**supabase URL**: https://wujlbjrouqcpnifbakmw.supabase.co

**supabase API** : eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6Ind1amxianJvdXFjcG5pZmJha213Iiwicm9sZSI6ImFub24iLCJpYXQiOjE3NTE4MTMxNjcsImV4cCI6MjA2NzM4OTE2N30.2-l82gsxWDLMj3gUnSpj8sHddMLtX-JgqrbnY5c\_9bg  
**supabase token**: sbp\_68f11523945c2923685a61f81ec6613395f5f5be

# Invest \_V3 — 台灣去中心化模擬投資競賽平台

## 1. 產品定位與願景

**使命**：以 LINE 式易用體驗，結合.透明度，讓台灣投資社群可以在安全合規環境下進行模擬競賽、閱讀高品質內容並回饋創作者。

**核心差異化**

1. **台股專屬**：資料源採證交所（TWSE/TPEx）與 TEJ 及 Alpha Vantage 台股擴充 API。
2. **Web3 透明度**：所有買賣指令、入會費、抖內與紅利記錄皆上鏈（Supabase）並以 Supabase 快取／備份。
3. **Creator Economy**：類 Seeking Alpha 的內容＋抖內／分潤機制，支援每日免費額度與付費訂閱。

## 2. 里程碑規劃

| **時程** | **版本** | **MVP 目標** | **次要功能** | **依賴 & 風險** |
| --- | --- | --- | --- | --- |
| M1 (0‑4 週) | **v0.1** | • iOS 框架 (5 Tab) • Supabase 架構 & RLS • 台股報價快取 (拉 5 檔) • 模擬組合 CRUD • 群組聊天 (無即時) | — | • 台股 API 需轉檔 • RLS 測試 |
| M2 (5‑8 週) | **v0.2** | • 即時聊天室 (Supabase Realtime) • 投資指令解析與回測 • 排名榜單 (每週/季) • InfoView 免費文章 (3/日) | • Headless CMS 整合 | • 即時訊息延遲 |
| M3 (9‑12 週) | **v0.3** | • 文章付費牆 + 訂閱 In‑App Purchase • 抖內禮物飛入動畫 • Wallet ‑ 模擬 NTD 餘額 • KYC Placeholder | — | • IAP Sandbox 流程 |
| M4 (13‑16 週) | **v1.0** | • Supabase tx 簽名（指令/抖內） • 主持人提領流程 (後台) • iPad Layout & 深色模式 • 法遵聲明／隱私 | • Governance token 草稿 | • 加密錢包 UX |

💡 **金流整合（街口/藍新）置於 Post‑MVP**，先以模擬餘額 + Apple IAP 付費牆驗證商業模式。

## 3. 系統架構

flowchart LR

subgraph iOS

A[SwiftUI App]

end

subgraph Cloud

B[Supabase<br/>Postgres • RLS • Realtime • Storage]

C[Edge Functions<br/>TypeScript Cron]

end

subgraph Chain

D[Supabase RPC]

end

subgraph Data

E[Alpha Vantage / TWSE]

end

A -- REST/WS --> B

B -- trigger --> C

C -- tx hash --> D

A -- price feed --> E

### 3.1 前端

* **語言/框架**：Swift 5.9、SwiftUI、Combine、@Observable
* **模組化**：每個 Tab 一個 Swift Package (Home, Chat, Info, Wallet, Settings)
* **路由**：NavigationStack + deep‑link，URL Scheme investv3://
* **本地快取**：AppStorage(UserDefaults) + FileCache 圖片
* **CI/CD**：Xcode Cloud → TestFlight –> App Store Connect

### 3.2 後端 (Supabase)

* **表結構**：users, investment\_groups, chat\_messages, portfolio\_transactions, articles, article\_likes, article\_comments, wallet\_transactions
* **RLS**：針對 user\_id 權限，群組操作 via membership 確認
* **Edge Functions**：
  + calculate\_return\_rate.ts — 5 min cron 抓股價計算回報率
  + creator\_bonus.ts — 每日結算閱讀量 / 訂閱分潤

### 3.3 第三方服務

| **功能** | **MVP Provider** | **替代方案** |
| --- | --- | --- |
| 台股報價 | Alpha Vantage (台股終端點) | Fugle API |
| IAP | Apple StoreKit 2 | RevenueCat |
| 推播 | OneSignal Free | Firebase FCM |

## 4. 資料模型（精簡）

-- 文章表

CREATE TABLE articles (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

author\_id UUID REFERENCES users(id),

title TEXT NOT NULL,

summary TEXT,

body TEXT,

category TEXT,

is\_free BOOLEAN DEFAULT TRUE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

-- 虛擬交易

CREATE TABLE portfolio\_transactions (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

user\_id UUID REFERENCES users(id),

symbol TEXT,

action TEXT CHECK (action IN ('buy','sell')),

amount NUMERIC,

price NUMERIC,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

(完整 schema 已收錄於 Supabase SQL)

## 5. 法遵 & 合規

1. **KYC**：採「後綁流程」—— 提領真實 NTD 才需提交身分證 + 第二證件。
2. **證交所授權**：僅用於模擬回測，展示延遲 > 20 分鐘台股價。
3. **金流**：Apple IAP 屬「數位內容／訂閱」；實體 NTD 入會費日後需接街口或藍新並辦理特店。
4. **個資**：Supabase Storage 存頭像，RLS 禁止跨用戶讀寫。

## 6. 待辦與分工（M2 前）

| **Owner** | **Issue** | **PRD Link** |
| --- | --- | --- |
| iOS | InfoView ↔️ Supabase articles | #12 |
| Backend | RLS Policy for portfolio\_transactions | #9 |
| UX | Figma 深色模式 token | #7 |

## 7. UI / UX 詳規（iOS Full‑Stack 實作參考）

✨ **所有尺寸以 iPhone 14 Pro (390×844 pt) 為基準，除非特別標註。**

### 7.1 Design Tokens

| **Token** | **Light** | **Dark** | **說明** |
| --- | --- | --- | --- |
| --brand‑green | #00B900 | #00B900 | 主要 CTA / Toggle ON |
| --brand‑orange | #FD7E14 | #FD7E14 | 付費／警示 CTA |
| --gray‑100 | #F7F7F7 | #121212 | 背景 Layer 1 |
| --gray‑900 | #1E1E1E | #E0E0E0 | 主要文本 |
| Radius | 12 pt | — | 全域圓角 |
| Shadow | 0 2 2 #0000000D | 0 2 2 #00000033 | 卡片 Elevation 1 |

### 7.2 共用元件

| **元件** | **File Name** | **Props** | **行為** |
| --- | --- | --- | --- |
| GreenButton | UI/Buttons/GreenButton.swift | title:String, icon:SF?, isDisabled:Bool | opacity 0.3 on disabled |
| SegmentedTab | UI/Tab/CategorySegment.swift | items:[String], selected:Int | withAnimation(.easeInOut(duration:0.2)) |
| PieChartView | UI/Charts/PieChartView.swift | data:[(label,String,percent,Color)] | 130 ms spring 交互放大 |

### 7.3 各畫面規格

#### Home

* Safe‑area top 54 pt (餘額列)，底部 TabBar 60 pt。
* ChampionCarousel 寬 360 pt，高 130 pt，使用 TabView(.page)。
* 類別篩選 ScrollView(.horizontal)；Item 46×32 pt。
* 群組 GroupCard：內邊距 16 pt，卡片高度 94 pt；使用 HStack + Spacer()；右側 JoinButton 寬 88 pt。

#### Chat

* Message Bubble：
  + Host 指令 isCommand==true → 背景 brand‑blue #007BFF, 白字。
  + 角半徑 host 12 pt, arrow 不做。
* 投資面板
  + 顶部折疊手柄 RoundedRectangle 24 × 4 pt。
  + PieChartView 固定 90 × 90 pt，右側文字列間距 6 pt。

#### Info (文章列表)

| **區域** | **尺寸** | **注意事項** |
| --- | --- | --- |
| 搜尋框 | 343×40 pt | 左 icon 20 pt，字體 body 14 pt |
| ArticleCard | 343×116 pt | 影陰：Elevation 1 |
| Tag | 46×20 pt | caption2 字體，圓角 10 pt |
| 推薦作者 | 卡片寬 70 pt | 按鈕 FollowButton ↔️ FollowingButton 動畫 fade |

#### Wallet

* 餘額標題右對齊；使用 Text + .monospacedDigit() 保持對齊。
* GiftStoreCell：列表寬 327 pt，高 92 pt；左 Emoji 48 pt。
* 提領彈窗：使用 Alert；二次確認 Destructive style。

#### Settings

* 個人頭像圖檔裁切為 512×512 JPG。
* QR Code 放大視圖：背景 systemBackground 90% → 白卡片圓角 24 pt。
* 通知開關使用 ToggleStyle(.switch) + tint: brand‑green。

### 7.4 動效與手勢

| **元件** | **動效** | **參數** |
| --- | --- | --- |
| Tab 切換 | slide | 0.15 s |
| Join Group | scale (0.1) → 1 | spring 0.4 |
| 禮物飛入 | offset(y:-120) + opacity | 0.5 s easeOut |

### 7.5 無障礙／在地化

* 使用 Text("主頁", tableName:"Localizable")；語系檔 zh‑Hant, en。
* Dynamic Type 支援：字體採 Font.body、Font.title3。
* VoiceOver：message bubble accessibilityLabel 組合「Tom 說：文字內容」。

### 7.6 Figma 檔案結構（for Handoff）

📁 Invest V3

├─ 🌐 Design Tokens

├─ 📱 iOS / iPhone14Pro / Light

│ ├─ Home

│ ├─ Chat

│ ├─ Info

│ ├─ Wallet

│ └─ Settings

├─ 📱 iOS / iPhone14Pro / Dark

└─ 🖥 Icons & Emoji Assets

每個 Component 皆命名 Prefix\_ComponentName; 例如 Info\_ArticleCard / Default。

若需補充 **API Contract** 或 **Storyboard Flowchart**，請再告訴我！

**Invest\_V3 平台月結算創作者分潤方案**

1. 收費邏輯與訂閱收益分配比例

\*\*平台訂閱模式：Invest\_V3 採用月訂閱制，訂閱用戶每月支付固定費用即可無限閱讀平台所有台股分析文章；非訂閱用戶每月可免費閱讀 3 篇文章，其後需等下月或成為訂閱者方可繼續閱讀。為了鼓勵優質內容生產者，平台將訂閱收入按一定比例在平台和作者之間分潤。建議平台從訂閱收入中抽取約20%～30%作為營運成本，剩餘約70%～80%作為「作者收益池」。例如，可設定平台抽成 30%，作者分潤 70%\*\*的比例（類似 YouTube 對超級留言實行約30%分成，創作者獲得70%

hollyland.com；YouTube 廣告收益則是平台45%，創作者55%web.tapereal.com

）。這樣的平台抽成比例既能覆蓋平台成本，又能確保作者獲得大部分收益，具有一定競爭力。 訂閱收入分配機制：作者收益池將依據付費用戶的實際閱讀量在作者間進行二次分配。具體而言，每位付費訂閱者每月訂閱費中分潤給作者的部分可視為一個獨立「用戶貢獻額」。平台會統計每位訂閱者當月閱讀了哪些作者的文章，並將該訂閱者的貢獻額按照閱讀情況分配給相應作者

about.seekingalpha.com

。這種機制類似 Seeking Alpha 等內容平台的做法：官方將每月訂閱收入先均分給每個付費用戶，再根據用戶閱讀各文章的比例，把他的那份預算分配給各作者

about.seekingalpha.com

。例如，Seeking Alpha 提供的示例是假設每位付費用戶帶來$10的作者分潤池：如果用戶當月只讀了2篇文章，那每篇文章的作者可得$5；若讀了10篇，每篇作者僅得約$1

about.seekingalpha.com

。同理，在 Invest\_V3 中，我們可以假設每位訂閱者每月有NT$300分潤額分配給作者：讀得少（例如2篇）時每篇作者可得NT$150，讀得多（例如10篇）時每篇僅約NT$30，總額不變地由該用戶帶給作者群

about.seekingalpha.com

。如此一來，「付費訂閱者觀看數」成為分潤的核心依據：某作者被越多付費用戶閱讀，其分潤越高。這種模型鼓勵作者創作優質內容以吸引訂閱用戶閱讀，同時避免單純以流量論英雄（免費流量不直接產生收益）。 特別情境處理：需要注意的是，非訂閱用戶的免費閱讀（每月3篇）雖然增加總閱讀量，但因未付費不直接產生收益。這部分瀏覽可供作者了解文章熱度，但在收益結算時應區分對待

about.seekingalpha.com

。只有付費用戶的閱讀量才計入作者收益池的分配，確保訂閱收入分配與付費閱讀行為直接掛鉤

about.seekingalpha.com

。為平衡作者生態，平台也可考慮設計一定的保底機制或激勵措施，例如：對於小眾但有價值的題材（如極少人覆蓋的冷門個股分析），提供額外獎金獎勵，類似 Seeking Alpha 對罕見標的文章給予固定稿酬的做法

about.seekingalpha.com

about.seekingalpha.com

。這將鼓勵作者不只追求熱門話題，也樂於產出多元內容。

2. 觀看數據記錄與結算（反作弊措施）

數據記錄方式：為了準確計算分潤，平台將建立嚴格的觀看數據記錄系統。每當用戶閱讀文章，後端會記錄文章ID、讀者用戶ID（若為未登入的免費用戶則記設備ID或瀏覽器指紋）、閱讀時間戳等資訊。計數規則採用「唯一付費閱讀」原則：每位付費訂閱者對同一篇文章只計一次有效閱讀，避免同一用戶反覆刷新增加計數

about.seekingalpha.com

。同時，可設定有效閱讀標準：例如閱讀時間超過一定秒數或閱讀進度達到文章的一定比例，才算一次有效閱讀，以過濾點開即關閉的流量造假。平台後台將每日彙總各文章的付費用戶閱讀次數，作為月度結算的基礎數據。 結算流程：每月結束時，系統針對當月所有文章累計的付費用戶閱讀數進行結算運算，計算各作者應得的訂閱分潤。具體步驟包括：①統計每位付費訂閱者當月閱讀了哪些作者的哪些文章；②計算每位訂閱者的貢獻額在各作者間的分配（如前述，每位訂閱者的NT$300按照其閱讀文章平分給所讀文章的作者）；③彙總所有訂閱者對每位作者的貢獻額，得到每位作者當月的訂閱分潤收益。這套計算應全程自動化完成，同時計入抖內收入（見下一節）以得出作者總收益。 \*\*反作弊與防造假：\*\*為維護分潤機制的公正，必須防範惡意刷流與數據造假。以下是建議的反作弊措施：

