儲存記錄
```

```
    coordinator.sharedData.savePomodoroRecord(
```

```
        focusDuration: Double(coordinator.sharedSettings.pomodoroFocusMinutes * 60),
```

```
        breakDuration: Double(coordinator.sharedSettings.pomodoroShortBreakMinutes * 60),
```

```
        type: currentPhase
```

```
    )
```

```
    // 發送完成事件 (自動獲得能量)
```

```
coordinator.sendEvent(.pomodoroCompleted(  
    focus: coordinator.sharedSettings.pomodoroFocusMinutes,  
    rest: coordinator.sharedSettings.pomodoroShortBreakMinutes  
))  
  
// 切換階段  
switchPhase()  
}  
}
```

成員 C - 遊戲模組

swift


```
// GameModuleView.swift

struct GameModuleView: View {
    @EnvironmentObject var coordinator: ModuleCoordinator
    @State private var gameScore = 0

    var body: some View {
        VStack {
            // 顯示能量
            HStack {
                Image(systemName: "bolt.fill")
                Text("能量: \(coordinator.globalEnergy)")
            }

            // 開始遊戲按鈕
            PrimaryButton(
                "開始遊戲 (消耗1能量)",
                isEnabled: coordinator.globalEnergy > 1
            ) {
                tryStartGame()
            }

            // 遊戲畫面
            if isPlaying {
                Game2048View(onScoreUpdate: updateScore)
            }
        }
    }

    func tryStartGame() {
        // 檢查並消耗能量
        let requiredEnergy = coordinator.sharedSettings.gameEnergyPerPlay

        if coordinator.spendEnergy(requiredEnergy) {
            // 能量足夠，開始遊戲
            coordinator.startModule(.game)
            coordinator.sharedHaptic.impact(.medium)
            startNewGame()
        } else {
            // 能量不足，顯示提示
            coordinator.sharedHaptic.notification(.warning)
            showEnergyAlert()
        }
    }

    func endGame() {
        // 儲存遊戲記錄
    }
}
```

```

coordinator.sharedData.saveGameRecord(
    score: gameScore,
    highestTile: getHighestTile(),
    duration: getGameDuration()
)

// 發送完成事件
coordinator.sendEvent(.gameCompleted(score: gameScore))

// 如果分數超過門檻，獲得額外能量
if gameScore > coordinator.sharedSettings.gameScoreThreshold {
    coordinator.earnEnergy(1, source: .game)
}

coordinator.stopModule(.game)
}
}

```

使用共同設計系統

顏色系統

```

swift

// 主要漸層
Theme.Colors.primaryGradient // 紫色漸層 (跑步)
Theme.Colors.secondaryGradient // 粉色漸層 (專注)
Theme.Colors.successGradient // 藍色漸層 (遊戲)

// 背景色
Theme.Colors.bgDark // 深色背景
Theme.Colors.bgMedium // 中等背景
Theme.Colors.bgLight // 淺色背景

// 文字色
Theme.Colors.textPrimary // 主要文字
Theme.Colors.textSecondary // 次要文字
Theme.Colors.accent // 強調色

```

字體系統

```

swift

```

```
ThemeTypography.largeTitle // 大標題
ThemeTypography.title1      // 標題1
ThemeTypography.headline    // 標題
ThemeTypography.body        // 內文
ThemeTypography.caption1    // 小字

// 特殊字體
ThemeTypography.timerDisplay // 計時器顯示
ThemeTypography.statNumber   // 統計數字
```

間距系統

```
swift

ThemeSpacing.xxs // 4
ThemeSpacing.xs  // 8
ThemeSpacing.sm  // 12
ThemeSpacing.md  // 16
ThemeSpacing.lg  // 20
ThemeSpacing.xl  // 24
ThemeSpacing.xxl // 32
```

共用元件

```
swift

// 玻璃卡片
GlassCard {
    // 內容
}

// 主要按鈕
PrimaryButton("按鈕文字", icon: "play.fill") {
    // 動作
}

// 圓形進度
CircularProgress(progress: 0.65, lineWidth: 15, size: 250)

// 統計藥丸
StatPill(icon: "flame.fill", label: "連續", value: "7天")
```

啟動整合

在 App.swift 中初始化

swift

```
@main
struct FocusFlowApp: App {
    @StateObject private var coordinator = ModuleCoordinator.shared

    var body: some Scene {
        WindowGroup {
            IntegratedContentView()
                .environmentObject(coordinator)
        }
        .modelContainer(sharedModelContainer)
    }
}
```

在主視圖中整合

swift

```

struct IntegratedContentView: View {
    @StateObject private var coordinator = ModuleCoordinator.shared
    @State private var selectedTab: ModuleType = .running

    var body: some View {
        ZStack {
            // 背景
            BackgroundView()

            VStack {
                // 全域狀態列 (能量、連續天數、金幣)
                GlobalStatusBar()

                // 每日進度卡片
                DailyProgressCard()

                // 模組內容
                ModuleContainer(selectedModule: selectedTab)

                // 標籤列
                IntegratedTabBar(selectedTab: $selectedTab)
            }
        }
        .environmentObject(coordinator)
    }
}

```

資料流程圖



✅ 整合檢查清單

- ☐ 實作 FlowModule 協定
- ☐ 使用 ModuleCoordinator 管理狀態
- ☐ 使用共用計時器服務
- ☐ 使用共用音效服務
- ☐ 整合能量系統
- ☐ 發送適當的事件
- ☐ 使用共同設計系統
- ☐ 儲存資料到 SwiftData
- ☐ 測試模組間切換
- ☐ 測試能量流轉

🐛 常見問題

Q: 如何在模組間共享資料？

A: 使用 `ModuleCoordinator` 的共享狀態或發送事件。

Q: 如何確保只有一個模組在運行？

A: `startModule()` 會自動停止其他模組。

Q: 如何自訂能量獲得規則？

A: 在 `processEvent()` 中修改能量計算邏輯。

Q: 如何添加新的共用服務？

A: 在 `ModuleCoordinator` 中建立新的服務實例。

☎ 聯絡與支援

整合過程中有任何問題，請隨時在團隊群組討論！

Happy Coding! 🚀