\*\*帳號與裝置限制：\*\*強制用戶登入後才能計算付費閱讀次數，並對每個帳號每篇文章僅記一次。在免費閱讀限額機制下，防止用戶透過反覆註冊新帳號無限閱讀——可考慮手機驗證或Email驗證以提高註冊成本。

\*\*唯一訪問計算：\*\*採用裝置指紋和IP檢查，防止單一用戶切換帳號或未登入狀態反覆閱讀同一文章計數。Seeking Alpha 就明確以「每用戶每頁只算一次Pageview」為標準

about.seekingalpha.com

。我們的平台也應類似，只計有效獨立閲讀。

\*\*閱讀行為檢測：\*\*透過前端腳本監測閱讀時間和滾動深度，如發現大量異常快速的點擊/跳出行為，可判斷為機器人流量並過濾掉。只有當用戶瀏覽時間超過例如30秒或閱讀過一定比例內容，才將此計入有效觀看。

流量模式分析：後端建立監控程序，分析各文章的流量來源和模式。如果某作者的文章在極短時間內出現異常高的付費閱讀量，而來源IP段或裝置相似度極高，則有可能是惡意刷流。平台可暫時凍結該作者當期分潤，進一步人工審核可疑流量的真實性。

\*\*信用與懲罰機制：\*\*建立作者信用評分，如發現作者涉嫌組織假流量（例如自己註冊大量帳號訂閱後專刷自身文章），一經核實可扣減其收益或暫停其分潤資格，情節嚴重者封禁帳號。反之，鼓勵作者通過正當手段宣傳文章吸引真實讀者。

\*\*資料透明與備查：\*\*每月結算完成後，我們將關鍵統計（例如全站總付費閱讀人次、各作者排名等）記錄在Supabase鏈上，以公開透明的方式讓社群監督。一旦數據上鏈即不可篡改，這也防止內部人員作弊調整數據。同時備份所有原始日志，以備日後審計。Seeking Alpha 提到即使系統故障也會補回所有流量與收益數據

about.seekingalpha.com

，我們的平台也應有類似機制確保數據可靠。

通過上述多層防護，Invest\_V3 能準確記錄真實的觀看數據並杜絕絕大多數作弊行為，確保分潤機制公平可信。

3. 抖內機制與金流流程

\*\*抖內功能設計：平台提供「抖內」（打賞）\*\*機制，允許讀者對喜愛的作者給予直接金錢支持。用戶可在作者的文章頁面或個人頁面點擊「抖內」按鈕，選擇預設金額（如NT$50、NT$100、NT$500等）或自定義金額，向該作者貢獻心意。抖內既適用於訂閱者也適用於非訂閱用戶，打造雙軌收益：訂閱收入反映內容長期價值，抖內收入則允許用戶即時獎勵優質內容。 支付渠道與流程：目前平台已支持模擬的新臺幣付款和 Apple 內購 (In-App Purchase) 機制。未來可進一步接入本地第三方支付，例如街口支付 (JKOPay) 或藍新金流 (NewebPay) 等：

\*\*Apple IAP：\*\*使用iOS App的用戶可直接透過Apple內購完成訂閱或抖內支付。需要注意Apple對數位內容交易抽取30%手續費，故透過IAP的交易淨收入僅70%進入平台

hollyland.com

。例如用戶在App內抖內NT$100，Apple會先收取NT$30手續費，平台實際收到約NT$70。

\*\*信用卡/第三方支付：\*\*在網站或Android端，建議使用信用卡支付閘道（藍新金流）或行動支付錢包（街口等）。這些渠道通常手續費較低（如信用卡約2%~3%），資金將直接進入平台的收款帳戶，由平台服務器記錄此次抖內交易。

內部錢包與模擬幣：平台可選擇實作用戶錢包，允許用戶先充值一筆金額到其平台帳戶，再用於訂閱或抖內。充值可透過上述各種金流渠道。這種方式的好處是可以減少小額交易手續費累積，同時用戶體驗更流暢（多次抖內不需每次支付流程）。錢包餘額以NTD計價（模擬NTD）並與實際1:100，確保用戶理解資金價值。

\*\*付款過程：\*\*用戶確認抖內金額後，選擇支付方式完成付款。平台服務端收到第三方支付成功回調後，在資料庫中記錄一筆抖內記錄：包括捐助人、受益作者、金額、時間等。同時更新作者的錢包餘額或收入統計。

收益分配與平台抽成：抖內收入主要是讀者對特定作者的打賞，因此該筆金額在扣除必要手續費後，絕大部分應計入該作者收入。我們建議平台對抖內金流抽成10%以內，優先保障創作者收益。例如，透過網站用信用卡抖內NT$100，扣除3%支付手續費還剩NT$97，平台再從中抽取7%（約NT$6.8），最終作者得到約NT$90（約佔原始金額90%）。對比YouTube超級留言約30%的平台分成

hollyland.com

和Patreon約8%~12%的佣金

en.wikipedia.org

，我們的平台抽成設定在10%上下屬於較合理範圍，可提升作者參與積極性。同時需考慮Apple渠道的特殊情況：由於Apple已抽取30%，平台可視情況減免自家抽成或僅象徵性抽取5%等，以免雙重扣費導致作者收益過低（YouTube在iOS上實際到創作者僅約50%

reddit.com

就是因為雙重抽成）。總之，抖內收益盡可能地直接回饋給作者，平台僅保留必要成本部分。 金流監管與安全：平台在集成各支付渠道時，需遵守相關法規（例如金融監管、Apple審核指南等）。所有金流操作須有完善的風險控管：如防範洗錢（一次性大額抖內需人工複核）、防盜刷（異常支付行為的風控）、以及提供支付憑證和發票功能等。每筆抖內交易完成後，系統可發送通知給作者和捐助者確認（例如App推播或電子郵件收據）。在資金流轉方面，平台可每日/每週將累計的抖內款項結算至平台自有帳戶，再於月結時與訂閱分潤一同發放給作者（詳見下一節）。

4. 分潤結算週期與提領門檻

結算週期：建議採用月結算機制。每月最後一天為結算日，統計本月每位作者的收益（包括訂閱分潤和抖內收入）。在結算日後，平台會將各作者的最終收益數據上鏈存證，並在後台生成可提領餘額。具體流程如下：

\*\*月度統計計算：\*\*月末時後端系統彙總本月訂閱分潤結果與抖內收入。訂閱分潤部分依照前述算法按觀看數據計算；抖內部分直接按實際收到金額（扣除平台抽成後）累計給對應作者。

記錄：將結算後的收益結果透過Supabase .進行記錄。一種做法是，每位作者的當月總收益作為一筆交易數據寫入Supabase上智能合約或備忘錄帳本中

about.seekingalpha.com

。例如，可部署一個簡單的Supabase智慧合約，保存<作者ID, 月份, 收益金額>的對應關係。這樣所有人都可以在.瀏覽器上查驗每月的分潤總額分配（作者ID可做匿名處理，以隱私散列表示）。.記錄不可竄改，保障分潤數據的公開透明，同時也防止平台事後更改數據或侵占作者收益。

\*\*生成結算單與通知：\*\*平台後台為每位作者生成電子結算單，列出本月訂閱分潤、抖內收入、平台扣除（如手續費、稅金）等明細以及最終可提領金額。系統通知作者本月收益結算完成，可前往錢包頁查看並申請提領。

提領申請與發放：作者在後台錢包頁面點擊「申請提領」即可提取收益。為了降低手續成本，平台應設定提領門檻：例如最低 NT$1,000或其他適當金額。若作者當月收益未達門檻，則累積至下月。一旦作者申請提領且符合門檻，平台財務將在一定工作日內處理打款。打款方式可包括：銀行轉帳（作者提供銀行帳戶資訊，由平台匯款新台幣）、電子支付轉帳（例如直接轉至作者的街口錢包）但亦需考慮作者的偏好與法規稅務要求，所以可將多種提領方式提供給作者選擇。

\*\*稅務與KYC：\*\*在作者提領前，平台應確認其已完成必要的身份認證與稅務資訊填寫（例如填寫個人/公司名稱、統編或稅號等）。如有法定扣繳義務（例如超過一定金額需預扣所得稅），平台也需在發放時依法辦理。同時提示作者自行申報所得。Seeking Alpha 就要求作者在付款資料齊全才能付款，且未提領金額逾12個月未領取將註銷

about.seekingalpha.com

。Invest\_V3 也可採用類似做法：未及時提領的累積收益如超過一定期限（如12個月）未領取且無回覆，將視為放棄處理（上鏈記錄則永久保留以昭公信）。

通過上述結算流程，平台做到每月固定頻率為創作者結算收益，及時發放、透明上鏈。月結算頻率既平衡了現金流壓力，又對作者而言有穩定的收入預期。提領門檻的設置則避免了過小額支付的繁瑣，同時鼓勵作者持續創作以達到門檻。整套機制配合Supabase上鏈記錄，使得每筆分潤都有跡可循，進一步提高作者對平台的信任。

5. UI 顯示方式建議

文章列表及內容頁：為避免創作者過度追逐點擊量以及防止讀者產生從眾心理，Invest\_V3 的文章卡片和文章頁面將不公開顯示精確的閱讀次數。在文章列表中，每篇文章卡片建議呈現的資訊包括：標題、封面縮圖、摘要、作者名、發佈日期、主題標籤等，但不展示閱讀人次。取而代之，可加入品質互動指標如按讚數或留言數（這些指標相對難以透過機器大量造假，且能反映內容互動度）。另外，為突顯付費機制，文章卡片上對於需要訂閱才能閱讀的內容，可加註「訂閱專區」或鎖定圖示，提示非訂閱者該文章受限。同時，在非訂閱用戶閱讀文章時，閱讀器上方可顯示剩餘免費閱讀篇數（如「本月免費閱讀剩餘：2/3」），增強用戶對自身免費額度的認知，促進轉化。當免費額度用盡時，文章內容的後半部分將自動隱藏並顯示引導訊息（例如「成為訂閱會員即可繼續閱讀全文」的提示按鈕）。 創作者個人頁面：作者的個人主頁將展示其內容成果與影響力，但同樣不直接公開文章總閱讀量等敏感數據。取而代之，可公開顯示作者的訂閱者人數（這裡的「訂閱者」指關注該作者的用戶數，包括付費訂閱用戶和免費關注者）。類似YouTube頻道頁會顯示訂閱人數，這能作為作者影響力的體現。同時可列出作者歷史文章列表（可按發佈時間或熱度排序），每篇顯示標題、發佈日期、按讚/留言概況等。讀者在作者頁面也可選擇關注（訂閱）該作者，收到新文章通知。需要強調的是，文章的閱讀次數僅供作者本人在後台查看，不會在前端公佈，以免引發文章間的惡性比較。 作者後台錢包頁：為了讓作者清晰了解自身收益，平台提供專門的錢包/收益儀表板，僅作者本人可見。該頁面重點展示：

\*\*累計收入：\*\*作者至今通過平台獲得的總收益（金額），以及當前可提領餘額。可進一步細分為「訂閱分潤收入」和「讀者抖內收入」兩部分，並以圖表或數字並列方式呈現比例。

當月收益統計：顯示本月迄今累積的收益，對比上月同期變化。並提供月度明細表格，例如：本月訂閱分潤NT$X，抖內NT$Y，合計NT$Z；平台抽成和稅費總計NT$W，預計發放NT$Z-W。

\*\*訂閱者/粉絲數：\*\*顯示關注該作者的用戶總數，以及其中付費訂閱者人數。這可讓作者了解自己的付費讀者群規模。比如「追蹤你的用戶：500（其中付費會員：120）」。

\*\*文章表現：\*\*列出最近幾篇文章的核心數據（僅作者可見）：如每篇文章的付費閱讀人次、免費閱讀人次、該篇帶來的訂閱分潤金額、獲得的抖內金額、按讚數、評論數等。方便作者了解哪些內容最受付費讀者歡迎，從而優化選題方向。

\*\*提領操作：\*\*顯示目前可提領餘額和提領門檻，當達到門檻時提供「請求提領」按鈕。一旦點擊，彈出確認資訊（包含下次付款日期等）。提領申請提交後，作者可在此查看提領狀態（如「處理中」「已發放」等），並可查閱歷史提領紀錄和對應.交易ID（若採用加密支付，上鏈交易可供查證）。

\*\*系統公告與指南：\*\*在錢包頁側欄或下方，可放置平台分潤機制說明、常見問題解答（FAQ）連結，如關於稅務申報、平台抽成比例調整通知等，方便作者隨時查閱。

讀者端錢包/帳戶頁：雖然問題重點在創作者分潤，但為完整性也簡述讀者端UI建議：訂閱者用戶在其個人帳戶頁面可以查看自己的訂閱狀態（例如剩餘多少天續訂）、歷史支付記錄，以及本月已閱讀文章數（提醒用戶充分利用訂閱權益）。若用戶有在平台充值錢包或購買代幣，也應能在此查看餘額和消費明細。這些功能有助於提升用戶對自身行為的掌控感。 總的來說，UI 設計遵循\*\*「前臺淡化數據、後台強化數據」\*\*的原則：前臺僅展示能促進良性互動和轉化的資訊（如作者影響力、互動程度），而將詳細的流量與收益數據收斂在作者自己的後台中。這樣既維持了社群氛圍的友好（不讓閱讀量變成攀比焦點），又保障了作者對自身表現的充分瞭解。

6. 與 Seeking Alpha / YouTube 模式比較的優勢與限制

我們將Invest\_V3的分潤方案與兩個知名平台做對比，以闡明其優劣： 👉 與 Seeking Alpha 模式比較：Seeking Alpha 是專注投資研究的平台，採用「訂閱+稿酬」混合模式。其優勢在於有固定稿酬獎勵冷門股票分析（每篇固定$45-$65獎金）以及基於付費用戶閱讀量的變動分潤

about.seekingalpha.com

。這確保作者不致於只追熱門題材，小眾內容也有基本收益。然而SA的限制包括：作者需通過嚴格審核才能加入，內容有編輯門檻；非訂閱者只能月看2篇，閱讀門檻高；分潤門檻也相對高（最低$100才能付款

about.seekingalpha.com

）。相比而言，Invest\_V3 降低了准入門檻，鼓勵更多投資達人分享見解，同時沿用按付費閱讀分配收益的核心模型，確保「讀者付費意願」與「作者收入」直接掛鈎，體現公平。不同的是，我們目前沒有像SA那樣的固定稿酬機制，這在鼓勵冷門優質內容方面可能略遜一籌——熱門文章的作者收益會遠高於冷門文章。為此，我們未來可以考慮增加例如「新人獎勵」或「優質冷門話題獎勵」等機制加以平衡。此外，在透明度上，我們透過.上鏈，使得收益分配更公開，這是SA所沒有的創新點。整體而言，Invest\_V3 模式借鑑了 Seeking Alpha 精髓（付費瀏覽分潤模式

about.seekingalpha.com

），但提供了更開放的創作環境與透明度，對創作者更友好。 👉 與 YouTube 模式比較：YouTube 作為影音平台，主要靠廣告變現，輔以頻道會員訂閱與超級留言抖內等。其優勢在於巨大的免費用戶流量和成熟的變現體系：創作者通過廣告點擊獲得收益（YouTube廣告收益分成55%給創作者

web.tapereal.com

），且有億萬潛在觀眾，內容易於病毒式傳播。YouTube 也允許創作者透過會員（類似訂閱者，每月付費獲取專屬內容）和直播打賞（Super Chat 超級留言）賺取收入

web.tapereal.com

。然而，其限制在於：對一般影片創作者，廣告收入受制於演算法和廣告主預算波動，不確定性大；新創作者需達到一定門檻（如1000訂閱、4000小時觀看）才可開啟變現，初期零收入。相比之下，Invest\_V3 的付費訂閱制確保內容直接由讀者買單，收益相對穩定且可預期（只要有訂閱者閱讀，就有固定分潤，不受廣告市場影響）。同時我們引入的抖內功能，相當於YouTube的超級留言，但適用範圍更廣（不局限於直播場合）。在分成比例上，我們對作者更慷慨：YouTube 直播打賞要抽取約30%

hollyland.com

外加Apple分成，而我們的平台抖內抽成僅約10%，讓作者實得比例更高。劣勢方面，Invest\_V3 屬於付費閲讀的小眾社群，總體流量和用戶基數無法與免費的YouTube相比，爆款內容的傳播範圍有限，創作者難以靠純流量獲取像YouTube那樣的廣告暴利收入。此外，我們不公開閱讀數，這雖然避免了炫耀與攀比，但也失去了一定的社群熱度指標（YouTube上公開的觀看量往往能刺激更多人點擊）。但我們透過展示按讚和評論等方式彌補了這一點，在營造良性社區氛圍和數據隱私上有所側重。 \*\*👉 Invest\_V3 模式自身的優劣：\*\*綜上所述，Invest\_V3 的創作者分潤方案結合了Seeking Alpha的專業付費閱讀分潤模型和YouTube多元變現渠道的優點，同時針對本地台股社群進行優化：

\*\*優勢：\*\*內容專業性強（專注台股分析，有訂閱付費門檻確保用戶質量）、收益與內容質量直接掛鉤（付費閱讀越多收入越高）、同時提供訂閱+抖內雙渠道收入；平台分成合理偏低（作者收益比例高於業界平均）、收益結算透明上鏈增加信任、UI設計保護創作者尊嚴（隱藏瀏覽量避免惡性競爭）。

限制：需要建立足夠大的付費用戶群體才能支撐可觀收益，相比開放平台初期成長可能較慢；沒有廣告收入作補充，全部收益仰賴訂閱和讀者付費，對內容質量要求更高、轉化壓力大；隱藏閱讀量可能降低內容外部傳播性（少了“XX萬人讀過”的社交證明）。另外，我們目前欠缺類似Seeking Alpha的固定獎金激勵機制，可能導致作者傾向討好大眾口味而忽略冷門題材。

總結：Invest\_V3 的分潤機制定位於「以付費閱讀為核心的創作者經濟」，在確保公平成熟的同時注重本地化與創新。相較Seeking Alpha，我們提供更友善的門檻與透明度；相較YouTube，我們強調專業內容價值轉化而非純流量變現。未來隨著用戶和內容生態發展，平台可靈活調整抽成比例、引入更多元的收益手段（如廣告、作者付費課程等）來完善模式。但在當前方案下，Invest\_V3 已具備清晰的分潤邏輯、可靠的數據機制與友好的產品體驗，有望成為台股投資社群中創作者獲利的優質平台。

**githubbcopilt：**

**┌───────────────┐ ┌───────────────┐ ┌───────────────┐**

**│ iOS App │ │ 后端服務 │ │ 外部 API │**

**│ (Swift) │<───>│ (Python) │<───>│ - yfinance │**

**└───────────────┘ └───────┬───────┘ │ - SMS API │**

**┌─┴─┐ └───────────────┘**

**│ │**

**│ DB │**

**│ │**

**└───┘**

**Supabase  
import Foundation**

**// 用戶模型**

**struct User: Codable, Identifiable {**

**let id: String**

**let username: String**

**let phoneNumber: String**

**let initialCapital: Double // 初始資本(100萬)**

**let availableCash: Double // 可用現金**

**let invitationCode: String // 邀請碼**

**let createdAt: Date**

**}**

**// 投資群組模型**

**struct InvestmentGroup: Codable, Identifiable {**

**let id: String**

**let name: String**

**let ownerId: String // 群組主持人ID**

**let description: String**

**let createdAt: Date**

**let members: [String] // 成員ID列表**

**}**

**// 股票持倉模型**

**struct Position: Codable, Identifiable {**

**let id: String**

**let userId: String**

**let symbol: String // 股票代碼**

**let quantity: Int // 持股數量(股)**

**let averageCost: Double // 平均成本**

**let purchaseDate: Date**

**let marketType: String // 市場類型: 上市/上櫃/興櫃**

**}**

**// 交易記錄模型**

**struct Transaction: Codable, Identifiable {**

**let id: String**

**let userId: String**

**let symbol: String // 股票代碼**

**let action: String // 買入/賣出**

**let quantity: Int // 交易數量**

**let price: Double // 交易價格**

**let fee: Double // 手續費**

**let totalAmount: Double // 總金額**

**let timestamp: Date**

**let profit: Double? // 賣出時的獲利/虧損**

**}**

**// 績效快照模型 - 用於排行榜**

**struct PerformanceSnapshot: Codable, Identifiable {**

**let id: String**

**let userId: String**

**let timestamp: Date**

**let portfolioValue: Double // 總資產價值**

**let returnRate: Double // 報酬率**

**let period: String // 週/月/年**

**}  
  
import Foundation**

**import Combine**

**class StockDataService {**

**// 單例模式**

**static let shared = StockDataService()**

**private init() {}**

**private let baseURL = "https://your-backend-api.com/stock"**

**// 獲取即時股價**

**func fetchCurrentPrice(symbol: String) -> AnyPublisher<Double, Error> {**

**let url = URL(string: "\(baseURL)/price?symbol=\(symbol)")!**

**return URLSession.shared.dataTaskPublisher(for: url)**

**.map { $0.data }**

**.decode(type: StockPriceResponse.self, decoder: JSONDecoder())**

**.map { $0.price }**

**.eraseToAnyPublisher()**

**}**

**// 獲取歷史數據**

**func fetchHistoricalData(symbol: String, period: String, interval: String) -> AnyPublisher<[HistoricalDataPoint], Error> {**

**let url = URL(string: "\(baseURL)/history?symbol=\(symbol)&period=\(period)&interval=\(interval)")!**

**return URLSession.shared.dataTaskPublisher(for: url)**

**.map { $0.data }**

**.decode(type: HistoricalDataResponse.self, decoder: JSONDecoder())**

**.map { $0.data }**

**.eraseToAnyPublisher()**

**}**

**// 獲取市場分類(上市/上櫃/興櫃)**

**func getMarketType(symbol: String) -> AnyPublisher<String, Error> {**

**let url = URL(string: "\(baseURL)/market-type?symbol=\(symbol)")!**

**return URLSession.shared.dataTaskPublisher(for: url)**

**.map { $0.data }**

**.decode(type: MarketTypeResponse.self, decoder: JSONDecoder())**

**.map { $0.marketType }**

**.eraseToAnyPublisher()**

**}**

**// 搜索股票**

**func searchStocks(query: String) -> AnyPublisher<[StockInfo], Error> {**

**let url = URL(string: "\(baseURL)/search?query=\(query)")!**

**return URLSession.shared.dataTaskPublisher(for: url)**

**.map { $0.data }**

**.decode(type: SearchResponse.self, decoder: JSONDecoder())**

**.map { $0.results }**

**.eraseToAnyPublisher()**

**}**

**}**

**// 響應模型**

**struct StockPriceResponse: Codable {**

**let symbol: String**

**let price: Double**

**let timestamp: Date**

**}**

**struct HistoricalDataPoint: Codable {**

**let date: Date**

**let open: Double**

**let high: Double**

**let low: Double**

**let close: Double**

**let volume: Int**

**}**

**struct HistoricalDataResponse: Codable {**

**let symbol: String**

**let data: [HistoricalDataPoint]**

**}**

**struct MarketTypeResponse: Codable {**

**let symbol: String**

**let marketType: String**

**}**

**struct StockInfo: Codable, Identifiable {**

**let id: String**

**let symbol: String**

**let name: String**

**let marketType: String**

**}**

**struct SearchResponse: Codable {**

**let results: [StockInfo]**

**}  
  
import Foundation**

**import Combine**

**class TradingService {**

**static let shared = TradingService()**

**private init() {}**

**private let baseURL = "https://your-backend-api.com/trading"**

**// 買入股票**

**func buyStock(userId: String, symbol: String, quantity: Int, price: Double) -> AnyPublisher<Transaction, Error> {**

**let parameters: [String: Any] = [**

**"userId": userId,**

**"symbol": symbol,**

**"quantity": quantity,**

**"price": price,**

**"action": "buy"**

**]**

**guard let url = URL(string: "\(baseURL)/execute") else {**

**return Fail(error: URLError(.badURL)).eraseToAnyPublisher()**

**}**

**var request = URLRequest(url: url)**

**request.httpMethod = "POST"**

**request.setValue("application/json", forHTTPHeaderField: "Content-Type")**

**do {**

**request.httpBody = try JSONSerialization.data(withJSONObject: parameters)**

**} catch {**

**return Fail(error: error).eraseToAnyPublisher()**

**}**

**return URLSession.shared.dataTaskPublisher(for: request)**

**.map { $0.data }**

**.decode(type: Transaction.self, decoder: JSONDecoder())**

**.eraseToAnyPublisher()**

**}**

**// 賣出股票**

**func sellStock(userId: String, symbol: String, quantity: Int, price: Double) -> AnyPublisher<Transaction, Error> {**

**let parameters: [String: Any] = [**

**"userId": userId,**

**"symbol": symbol,**

**"quantity": quantity,**

**"price": price,**

**"action": "sell"**

**]**

**guard let url = URL(string: "\(baseURL)/execute") else {**

**return Fail(error: URLError(.badURL)).eraseToAnyPublisher()**

**}**

**var request = URLRequest(url: url)**

**request.httpMethod = "POST"**

**request.setValue("application/json", forHTTPHeaderField: "Content-Type")**

**do {**

**request.httpBody = try JSONSerialization.data(withJSONObject: parameters)**

**} catch {**

**return Fail(error: error).eraseToAnyPublisher()**

**}**

**return URLSession.shared.dataTaskPublisher(for: request)**

**.map { $0.data }**

**.decode(type: Transaction.self, decoder: JSONDecoder())**

**.eraseToAnyPublisher()**

**}**

**// 獲取用戶持倉**

**func getUserPositions(userId: String) -> AnyPublisher<[Position], Error> {**

**guard let url = URL(string: "\(baseURL)/positions?userId=\(userId)") else {**

**return Fail(error: URLError(.badURL)).eraseToAnyPublisher()**

**}**

**return URLSession.shared.dataTaskPublisher(for: url)**

**.map { $0.data }**

**.decode(type: PositionsResponse.self, decoder: JSONDecoder())**

**.map { $0.positions }**

**.eraseToAnyPublisher()**

**}**

**// 獲取用戶交易歷史**

**func getTransactionHistory(userId: String) -> AnyPublisher<[Transaction], Error> {**

**guard let url = URL(string: "\(baseURL)/history?userId=\(userId)") else {**

**return Fail(error: URLError(.badURL)).eraseToAnyPublisher()**

**}**

**return URLSession.shared.dataTaskPublisher(for: url)**

**.map { $0.data }**

**.decode(type: TransactionsResponse.self, decoder: JSONDecoder())**

**.map { $0.transactions }**

**.eraseToAnyPublisher()**

**}**

**// 獲取績效數據**

**func getPerformance(userId: String, period: String) -> AnyPublisher<PerformanceData, Error> {**

**guard let url = URL(string: "\(baseURL)/performance?userId=\(userId)&period=\(period)") else {**

**return Fail(error: URLError(.badURL)).eraseToAnyPublisher()**

**}**

**return URLSession.shared.dataTaskPublisher(for: url)**

**.map { $0.data }**

**.decode(type: PerformanceData.self, decoder: JSONDecoder())**

**.eraseToAnyPublisher()**

**}**

**}**

**// 響應模型**

**struct PositionsResponse: Codable {**

**let positions: [Position]**

**}**

**struct TransactionsResponse: Codable {**

**let transactions: [Transaction]**

**}**

**struct PerformanceData: Codable {**

**let initialCapital: Double**

**let currentValue: Double**

**let returnRate: Double**

**let weeklyChange: Double**

**let monthlyChange: Double**

**let yearlyChange: Double**

**let ranking: Int**

**let totalUsers: Int**

**}  
  
import Foundation**

**import Combine**

**class UserService {**

**static let shared = UserService()**

**private init() {}**

**private let baseURL = "https://your-backend-api.com/users"**

**// 用戶註冊**

**func registerUser(username: String, phoneNumber: String) -> AnyPublisher<RegistrationResponse, Error> {**

**let parameters: [String: Any] = [**

**"username": username,**

**"phoneNumber": phoneNumber**

**]**

**guard let url = URL(string: "\(baseURL)/register") else {**

**return Fail(error: URLError(.badURL)).eraseToAnyPublisher()**

**}**

**var request = URLRequest(url: url)**

**request.httpMethod = "POST"**

**request.setValue("application/json", forHTTPHeaderField: "Content-Type")**

**do {**

**request.httpBody = try JSONSerialization.data(withJSONObject: parameters)**

**} catch {**

**return Fail(error: error).eraseToAnyPublisher()**

**}**

**return URLSession.shared.dataTaskPublisher(for: request)**

**.map { $0.data }**

**.decode(type: RegistrationResponse.self, decoder: JSONDecoder())**

**.eraseToAnyPublisher()**

**}**

**// 發送SMS驗證碼**

**func sendVerificationCode(phoneNumber: String) -> AnyPublisher<SMSResponse, Error> {**

**let parameters: [String: Any] = [**

**"phoneNumber": phoneNumber**

**]**

**guard let url = URL(string: "\(baseURL)/send-verification") else {**

**return Fail(error: URLError(.badURL)).eraseToAnyPublisher()**

**}**

**var request = URLRequest(url: url)**

**request.httpMethod = "POST"**

**request.setValue("application/json", forHTTPHeaderField: "Content-Type")**

**do {**

**request.httpBody = try JSONSerialization.data(withJSONObject: parameters)**

**} catch {**

**return Fail(error: error).eraseToAnyPublisher()**

**}**

**return URLSession.shared.dataTaskPublisher(for: request)**

**.map { $0.data }**

**.decode(type: SMSResponse.self, decoder: JSONDecoder())**

**.eraseToAnyPublisher()**

**}**

**// 驗證SMS驗證碼**

**func verifyCode(phoneNumber: String, code: String) -> AnyPublisher<VerificationResponse, Error> {**

**let parameters: [String: Any] = [**

**"phoneNumber": phoneNumber,**

**"code": code**

**]**

**guard let url = URL(string: "\(baseURL)/verify-code") else {**

**return Fail(error: URLError(.badURL)).eraseToAnyPublisher()**

**}**

**var request = URLRequest(url: url)**

**request.httpMethod = "POST"**

**request.setValue("application/json", forHTTPHeaderField: "Content-Type")**

**do {**

**request.httpBody = try JSONSerialization.data(withJSONObject: parameters)**

**} catch {**

**return Fail(error: error).eraseToAnyPublisher()**

**}**

**return URLSession.shared.dataTaskPublisher(for: request)**

**.map { $0.data }**

**.decode(type: VerificationResponse.self, decoder: JSONDecoder())**

**.eraseToAnyPublisher()**

**}**

**// 使用邀請碼**

**func useInvitationCode(userId: String, code: String) -> AnyPublisher<InvitationResponse, Error> {**

**let parameters: [String: Any] = [**

**"userId": userId,**

**"invitationCode": code**

**]**

**guard let url = URL(string: "\(baseURL)/use-invitation") else {**

**return Fail(error: URLError(.badURL)).eraseToAnyPublisher()**

**}**

**var request = URLRequest(url: url)**

**request.httpMethod = "POST"**

**request.setValue("application/json", forHTTPHeaderField: "Content-Type")**

**do {**

**request.httpBody = try JSONSerialization.data(withJSONObject: parameters)**

**} catch {**

**return Fail(error: error).eraseToAnyPublisher()**

**}**

**return URLSession.shared.dataTaskPublisher(for: request)**

**.map { $0.data }**

**.decode(type: InvitationResponse.self, decoder: JSONDecoder())**

**.eraseToAnyPublisher()**

**}**

**// 創建投資群組**

**func createInvestmentGroup(name: String, ownerId: String, description: String) -> AnyPublisher<InvestmentGroup, Error> {**

**let parameters: [String: Any] = [**

**"name": name,**

**"ownerId": ownerId,**

**"description": description**

**]**

**guard let url = URL(string: "\(baseURL)/groups") else {**

**return Fail(error: URLError(.badURL)).eraseToAnyPublisher()**

**}**

**var request = URLRequest(url: url)**

**request.httpMethod = "POST"**

**request.setValue("application/json", forHTTPHeaderField: "Content-Type")**

**do {**

**request.httpBody = try JSONSerialization.data(withJSONObject: parameters)**

**} catch {**

**return Fail(error: error).eraseToAnyPublisher()**

**}**

**return URLSession.shared.dataTaskPublisher(for: request)**

**.map { $0.data }**

**.decode(type: InvestmentGroup.self, decoder: JSONDecoder())**

**.eraseToAnyPublisher()**

**}**

**}**

**// 響應模型**

**struct RegistrationResponse: Codable {**

**let userId: String**

**let username: String**

**let initialCapital: Double**

**let invitationCode: String**

**}**

**struct SMSResponse: Codable {**

**let success: Bool**

**let message: String**

**}**

**struct VerificationResponse: Codable {**

**let verified: Bool**

**let userId: String?**

**let token: String?**

**}**

**struct InvitationResponse: Codable {**

**let success: Bool**

**let bonusAmount: Double?**

**let message: String**

**}  
  
import Foundation**

**import Combine**

**class AIAnalysisService {**

**static let shared = AIAnalysisService()**

**private init() {}**

**private let baseURL = "https://your-backend-api.com/ai"**

**// 獲取交易分析**

**func analyzeTransaction(userId: String, transactionId: String) -> AnyPublisher<TradingAnalysis, Error> {**

**guard let url = URL(string: "\(baseURL)/analyze?userId=\(userId)&transactionId=\(transactionId)") else {**

**return Fail(error: URLError(.badURL)).eraseToAnyPublisher()**

**}**

**return URLSession.shared.dataTaskPublisher(for: url)**

**.map { $0.data }**

**.decode(type: TradingAnalysis.self, decoder: JSONDecoder())**

**.eraseToAnyPublisher()**

**}**

**// 獲取課程推薦**

**func getCourseRecommendations(userId: String, profitLoss: Double) -> AnyPublisher<[CourseRecommendation], Error> {**

**guard let url = URL(string: "\(baseURL)/courses?userId=\(userId)&profitLoss=\(profitLoss)") else {**

**return Fail(error: URLError(.badURL)).eraseToAnyPublisher()**

**}**

**return URLSession.shared.dataTaskPublisher(for: url)**

**.map { $0.data }**

**.decode(type: CourseRecommendationsResponse.self, decoder: JSONDecoder())**

**.map { $0.recommendations }**

**.eraseToAnyPublisher()**

**}**

**// 獲取整體投資組合分析**

**func getPortfolioAnalysis(userId: String) -> AnyPublisher<PortfolioAnalysis, Error> {**

**guard let url = URL(string: "\(baseURL)/portfolio?userId=\(userId)") else {**

**return Fail(error: URLError(.badURL)).eraseToAnyPublisher()**

**}**

**return URLSession.shared.dataTaskPublisher(for: url)**

**.map { $0.data }**

**.decode(type: PortfolioAnalysis.self, decoder: JSONDecoder())**

**.eraseToAnyPublisher()**

**}**

**}**

**// 響應模型**

**struct TradingAnalysis: Codable {**

**let transactionId: String**

**let buyPrice: Double**

**let sellPrice: Double**

**let holdingPeriod: Int // 持有天數**

**let profitLoss: Double**

**let profitPercent: Double**

**let analysis: String // AI分析文本**

**let suggestions: [String] // 建議列表**

**let marketTrend: String // 該期間市場走勢**

**let relatedCourses: [CourseRecommendation]?**

**}**

**struct CourseRecommendation: Codable, Identifiable {**

**let id: String**

**let title: String**

**let description: String**

**let price: Double**

**let url: URL**

**let category: String**

**let relevanceScore: Double // 相關度評分**

**}**

**struct CourseRecommendationsResponse: Codable {**

**let recommendations: [CourseRecommendation]**

**}**

**struct PortfolioAnalysis: Codable {**

**let diversificationScore: Double // 多樣化評分**

**let riskScore: Double // 風險評分**

**let sectorAllocation: [SectorAllocation] // 產業配置**

**let suggestions: [String] // 組合優化建議**

**let expectedReturn: Double // 預期報酬率**

**}**

**struct SectorAllocation: Codable {**

**let sector: String**

**let percentage: Double**

**}  
  
import yfinance as yf**

**from datetime import datetime, timedelta**

**import pandas as pd**

**from fastapi import FastAPI, HTTPException, Query**

**app = FastAPI()**

**# 根據台灣股票代號格式化 yfinance 代號 (加上.TW)**

**def format\_symbol(symbol):**

**# 處理台股代號**

**if symbol.isdigit():**

**return f"{symbol}.TW"**

**return symbol**

**@app.get("/stock/price")**

**async def get\_stock\_price(symbol: str):**

**try:**

**formatted\_symbol = format\_symbol(symbol)**

**ticker = yf.Ticker(formatted\_symbol)**

**data = ticker.history(period="1d")**

**if data.empty:**

**raise HTTPException(status\_code=404, detail=f"No data found for symbol {symbol}")**

**latest\_price = data['Close'].iloc[-1]**

**return {**

**"symbol": symbol,**

**"price": latest\_price,**

**"timestamp": datetime.now().isoformat()**

**}**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))**

**@app.get("/stock/history")**

**async def get\_stock\_history(**

**symbol: str,**

**period: str = "1mo",**

**interval: str = "1d"**

**):**

**try:**

**formatted\_symbol = format\_symbol(symbol)**

**ticker = yf.Ticker(formatted\_symbol)**

**# 檢查 period 和 interval 有效性**

**valid\_periods = ["1d", "5d", "1mo", "3mo", "6mo", "1y", "2y", "5y", "10y", "max"]**

**valid\_intervals = ["1m", "2m", "5m", "15m", "30m", "60m", "90m", "1h", "1d", "5d", "1wk", "1mo", "3mo"]**

**if period not in valid\_periods:**

**raise HTTPException(status\_code=400, detail=f"Invalid period. Choose from {valid\_periods}")**

**if interval not in valid\_intervals:**

**raise HTTPException(status\_code=400, detail=f"Invalid interval. Choose from {valid\_intervals}")**

**data = ticker.history(period=period, interval=interval)**

**if data.empty:**

**raise HTTPException(status\_code=404, detail=f"No historical data found for symbol {symbol}")**

**# 轉換為列表格式**

**result = []**

**for index, row in data.iterrows():**

**result.append({**

**"date": index.isoformat(),**

**"open": row["Open"],**

**"high": row["High"],**

**"low": row["Low"],**

**"close": row["Close"],**

**"volume": int(row["Volume"])**

**})**

**return {**

**"symbol": symbol,**

**"data": result**

**}**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))**

**@app.get("/stock/market-type")**

**async def get\_market\_type(symbol: str):**

**# 這裡可以實現一個查詢台灣股票市場類型的功能**

**# 可以通過爬取公開資訊觀測站或其他可靠來源來實現**

**# 簡化版本：**

**try:**

**# 假設我們有一個存儲市場類型的字典**

**market\_types = {**

**# 上市股票範例**

**"2330": "上市",**

**"2317": "上市",**

**"2454": "上市",**

**# 上櫃股票範例**

**"6488": "上櫃",**

**"8299": "上櫃",**

**# 興櫃股票範例**

**"7777": "興櫃"**

**}**

**# 清理輸入符號**

**clean\_symbol = symbol.strip()**

**if clean\_symbol in market\_types:**

**return {"symbol": clean\_symbol, "marketType": market\_types[clean\_symbol]}**

**else:**

**# 如果找不到，可以嘗試從 yfinance 獲取資訊**

**ticker = yf.Ticker(format\_symbol(clean\_symbol))**

**info = ticker.info**

**# 檢查是否為台股**

**if "." in format\_symbol(clean\_symbol) and format\_symbol(clean\_symbol).endswith(".TW"):**

**return {"symbol": clean\_symbol, "marketType": "上市"} # 默認為上市**

**else:**

**return {"symbol": clean\_symbol, "marketType": "其他"}**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))**

**@app.get("/stock/search")**

**async def search\_stocks(query: str):**

**try:**

**# 實現股票搜索功能**

**# 可以通過爬取或使用其他 API 來實現更完整的搜索**

**# 簡化版本：**

**stocks = [**

**{"id": "2330", "symbol": "2330", "name": "台積電", "marketType": "上市"},**

**{"id": "2317", "symbol": "2317", "name": "鴻海", "marketType": "上市"},**

**{"id": "2454", "symbol": "2454", "name": "聯發科", "marketType": "上市"},**

**{"id": "6488", "symbol": "6488", "name": "環球晶", "marketType": "上櫃"},**

**{"id": "8299", "symbol": "8299", "name": "群聯", "marketType": "上櫃"}**

**]**

**# 過濾結果**

**results = []**

**for stock in stocks:**

**if (query.lower() in stock["symbol"].lower() or**

**query.lower() in stock["name"].lower()):**

**results.append(stock)**

**return {"results": results}**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))  
  
from fastapi import FastAPI, HTTPException, Depends, Body**

**from pydantic import BaseModel**

**from typing import List, Optional**

**from datetime import datetime**

**import uuid**

**import os**

**from supabase import create\_client, Client**

**app = FastAPI()**

**# Supabase 連接設定**

**SUPABASE\_URL = os.getenv("SUPABASE\_URL")**

**SUPABASE\_KEY = os.getenv("SUPABASE\_KEY")**

**supabase: Client = create\_client(SUPABASE\_URL, SUPABASE\_KEY)**

**# 交易執行模型**

**class TradeRequest(BaseModel):**

**userId: str**

**symbol: str**

**quantity: int**

**price: float**

**action: str # "buy" or "sell"**

**# 執行交易**

**@app.post("/trading/execute")**

**async def execute\_trade(request: TradeRequest):**

**try:**

**# 獲取用戶資料**

**user\_data = supabase.table("users").select("\*").eq("id", request.userId).single().execute()**

**user = user\_data["data"]**

**if not user:**

**raise HTTPException(status\_code=404, detail="User not found")**

**# 計算交易總額與手續費**

**total\_amount = request.quantity \* request.price**

**fee = total\_amount \* 0.001425 # 台股手續費率**

**# 買入時檢查資金是否足夠**

**if request.action == "buy":**

**total\_cost = total\_amount + fee**

**if user["availableCash"] < total\_cost:**

**raise HTTPException(status\_code=400, detail="Insufficient funds")**

**# 更新用戶可用現金**

**new\_available\_cash = user["availableCash"] - total\_cost**

**supabase.table("users").update({"availableCash": new\_available\_cash}).eq("id", request.userId).execute()**

**# 檢查是否已有該股票持倉**

**position\_data = supabase.table("positions").select("\*").eq("userId", request.userId).eq("symbol", request.symbol).execute()**

**existing\_position = position\_data["data"]**

**if existing\_position and len(existing\_position) > 0:**

**# 更新現有持倉**

**position = existing\_position[0]**

**new\_quantity = position["quantity"] + request.quantity**

**new\_avg\_cost = ((position["quantity"] \* position["averageCost"]) + total\_amount) / new\_quantity**

**supabase.table("positions").update({**

**"quantity": new\_quantity,**

**"averageCost": new\_avg\_cost**

**}).eq("id", position["id"]).execute()**

**else:**

**# 創建新持倉**

**supabase.table("positions").insert({**

**"id": str(uuid.uuid4()),**

**"userId": request.userId,**

**"symbol": request.symbol,**

**"quantity": request.quantity,**

**"averageCost": request.price,**

**"purchaseDate": datetime.now().isoformat(),**

**"marketType": "上市" # 默認值，可以通過市場API獲取實際值**

**}).execute()**

**# 記錄交易**

**transaction = supabase.table("transactions").insert({**

**"id": str(uuid.uuid4()),**

**"userId": request.userId,**

**"symbol": request.symbol,**

**"action": "buy",**

**"quantity": request.quantity,**

**"price": request.price,**

**"fee": fee,**

**"totalAmount": total\_amount,**

**"timestamp": datetime.now().isoformat()**

**}).execute()**

**return transaction["data"][0]**

**elif request.action == "sell":**

**# 檢查是否擁有足夠股票**

**position\_data = supabase.table("positions").select("\*").eq("userId", request.userId).eq("symbol", request.symbol).execute()**

**if not position\_data["data"] or len(position\_data["data"]) == 0:**

**raise HTTPException(status\_code=400, detail="No position found for this stock")**

**position = position\_data["data"][0]**

**if position["quantity"] < request.quantity:**

**raise HTTPException(status\_code=400, detail="Insufficient stock quantity")**

**# 計算獲利/虧損**

**profit = (request.price - position["averageCost"]) \* request.quantity - fee**

**# 更新持倉**

**new\_quantity = position["quantity"] - request.quantity**

**if new\_quantity > 0:**

**# 部分賣出**

**supabase.table("positions").update({**

**"quantity": new\_quantity**

**}).eq("id", position["id"]).execute()**

**else:**

**# 全部賣出**

**supabase.table("positions").delete().eq("id", position["id"]).execute()**

**# 更新用戶可用現金**

**new\_available\_cash = user["availableCash"] + total\_amount - fee**

**supabase.table("users").update({"availableCash": new\_available\_cash}).eq("id", request.userId).execute()**

**# 記錄交易**

**transaction = supabase.table("transactions").insert({**

**"id": str(uuid.uuid4()),**

**"userId": request.userId,**

**"symbol": request.symbol,**

**"action": "sell",**

**"quantity": request.quantity,**

**"price": request.price,**

**"fee": fee,**

**"totalAmount": total\_amount,**

**"timestamp": datetime.now().isoformat(),**

**"profit": profit**

**}).execute()**

**return transaction["data"][0]**

**else:**

**raise HTTPException(status\_code=400, detail="Invalid action")**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))**

**# 獲取用戶持倉**

**@app.get("/trading/positions")**

**async def get\_user\_positions(userId: str):**

**try:**

**position\_data = supabase.table("positions").select("\*").eq("userId", userId).execute()**

**return {"positions": position\_data["data"]}**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))**

**# 獲取交易歷史**

**@app.get("/trading/history")**

**async def get\_transaction\_history(userId: str):**

**try:**

**transaction\_data = supabase.table("transactions").select("\*").eq("userId", userId).order("timestamp", desc=True).execute()**

**return {"transactions": transaction\_data["data"]}**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))**

**# 獲取績效數據**

**@app.get("/trading/performance")**

**async def get\_performance(userId: str, period: str = "all"):**

**try:**

**# 獲取用戶資料**

**user\_data = supabase.table("users").select("\*").eq("id", userId).single().execute()**

**user = user\_data["data"]**

**if not user:**

**raise HTTPException(status\_code=404, detail="User not found")**

**# 獲取用戶持倉**

**position\_data = supabase.table("positions").select("\*").eq("userId", userId).execute()**

**positions = position\_data["data"]**

**# 計算當前持倉總值**

**portfolio\_value = 0**

**for position in positions:**

**# 這裡可以調用股票API獲取即時價格**

**# 簡化處理：使用平均成本**

**position\_value = position["quantity"] \* position["averageCost"]**

**portfolio\_value += position\_value**

**# 計算總資產**

**total\_assets = user["availableCash"] + portfolio\_value**

**# 計算報酬率**

**return\_rate = (total\_assets - user["initialCapital"]) / user["initialCapital"] \* 100**

**# 獲取排名數據**

**ranking\_data = None**

**if period == "week":**

**# 獲取週排行**

**snapshot\_data = supabase.table("performance\_snapshots").select("\*").eq("period", "week").order("returnRate", desc=True).execute()**

**ranking\_data = snapshot\_data["data"]**

**elif period == "month":**

**# 獲取月排行**

**snapshot\_data = supabase.table("performance\_snapshots").select("\*").eq("period", "month").order("returnRate", desc=True).execute()**

**ranking\_data = snapshot\_data["data"]**

**elif period == "year":**

**# 獲取年排行**

**snapshot\_data = supabase.table("performance\_snapshots").select("\*").eq("period", "year").order("returnRate", desc=True).execute()**

**ranking\_data = snapshot\_data["data"]**

**# 計算排名**

**ranking = 1**

**total\_users = 1**

**if ranking\_data:**

**total\_users = len(ranking\_data)**

**for i, item in enumerate(ranking\_data):**

**if item["userId"] == userId:**

**ranking = i + 1**

**break**

**return {**

**"initialCapital": user["initialCapital"],**

**"currentValue": total\_assets,**

**"returnRate": return\_rate,**

**"weeklyChange": 0, # 需要實現週變化計算**

**"monthlyChange": 0, # 需要實現月變化計算**

**"yearlyChange": 0, # 需要實現年變化計算**

**"ranking": ranking,**

**"totalUsers": total\_users**

**}**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))  
  
from fastapi import FastAPI, HTTPException, Query**

**from pydantic import BaseModel**

**from typing import List, Optional**

**import uuid**

**import os**

**import json**

**import pandas as pd**

**import numpy as np**

**from datetime import datetime, timedelta**

**import yfinance as yf**

**from supabase import create\_client, Client**

**app = FastAPI()**

**# Supabase 連接設定**

**SUPABASE\_URL = os.getenv("SUPABASE\_URL")**

**SUPABASE\_KEY = os.getenv("SUPABASE\_KEY")**

**supabase: Client = create\_client(SUPABASE\_URL, SUPABASE\_KEY)**

**# 根據台灣股票代號格式化 yfinance 代號**

**def format\_symbol(symbol):**

**if symbol.isdigit():**

**return f"{symbol}.TW"**

**return symbol**

**@app.get("/ai/analyze")**

**async def analyze\_transaction(userId: str, transactionId: str):**

**try:**

**# 獲取交易數據**

**transaction\_data = supabase.table("transactions").select("\*").eq("id", transactionId).single().execute()**

**transaction = transaction\_data["data"]**

**if not transaction:**

**raise HTTPException(status\_code=404, detail="Transaction not found")**

**# 確保是賣出交易**

**if transaction["action"] != "sell":**

**raise HTTPException(status\_code=400, detail="Analysis only available for sell transactions")**

**# 獲取該股票的購買記錄**

**buy\_transactions = supabase.table("transactions").select("\*").eq("userId", userId).eq("symbol", transaction["symbol"]).eq("action", "buy").order("timestamp", asc=True).execute()**

**if not buy\_transactions["data"]:**

**raise HTTPException(status\_code=404, detail="No buy transactions found for this stock")**

**# 計算平均買入價**

**total\_quantity = 0**

**total\_cost = 0**

**for buy in buy\_transactions["data"]:**

**total\_quantity += buy["quantity"]**

**total\_cost += buy["price"] \* buy["quantity"]**

**avg\_buy\_price = total\_cost / total\_quantity if total\_quantity > 0 else 0**

**# 計算持有天數**

**first\_buy\_date = datetime.fromisoformat(buy\_transactions["data"][0]["timestamp"].replace("Z", "+00:00"))**

**sell\_date = datetime.fromisoformat(transaction["timestamp"].replace("Z", "+00:00"))**

**holding\_days = (sell\_date - first\_buy\_date).days**

**# 獲取市場趨勢數據**

**formatted\_symbol = format\_symbol(transaction["symbol"])**

**market\_data = yf.download(formatted\_symbol,**

**start=(first\_buy\_date - timedelta(days=5)).strftime('%Y-%m-%d'),**

**end=(sell\_date + timedelta(days=5)).strftime('%Y-%m-%d'))**

**# 確定市場趨勢**

**if market\_data.empty:**

**market\_trend = "無法獲取市場數據"**

**else:**

**start\_price = market\_data.iloc[0]["Close"]**

**end\_price = market\_data.iloc[-1]["Close"]**

**market\_change = (end\_price - start\_price) / start\_price \* 100**

**if market\_change > 5:**

**market\_trend = "強勢上漲"**

**elif market\_change > 0:**

**market\_trend = "緩步上漲"**

**elif market\_change > -5:**

**market\_trend = "緩步下跌"**

**else:**

**market\_trend = "強勢下跌"**

**# 計算獲利百分比**

**profit\_percent = ((transaction["price"] - avg\_buy\_price) / avg\_buy\_price) \* 100**

**# 生成分析結果**

**analysis = ""**

**suggestions = []**

**if profit\_percent > 0:**

**analysis = f"恭喜您！這次交易獲利 {profit\_percent:.2f}%。您在{holding\_days}天內以平均{avg\_buy\_price:.2f}元買入，{transaction['price']}元賣出。"**

**if profit\_percent < 10:**

**analysis += " 雖然獲利，但收益相對有限。"**

**suggestions.append("考慮設定更高的獲利目標。")**

**suggestions.append("研究技術指標，尋找更佳賣出時機。")**

**else:**

**analysis += " 這是一次優異的交易結果！"**

**suggestions.append("記錄您的成功策略，以便日後參考。")**

**suggestions.append("考慮將部分獲利用於分散投資。")**

**if market\_trend == "強勢上漲":**

**analysis += " 您順應了市場上漲趨勢，成功獲利。"**

**elif market\_trend == "緩步下跌" or market\_trend == "強勢下跌":**

**analysis += " 您在市場下跌趨勢中成功獲利，表現出色。"**

**suggestions.append("分析您的賣出時機，為何能在下跌市場中獲利？")**

**else:**

**analysis = f"這次交易虧損 {abs(profit\_percent):.2f}%。您在{holding\_days}天內以平均{avg\_buy\_price:.2f}元買入，{transaction['price']}元賣出。"**

**if abs(profit\_percent) > 20:**

**analysis += " 這是一次較大的虧損。"**

**suggestions.append("考慮設定止損點，避免單次交易損失過大。")**

**suggestions.append("回顧您的買入決策，評估是否有改進空間。")**

**else:**

**analysis += " 虧損幅度較為有限。"**

**suggestions.append("分析買入時機與賣出決策，找出可改進的地方。")**

**if market\_trend == "強勢下跌":**

**analysis += " 您的交易與市場整體下跌趨勢相符。"**

**suggestions.append("在強烈下跌趨勢中，考慮更早止損或轉為分批賣出策略。")**

**elif market\_trend == "強勢上漲" or market\_trend == "緩步上漲":**

**analysis += " 值得注意的是，市場整體呈上升趨勢，但您的交易未能獲利。"**

**suggestions.append("分析為何在上漲市場中未能獲利，檢視賣出時機。")**

**# 根據績效獲取課程推薦**

**course\_recommendations = []**

**if profit\_percent < 0:**

**# 虧損時推薦基礎投資課程**

**course\_recommendations = [**

**{**

**"id": str(uuid.uuid4()),**

**"title": "股票投資基礎：從零開始",**

**"description": "適合初學者的股票投資入門課程，教您如何分析基本面與技術面。",**

**"price": 1999,**

**"url": "https://investacademy.com/basic-course",**

**"category": "基礎課程",**

**"relevanceScore": 0.95**

**},**

**{**

**"id": str(uuid.uuid4()),**

**"title": "投資心理學：克服情緒交易",**

**"description": "學習如何控制恐懼與貪婪情緒，做出更理性的投資決策。",**

**"price": 2499,**

**"url": "https://investacademy.com/psychology",**

**"category": "投資心理",**

**"relevanceScore": 0.88**

**}**

**]**

**else:**

**# 獲利時推薦進階課程**

**course\_recommendations = [**

**{**

**"id": str(uuid.uuid4()),**

**"title": "進階技術分析：趨勢研判與時機抓取",**

**"description": "深入學習如何運用技術指標判斷市場趨勢，掌握最佳買賣時機。",**

**"price": 3999,**

**"url": "https://investacademy.com/advanced-ta",**

**"category": "技術分析",**

**"relevanceScore": 0.92**

**},**

**{**

**"id": str(uuid.uuid4()),**

**"title": "投資組合最佳化策略",**

**"description": "學習如何建立多元化投資組合，平衡風險與報酬。",**

**"price": 2999,**

**"url": "https://investacademy.com/portfolio",**

**"category": "投資策略",**

**"relevanceScore": 0.85**

**}**

**]**

**return {**

**"transactionId": transactionId,**

**"buyPrice": avg\_buy\_price,**

**"sellPrice": transaction["price"],**

**"holdingPeriod": holding\_days,**

**"profitLoss": transaction["profit"],**

**"profitPercent": profit\_percent,**

**"analysis": analysis,**

**"suggestions": suggestions,**

**"marketTrend": market\_trend,**

**"relatedCourses": course\_recommendations**

**}**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))**

**@app.get("/ai/courses")**

**async def get\_course\_recommendations(userId: str, profitLoss: float):**

**try:**

**# 根據盈虧情況推薦不同課程**

**recommendations = []**

**if profitLoss < 0:**

**# 虧損情況推薦基礎與風險管理課程**

**recommendations = [**

**{**

**"id": str(uuid.uuid4()),**

**"title": "投資新手必修：風險管理入門",**

**"description": "學習如何在投資中保護資金，設定止損位，降低風險。",**

**"price": 1999,**

**"url": "https://investacademy.com/risk-management",**

**"category": "風險管理",**

**"relevanceScore": 0.96**

**},**

**{**

**"id": str(uuid.uuid4()),**

**"title": "技術分析基礎：掌握關鍵支撐與阻力",**

**"description": "了解如何識別價格趨勢與關鍵價位，做出更明智的買賣決定。",**

**"price": 2499,**

**"url": "https://investacademy.com/technical-basics",**

**"category": "技術分析",**

**"relevanceScore": 0.89**

**},**

**{**

**"id": str(uuid.uuid4()),**

**"title": "投資心理學：克服恐懼與貪婪",**

**"description": "探索交易心理學，學習如何控制情緒，避免衝動交易。",**

**"price": 1799,**

**"url": "https://investacademy.com/trading-psychology",**

**"category": "投資心理",**

**"relevanceScore": 0.93**

**}**

**]**

**elif profitLoss > 0 and profitLoss < 10000:**

**# 小額獲利推薦進階策略課程**

**recommendations = [**

**{**

**"id": str(uuid.uuid4()),**

**"title": "突破交易：識別關鍵價位與量能變化",**

**"description": "學習如何判斷價格突破有效性，把握大行情開始前的機會。",**

**"price": 2999,**

**"url": "https://investacademy.com/breakout-strategies",**

**"category": "交易策略",**

**"relevanceScore": 0.91**

**},**

**{**

**"id": str(uuid.uuid4()),**

**"title": "利潤最大化：優化交易規模與進出場時機",**

**"description": "學習如何調整持股比例與持有時間，最大化獲利。",**

**"price": 3499,**

**"url": "https://investacademy.com/profit-optimization",**

**"category": "交易策略",**

**"relevanceScore": 0.88**

**}**

**]**

**else:**

**# 大額獲利推薦高階課程**

**recommendations = [**

**{**

**"id": str(uuid.uuid4()),**

**"title": "專業交易者的心法：從賺錢到穩定獲利",**

**"description": "針對已有獲利經驗的投資者，提供更系統化的交易方法與心態建設。",**

**"price": 4999,**

**"url": "https://investacademy.com/pro-trading",**

**"category": "高階交易",**

**"relevanceScore": 0.95**

**},**

**{**

**"id": str(uuid.uuid4()),**

**"title": "投資組合管理與風險分散",**

**"description": "學習如何建立多元資產配置，平衡風險與報酬，實現長期穩定增長。",**

**"price": 3999,**

**"url": "https://investacademy.com/portfolio-management",**

**"category": "資產配置",**

**"relevanceScore": 0.92**

**}**

**]**

**return {"recommendations": recommendations}**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))**

**@app.get("/ai/portfolio")**

**async def get\_portfolio\_analysis(userId: str):**

**try:**

**# 獲取用戶持倉數據**

**position\_data = supabase.table("positions").select("\*").eq("userId", userId).execute()**

**positions = position\_data["data"]**

**if not positions:**

**raise HTTPException(status\_code=404, detail="No positions found for this user")**

**# 假設每個持倉都有對應的行業分類(實際應該從外部數據源獲取)**

**sectors = {**

**"2330": "半導體",**

**"2317": "電子零組件",**

**"2454": "半導體",**

**"2308": "電子零組件",**

**"2881": "金融業",**

**"2882": "金融業",**

**"2412": "電信服務",**

**"2303": "電腦及周邊",**

**"1301": "水泥工業",**

**"2002": "食品工業"**

**}**

**# 計算每個行業的持股比例**

**sector\_values = {}**

**total\_value = 0**

**for position in positions:**

**# 獲取股票價值(簡化版本，實際應該使用即時價格)**

**position\_value = position["quantity"] \* position["averageCost"]**

**total\_value += position\_value**

**# 獲取行業分類**

**symbol = position["symbol"]**

**sector = sectors.get(symbol, "其他")**

**if sector in sector\_values:**

**sector\_values[sector] += position\_value**

**else:**

**sector\_values[sector] = position\_value**

**# 計算行業配置百分比**

**sector\_allocation = []**

**for sector, value in sector\_values.items():**

**percentage = (value / total\_value) \* 100 if total\_value > 0 else 0**

**sector\_allocation.append({**

**"sector": sector,**

**"percentage": percentage**

**})**

**# 計算多樣化分數(簡單版本)**

**diversification\_score = min(len(sector\_values) \* 10, 100) if sector\_values else 0**

**# 計算風險分數(簡單版本 - 基於行業集中度)**

**max\_sector\_percentage = max([item["percentage"] for item in sector\_allocation]) if sector\_allocation else 0**

**risk\_score = max\_sector\_percentage / 10 # 0-10分，越高風險越大**

**# 生成投資組合建議**

**suggestions = []**

**if len(positions) < 3:**

**suggestions.append("增加持股數量，分散單一股票風險。")**

**if max\_sector\_percentage > 50:**

**suggestions.append(f"您在{max([item for item in sector\_allocation if item['percentage'] == max\_sector\_percentage])[0]['sector']}行業的配置過於集中，考慮分散投資到其他行業。")**

**if diversification\_score < 30:**

**suggestions.append("增加不同行業的配置，提高投資組合多樣性。")**

**# 計算預期報酬率(簡化版本)**

**expected\_return = 7.0 - (risk\_score / 10) \* 2 + (diversification\_score / 100) \* 3**

**return {**

**"diversificationScore": diversification\_score,**

**"riskScore": risk\_score,**

**"sectorAllocation": sector\_allocation,**

**"suggestions": suggestions,**

**"expectedReturn": expected\_return**

**}**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))  
  
 private func loadUserData() {**

**// 這裡應該連接到您的 API 服務**

**// 簡化示例代碼**

**let userId = UserDefaults.standard.string(forKey: "userId") ?? ""**

**// 模擬 API 呼叫**

**DispatchQueue.main.asyncAfter(deadline: .now() + 0.5) {**

**// 模擬數據 - 實際應從API獲取**

**self.initialCapital = 1000000**

**self.availableCash = 250000**

**self.assetsTrend = .up**

**// 更新總資產和報酬率 - 總資產應該是現金加上持股市值**

**self.calculateTotalAssets()**

**}**

**}**

**private func loadPositions() {**

**let userId = UserDefaults.standard.string(forKey: "userId") ?? ""**

**// 實際應用中應該呼叫 TradingService**

**TradingService.shared.getUserPositions(userId: userId)**

**.receive(on: DispatchQueue.main)**

**.sink(receiveCompletion: { completion in**

**if case .failure(let error) = completion {**

**print("Error loading positions: \(error)")**

**}**

**}, receiveValue: { response in**

**self.positions = response**

**self.calculateTotalAssets()**

**})**

**.store(in: &cancellables)**

**}**

**private func loadTransactions() {**

**let userId = UserDefaults.standard.string(forKey: "userId") ?? ""**

**TradingService.shared.getTransactionHistory(userId: userId)**

**.receive(on: DispatchQueue.main)**

**.sink(receiveCompletion: { completion in**

**if case .failure(let error) = completion {**

**print("Error loading transactions: \(error)")**

**}**

**}, receiveValue: { response in**

**self.transactions = response**

**})**

**.store(in: &cancellables)**

**}**

**private func loadPerformance() {**

**let userId = UserDefaults.standard.string(forKey: "userId") ?? ""**

**TradingService.shared.getPerformance(userId: userId, period: "all")**

**.receive(on: DispatchQueue.main)**

**.sink(receiveCompletion: { completion in**

**if case .failure(let error) = completion {**

**print("Error loading performance: \(error)")**

**}**

**}, receiveValue: { data in**

**self.performance = data**

**self.returnRate = data.returnRate**

**})**

**.store(in: &cancellables)**

**}**

**private func calculateTotalAssets() {**

**// 計算持股市值**

**var portfolioValue: Double = 0**

**for position in positions {**

**// 使用平均成本作為市值計算 (實際應該使用即時價格)**

**let positionValue = Double(position.quantity) \* position.averageCost**

**portfolioValue += positionValue**

**}**

**// 總資產 = 現金 + 持股市值**

**self.totalAssets = availableCash + portfolioValue**

**// 計算報酬率**

**if initialCapital > 0 {**

**self.returnRate = ((totalAssets - initialCapital) / initialCapital) \* 100**

**}**

**}**

**func formatCurrency(\_ value: Double) -> String {**

**let formatter = NumberFormatter()**

**formatter.numberStyle = .currency**

**formatter.locale = Locale(identifier: "zh\_TW")**

**return formatter.string(from: NSNumber(value: value)) ?? "NT$0"**

**}**

**// 用於管理 Combine 訂閱**

**private var cancellables = Set<AnyCancellable>()**

**}  
  
import SwiftUI**

**struct PortfolioView: View {**

**let positions: [Position]**

**var body: some View {**

**VStack {**

**if positions.isEmpty {**

**VStack(spacing: 16) {**

**Image(systemName: "chart.pie")**

**.font(.system(size: 50))**

**.foregroundColor(.gray)**

**Text("尚無持股")**

**.font(.headline)**

**.foregroundColor(.gray)**

**Text("點擊右上角的「+」開始購買股票")**

**.font(.subheadline)**

**.foregroundColor(.gray)**

**}**

**.frame(maxHeight: .infinity)**

**} else {**

**// 持股配置圓餅圖**

**PortfolioPieChartView(positions: positions)**

**.frame(height: 200)**

**.padding()**

**// 持股清單**

**List {**

**ForEach(positions) { position in**

**PositionRow(position: position)**

**}**

**}**

**.listStyle(PlainListStyle())**

**}**

**}**

**}**

**}**

**struct PositionRow: View {**

**let position: Position**

**@State private var currentPrice: Double?**

**@State private var isLoading = false**

**var body: some View {**

**HStack {**

**VStack(alignment: .leading) {**

**Text(position.symbol)**

**.font(.headline)**

**Text("\(position.quantity) 股 - \(marketTypeString(position.marketType))")**

**.font(.subheadline)**

**.foregroundColor(.gray)**

**}**

**Spacer()**

**VStack(alignment: .trailing) {**

**Text(formatCurrency(position.averageCost))**

**.font(.subheadline)**

**if let currentPrice = currentPrice {**

**HStack(spacing: 4) {**

**let change = ((currentPrice - position.averageCost) / position.averageCost) \* 100**

**Text(String(format: "%.2f%%", change))**

**.foregroundColor(change >= 0 ? .green : .red)**

**Image(systemName: change >= 0 ? "arrow.up" : "arrow.down")**

**.foregroundColor(change >= 0 ? .green : .red)**

**.font(.footnote)**

**}**

**} else if isLoading {**

**ProgressView()**

**.scaleEffect(0.7)**

**} else {**

**Text("載入中...")**

**.font(.caption)**

**.foregroundColor(.gray)**

**}**

**}**

**}**

**.padding(.vertical, 8)**

**.onAppear {**

**loadCurrentPrice()**

**}**

**}**

**private func loadCurrentPrice() {**

**isLoading = true**

**// 應該從 StockDataService 獲取即時價格**

**// 簡化示例**

**DispatchQueue.main.asyncAfter(deadline: .now() + 0.5) {**

**// 模擬價格波動**

**let randomVariation = Double.random(in: -5...5) / 100**

**self.currentPrice = position.averageCost \* (1 + randomVariation)**

**self.isLoading = false**

**}**

**}**

**private func marketTypeString(\_ type: String) -> String {**

**switch type {**

**case "上市":**

**return "上市"**

**case "上櫃":**

**return "上櫃"**

**case "興櫃":**

**return "興櫃"**

**default:**

**return type**

**}**

**}**

**private func formatCurrency(\_ value: Double) -> String {**

**let formatter = NumberFormatter()**

**formatter.numberStyle = .currency**

**formatter.locale = Locale(identifier: "zh\_TW")**

**return formatter.string(from: NSNumber(value: value)) ?? "NT$0"**

**}**

**}**

**struct PortfolioPieChartView: View {**

**let positions: [Position]**

**// 計算總持股價值**

**private var totalValue: Double {**

**positions.reduce(0) { $0 + Double($1.quantity) \* $1.averageCost }**

**}**

**// 產生持股配置顏色**

**private let colors: [Color] = [**

**.blue, .green, .orange, .red, .purple, .yellow, .pink,**

**.cyan, .indigo, .mint, .teal, .brown**

**]**

**var body: some View {**

**VStack {**

**Text("持股配置")**

**.font(.headline)**

**.padding(.bottom, 4)**

**// 使用 SwiftUI Charts 繪製圓餅圖**

**// 注意：在iOS 16+可以使用原生Charts框架**

**// 這裡使用簡化版本**

**ZStack {**

**ForEach(0..<positions.count, id: \.self) { index in**

**let position = positions[index]**

**let value = Double(position.quantity) \* position.averageCost**

**let percentage = value / totalValue**

**PieSliceView(**

**startAngle: self.startAngle(for: index),**

**endAngle: self.endAngle(for: index),**

**color: colors[index % colors.count]**

**)**

**}**

**Circle()**

**.fill(Color(.systemBackground))**

**.frame(width: 100)**

**VStack {**

**Text("\(positions.count)")**

**.font(.system(size: 24, weight: .bold))**

**Text("持股")**

**.font(.caption)**

**.foregroundColor(.gray)**

**}**

**}**

**// 顯示持股比例圖例**

**ScrollView(.horizontal, showsIndicators: false) {**

**HStack(spacing: 16) {**

**ForEach(0..<positions.count, id: \.self) { index in**

**let position = positions[index]**

**let value = Double(position.quantity) \* position.averageCost**

**let percentage = value / totalValue**

**HStack(spacing: 4) {**

**Circle()**

**.fill(colors[index % colors.count])**

**.frame(width: 10, height: 10)**

**Text(position.symbol)**

**.font(.caption)**

**Text(String(format: "%.1f%%", percentage \* 100))**

**.font(.caption)**

**.fontWeight(.semibold)**

**}**

**}**

**}**

**.padding(.horizontal)**

**}**

**}**

**}**

**// 計算每個持股在圓餅圖中的起始角度**

**private func startAngle(for index: Int) -> Double {**

**if index == 0 { return 0 }**

**var sum: Double = 0**

**for i in 0..<index {**

**let position = positions[i]**

**let value = Double(position.quantity) \* position.averageCost**

**sum += value / totalValue**

**}**

**return sum \* 360**

**}**

**// 計算每個持股在圓餅圖中的結束角度**

**private func endAngle(for index: Int) -> Double {**

**let position = positions[index]**

**let value = Double(position.quantity) \* position.averageCost**

**let percentage = value / totalValue**

**return startAngle(for: index) + percentage \* 360**

**}**

**}**

**struct PieSliceView: View {**

**let startAngle: Double**

**let endAngle: Double**

**let color: Color**

**var body: some View {**

**GeometryReader { geometry in**

**Path { path in**

**let center = CGPoint(x: geometry.size.width / 2, y: geometry.size.height / 2)**

**let radius = min(geometry.size.width, geometry.size.height) / 2**

**path.move(to: center)**

**path.addArc(**

**center: center,**

**radius: radius,**

**startAngle: .degrees(startAngle),**

**endAngle: .degrees(endAngle),**

**clockwise: false**

**)**

**path.closeSubpath()**

**}**

**.fill(color)**

**}**

**}**

**}  
  
import SwiftUI**

**struct TransactionHistoryView: View {**

**let transactions: [Transaction]**

**var body: some View {**

**VStack {**

**if transactions.isEmpty {**

**VStack(spacing: 16) {**

**Image(systemName: "doc.plaintext")**

**.font(.system(size: 50))**

**.foregroundColor(.gray)**

**Text("尚無交易紀錄")**

**.font(.headline)**

**.foregroundColor(.gray)**

**Text("完成第一筆交易後，紀錄將顯示在這裡")**

**.font(.subheadline)**

**.foregroundColor(.gray)**

**}**

**.frame(maxHeight: .infinity)**

**} else {**

**List {**

**ForEach(transactions) { transaction in**

**NavigationLink(destination: TransactionDetailView(transaction: transaction)) {**

**TransactionRow(transaction: transaction)**

**}**

**}**

**}**

**.listStyle(PlainListStyle())**

**}**

**}**

**}**

**}**

**struct TransactionRow: View {**

**let transaction: Transaction**

**var body: some View {**

**HStack {**

**// 買入/賣出圖標**

**Image(systemName: transaction.action == "buy" ? "arrow.down" : "arrow.up")**

**.foregroundColor(transaction.action == "buy" ? .blue : .red)**

**.font(.system(size: 16, weight: .semibold))**

**.frame(width: 32, height: 32)**

**.background(**

**Circle()**

**.fill(transaction.action == "buy" ? Color.blue.opacity(0.15) : Color.red.opacity(0.15))**

**)**

**VStack(alignment: .leading) {**

**Text(transaction.symbol)**

**.font(.headline)**

**Text("\(transaction.action == "buy" ? "買入" : "賣出") \(transaction.quantity) 股")**

**.font(.subheadline)**

**.foregroundColor(.gray)**

**}**

**Spacer()**

**VStack(alignment: .trailing) {**

**Text(formatCurrency(transaction.price))**

**.font(.subheadline)**

**// 顯示交易日期**

**Text(formatDate(transaction.timestamp))**

**.font(.caption)**

**.foregroundColor(.gray)**

**}**

**}**

**.padding(.vertical, 8)**

**}**

**private func formatCurrency(\_ value: Double) -> String {**

**let formatter = NumberFormatter()**

**formatter.numberStyle = .currency**

**formatter.locale = Locale(identifier: "zh\_TW")**

**return formatter.string(from: NSNumber(value: value)) ?? "NT$0"**

**}**

**private func formatDate(\_ date: Date) -> String {**

**let formatter = DateFormatter()**

**formatter.dateStyle = .short**

**formatter.timeStyle = .short**

**formatter.locale = Locale(identifier: "zh\_TW")**

**return formatter.string(from: date)**

**}**

**}**

**struct TransactionDetailView: View {**

**let transaction: Transaction**

**@State private var analysisData: TradingAnalysis?**

**@State private var isLoading = false**

**var body: some View {**

**ScrollView {**

**VStack(alignment: .leading, spacing: 16) {**

**// 交易基本信息**

**VStack(spacing: 20) {**

**HStack {**

**Text(transaction.symbol)**

**.font(.system(size: 24, weight: .bold))**

**Spacer()**

**Text(transaction.action == "buy" ? "買入" : "賣出")**

**.font(.headline)**

**.foregroundColor(.white)**

**.padding(.horizontal, 12)**

**.padding(.vertical, 6)**

**.background(transaction.action == "buy" ? Color.blue : Color.red)**

**.cornerRadius(6)**

**}**

**Divider()**

**HStack(spacing: 40) {**

**VStack(alignment: .leading) {**

**Text("價格")**

**.font(.subheadline)**

**.foregroundColor(.gray)**

**Text(formatCurrency(transaction.price))**

**.font(.headline)**

**}**

**VStack(alignment: .leading) {**

**Text("數量")**

**.font(.subheadline)**

**.foregroundColor(.gray)**

**Text("\(transaction.quantity) 股")**

**.font(.headline)**

**}**

**VStack(alignment: .leading) {**

**Text("手續費")**

**.font(.subheadline)**

**.foregroundColor(.gray)**

**Text(formatCurrency(transaction.fee))**

**.font(.headline)**

**}**

**}**

**Divider()**

**HStack {**

**Text("總金額")**

**.font(.subheadline)**

**.foregroundColor(.gray)**

**Spacer()**

**Text(formatCurrency(transaction.totalAmount))**

**.font(.title2)**

**.fontWeight(.bold)**

**}**

**if transaction.action == "sell", let profit = transaction.profit {**

**HStack {**

**Text("獲利/虧損")**

**.font(.subheadline)**

**.foregroundColor(.gray)**

**Spacer()**

**Text(formatCurrency(profit))**

**.font(.title3)**

**.fontWeight(.semibold)**

**.foregroundColor(profit >= 0 ? .green : .red)**

**}**

**}**

**Text("交易時間: \(formatDateTime(transaction.timestamp))")**

**.font(.footnote)**

**.foregroundColor(.gray)**

**}**

**.padding()**

**.background(Color(.systemBackground))**

**.cornerRadius(12)**

**.shadow(color: Color.black.opacity(0.05), radius: 5, x: 0, y: 2)**

**// 賣出時顯示交易分析**

**if transaction.action == "sell" {**

**if isLoading {**

**HStack {**

**Spacer()**

**ProgressView("分析中...")**

**Spacer()**

**}**

**.padding()**

**} else if let analysis = analysisData {**

**VStack(alignment: .leading, spacing: 16) {**

**Text("AI 交易分析")**

**.font(.headline)**

**.padding(.top, 4)**

**// 交易分析結果**

**VStack(alignment: .leading, spacing: 12) {**

**Text(analysis.analysis)**

**.font(.body)**

**.lineSpacing(4)**

**Divider()**

**Text("持有時間: \(analysis.holdingPeriod) 天")**

**.font(.subheadline)**

**Text("市場趨勢: \(analysis.marketTrend)")**

**.font(.subheadline)**

**Divider()**

**Text("建議")**

**.font(.headline)**

**.padding(.top, 4)**

**ForEach(analysis.suggestions, id: \.self) { suggestion in**

**HStack(alignment: .top, spacing: 8) {**

**Image(systemName: "checkmark.circle.fill")**

**.foregroundColor(.blue)**

**.font(.footnote)**

**Text(suggestion)**

**.font(.body)**

**.lineSpacing(4)**

**}**

**}**

**}**

**.padding()**

**.background(Color(.systemBackground))**

**.cornerRadius(12)**

**.shadow(color: Color.black.opacity(0.05), radius: 5, x: 0, y: 2)**

**// 相關課程推薦**

**if let courses = analysis.relatedCourses, !courses.isEmpty {**

**Text("課程推薦")**

**.font(.headline)**

**.padding(.top, 8)**

**ForEach(courses) { course in**

**CourseCardView(course: course)**

**}**

**}**

**}**

**.padding()**

**} else {**

**Button(action: {**

**loadAnalysis()**

**}) {**

**HStack {**

**Spacer()**

**Text("獲取 AI 交易分析")**

**.fontWeight(.semibold)**

**Spacer()**

**}**

**.padding()**

**.background(Color.blue)**

**.foregroundColor(.white)**

**.cornerRadius(8)**

**.padding()**

**}**

**}**

**}**

**}**

**.padding()**

**}**

**.navigationTitle("交易詳情")**

**.navigationBarTitleDisplayMode(.inline)**

**.onAppear {**

**if transaction.action == "sell" {**

**loadAnalysis()**

**}**

**}**

**}**

**private func loadAnalysis() {**

**isLoading = true**

**// 實際應該呼叫 AIAnalysisService**

**let userId = UserDefaults.standard.string(forKey: "userId") ?? ""**

**// 模擬 API 呼叫**

**DispatchQueue.main.asyncAfter(deadline: .now() + 1.5) {**

**// 模擬數據 - 實際應從 API 獲取**

**self.analysisData = TradingAnalysis(**

**transactionId: self.transaction.id,**

**buyPrice: self.transaction.price \* 0.9, // 假設買入價格比賣出低 10%**

**sellPrice: self.transaction.price,**

**holdingPeriod: 14,**

**profitLoss: self.transaction.profit ?? 0,**

**profitPercent: 10.5, // 假設獲利 10.5%**

**analysis: "本次交易您獲利10.5%，持有期間為14天。這筆交易展現了良好的時機掌握，賣出價格高於您的買入成本，成功獲利。在這段期間，大盤處於緩步上漲趨勢，您的交易表現優於大盤平均水平。",**

**suggestions: [**

**"繼續研究此類型成功交易模式，尋找相似的買點特徵。",**

**"考慮設定更明確的獲利目標，以便在未來交易中能更有系統地獲利了結。",**

**"分析此交易成功的關鍵技術指標，並在未來交易中重點關注這些指標。"**

**],**

**marketTrend: "緩步上漲",**

**relatedCourses: [**

**CourseRecommendation(**

**id: "course1",**

**title: "技術分析進階：突破型態與交易時機",**

**description: "學習如何識別股價突破形態，掌握關鍵買賣時機，提高交易勝率。",**

**price: 2999,**

**url: URL(string: "https://investacademy.com/technical-advanced")!,**

**category: "技術分析",**

**relevanceScore: 0.92**

**),**

**CourseRecommendation(**

**id: "course2",**

**title: "交易心理學：克服情緒，提高績效",**

**description: "學習如何控制交易情緒，避免常見心理陷阱，培養紀律交易習慣。",**

**price: 1999,**

**url: URL(string: "https://investacademy.com/trading-psychology")!,**

**category: "交易心理",**

**relevanceScore: 0.85**

**)**

**]**

**)**

**self.isLoading = false**

**}**

**}**

**private func formatCurrency(\_ value: Double) -> String {**

**let formatter = NumberFormatter()**

**formatter.numberStyle = .currency**

**formatter.locale = Locale(identifier: "zh\_TW")**

**return formatter.string(from: NSNumber(value: value)) ?? "NT$0"**

**}**

**private func formatDateTime(\_ date: Date) -> String {**

**let formatter = DateFormatter()**

**formatter.dateStyle = .medium**

**formatter.timeStyle = .medium**

**formatter.locale = Locale(identifier: "zh\_TW")**

**return formatter.string(from: date)**

**}**

**}**

**struct CourseCardView: View {**

**let course: CourseRecommendation**

**var body: some View {**

**VStack(alignment: .leading, spacing: 12) {**

**HStack {**

**Text(course.category)**

**.font(.caption)**

**.fontWeight(.semibold)**

**.foregroundColor(.white)**

**.padding(.horizontal, 8)**

**.padding(.vertical, 4)**

**.background(Color.blue)**

**.cornerRadius(4)**

**Spacer()**

**Text("相關度 \(Int(course.relevanceScore \* 100))%")**

**.font(.caption)**

**.foregroundColor(.gray)**

**}**

**Text(course.title)**

**.font(.headline)**

**.lineLimit(2)**

**Text(course.description)**

**.font(.body)**

**.foregroundColor(.gray)**

**.lineLimit(3)**

**HStack {**

**Text(formatCurrency(course.price))**

**.font(.headline)**

**.foregroundColor(.blue)**

**Spacer()**

**Link(destination: course.url) {**

**Text("了解更多")**

**.font(.subheadline)**

**.fontWeight(.semibold)**

**.foregroundColor(.white)**

**.padding(.horizontal, 16)**

**.padding(.vertical, 8)**

**.background(Color.blue)**

**.cornerRadius(6)**

**}**

**}**

**}**

**.padding()**

**.background(Color(.systemBackground))**

**.cornerRadius(12)**

**.shadow(color: Color.black.opacity(0.05), radius: 5, x: 0, y: 2)**

**}**

**private func formatCurrency(\_ value: Double) -> String {**

**let formatter = NumberFormatter()**

**formatter.numberStyle = .currency**

**formatter.locale = Locale(identifier: "zh\_TW")**

**return formatter.string(from: NSNumber(value: value)) ?? "NT$0"**

**}**

**}  
  
import SwiftUI**

**struct PerformanceRankingView: View {**

**let performance: PerformanceData?**

**@State private var selectedPeriod = "week"**

**var body: some View {**

**VStack {**

**// 排行類型選擇器**

**Picker("排行時間", selection: $selectedPeriod) {**

**Text("週排行").tag("week")**

**Text("月排行").tag("month")**

**Text("年排行").tag("year")**

**}**

**.pickerStyle(SegmentedPickerStyle())**

**.padding()**

**if let performanceData = performance {**

**VStack(spacing: 16) {**

**// 用戶排名信息**

**VStack {**

**Text("您的排名")**

**.font(.headline)**

**.padding(.bottom, 4)**

**HStack(spacing: 20) {**

**RankingStatView(**

**title: "報酬率",**

**value: String(format: "%.2f%%", performanceData.returnRate),**

**valueColor: performanceData.returnRate >= 0 ? .green : .red**

**)**

**RankingStatView(**

**title: "排名",**

**value: "\(performanceData.ranking) / \(performanceData.totalUsers)",**

**valueColor: .primary**

**)**

**}**

**}**

**.padding()**

**.background(Color(.systemBackground))**

**.cornerRadius(12)**

**.shadow(color: Color.black.opacity(0.05), radius: 5, x: 0, y: 2)**

**// 績效變化**

**VStack(alignment: .leading, spacing: 12) {**

**Text("績效變化")**

**.font(.headline)**

**.padding(.bottom, 4)**

**PerformanceChangeRow(title: "週變動", value: performanceData.weeklyChange)**

**Divider()**

**PerformanceChangeRow(title: "月變動", value: performanceData.monthlyChange)**

**Divider()**

**PerformanceChangeRow(title: "年變動", value: performanceData.yearlyChange)**

**}**

**.padding()**

**.background(Color(.systemBackground))**

**.cornerRadius(12)**

**.shadow(color: Color.black.opacity(0.05), radius: 5, x: 0, y: 2)**

**// 排行榜 (假設)**

**VStack(alignment: .leading, spacing: 12) {**

**Text("TOP 投資者")**

**.font(.headline)**

**.padding(.bottom, 4)**

**ForEach(1...5, id: \.self) { index in**

**HStack {**

**Text("\(index)")**

**.font(.headline)**

**.frame(width: 30, height: 30)**

**.background(**

**Circle()**

**.fill(rankColor(for: index))**

**)**

**.foregroundColor(.white)**

**Text("用戶 \(100000 + index)")**

**.font(.subheadline)**

**Spacer()**

**Text(String(format: "+%.2f%%", 30.0 - Double(index) \* 5))**

**.font(.headline)**

**.foregroundColor(.green)**

**}**

**.padding(.vertical, 4)**

**if index < 5 {**

**Divider()**

**}**

**}**

**}**

**.padding()**

**.background(Color(.systemBackground))**

**.cornerRadius(12)**

**.shadow(color: Color.black.opacity(0.05), radius: 5, x: 0, y: 2)**

**}**

**.padding()**

**} else {**

**VStack(spacing: 16) {**

**ProgressView()**

**Text("載入排行榜中...")**

**.foregroundColor(.gray)**

**}**

**.frame(maxHeight: .infinity)**

**}**

**}**

**}**

**private func rankColor(for rank: Int) -> Color {**

**switch rank {**

**case 1:**

**return Color.yellow**

**case 2:**

**return Color.gray**

**case 3:**

**return Color.orange**

**default:**

**return Color.blue**

**}**

**}**

**}**

**struct RankingStatView: View {**

**let title: String**

**let value: String**

**let valueColor: Color**

**var body: some View {**

**VStack(spacing: 4) {**

**Text(title)**

**.font(.subheadline)**

**.foregroundColor(.gray)**

**Text(value)**

**.font(.title2)**

**.fontWeight(.bold)**

**.foregroundColor(valueColor)**

**}**

**.frame(maxWidth: .infinity)**

**}**

**}**

**struct PerformanceChangeRow: View {**

**let title: String**

**let value: Double**

**var body: some View {**

**HStack {**

**Text(title)**

**.font(.subheadline)**

**Spacer()**

**HStack(spacing: 4) {**

**Image(systemName: value >= 0 ? "arrow.up.right" : "arrow.down.right")**

**.foregroundColor(value >= 0 ? .green : .red)**

**.font(.caption)**

**Text(String(format: "%.2f%%", abs(value)))**

**.font(.subheadline)**

**.fontWeight(.semibold)**

**.foregroundColor(value >= 0 ? .green : .red)**

**}**

**}**

**}**

**}  
  
import SwiftUI**

**struct SearchStockView: View {**

**@State private var searchQuery = ""**

**@State private var searchResults: [StockInfo] = []**

**@State private var isSearching = false**

**@Environment(\.presentationMode) var presentationMode**

**var body: some View {**

**VStack {**

**// 搜索欄**

**HStack {**

**Image(systemName: "magnifyingglass")**

**.foregroundColor(.gray)**

**TextField("輸入股票代碼或名稱", text: $searchQuery)**

**.autocapitalization(.none)**

**.disableAutocorrection(true)**

**if !searchQuery.isEmpty {**

**Button(action: {**

**searchQuery = ""**

**}) {**

**Image(systemName: "xmark.circle.fill")**

**.foregroundColor(.gray)**

**}**

**}**

**}**

**.padding(.vertical, 10)**

**.padding(.horizontal, 12)**

**.background(Color(.systemGray6))**

**.cornerRadius(10)**

**.padding(.horizontal)**

**.padding(.top, 8)**

**.onChange(of: searchQuery) { query in**

**if query.isEmpty {**

**searchResults = []**

**isSearching = false**

**} else {**

**performSearch()**

**}**

**}**

**if isSearching {**

**// 搜尋中**

**ProgressView("搜尋中...")**

**.padding()**

**Spacer()**

**} else if searchResults.isEmpty && !searchQuery.isEmpty {**

**// 無結果**

**VStack(spacing: 16) {**

**Image(systemName: "magnifyingglass")**

**.font(.system(size: 40))**

**.foregroundColor(.gray)**

**Text("找不到符合的股票")**

**.font(.headline)**

**.foregroundColor(.gray)**

**}**

**.frame(maxHeight: .infinity)**

**} else if searchResults.isEmpty {**

**// 初始狀態**

**VStack(spacing: 16) {**

**Image(systemName: "text.magnifyingglass")**

**.font(.system(size: 40))**

**.foregroundColor(.gray)**

**Text("搜尋股票以開始交易")**

**.font(.headline)**

**.foregroundColor(.gray)**

**Text("輸入股票代碼或名稱")**

**.font(.subheadline)**

**.foregroundColor(.gray)**

**}**

**.frame(maxHeight: .infinity)**

**} else {**

**// 搜尋結果**

**List {**

**ForEach(searchResults) { stock in**

**NavigationLink(destination: StockDetailView(stock: stock)) {**

**StockRow(stock: stock)**

**}**

**}**

**}**

**.listStyle(PlainListStyle())**

**}**

**}**

**.navigationTitle("搜尋股票")**

**.navigationBarTitleDisplayMode(.inline)**

**}**

**private func performSearch() {**

**isSearching = true**

**// 實際應呼叫 StockDataService**

**StockDataService.shared.searchStocks(query: searchQuery)**

**.receive(on: DispatchQueue.main)**

**.sink(receiveCompletion: { completion in**

**if case .failure(let error) = completion {**

**print("Search error: \(error)")**

**}**

**isSearching = false**

**}, receiveValue: { response in**

**self.searchResults = response**

**isSearching = false**

**})**

**.store(in: &cancellables)**

**}**

**// 用於管理 Combine 訂閱**

**private var cancellables = Set<AnyCancellable>()**

**}**

**struct StockRow: View {**

**let stock: StockInfo**

**var body: some View {**

**HStack {**

**VStack(alignment: .leading, spacing: 4) {**

**HStack {**

**Text(stock.symbol)**

**.font(.headline)**

**Text(stock.marketType)**

**.font(.caption)**

**.foregroundColor(.white)**

**.padding(.horizontal, 6)**

**.padding(.vertical, 2)**

**.background(marketTypeColor(stock.marketType))**

**.cornerRadius(4)**

**}**

**Text(stock.name)**

**.font(.subheadline)**

**.foregroundColor(.gray)**

**}**

**Spacer()**

**Image(systemName: "chevron.right")**

**.font(.caption)**

**.foregroundColor(.gray)**

**}**

**.padding(.vertical, 4)**

**}**

**private func marketTypeColor(\_ type: String) -> Color {**

**switch type {**

**case "上市":**

**return Color.blue**

**case "上櫃":**

**return Color.green**

**case "興櫃":**

**return Color.orange**

**default:**

**return Color.gray**

**}**

**}**

**}**

**struct StockDetailView: View {**

**let stock: StockInfo**

**@State private var currentPrice: Double?**

**@State private var historicalData: [HistoricalDataPoint] = []**

**@State private var isLoadingPrice = false**

**@State private var isLoadingHistory = false**

**@State private var quantity: String = "1"**

**@State private var showBuySheet = false**

**@State private var showSellSheet = false**

**var body: some View {**

**ScrollView {**

**VStack(alignment: .leading, spacing: 16) {**

**// 股票基本信息**

**VStack(spacing: 12) {**

**HStack {**

**Text(stock.name)**

**.font(.title)**

**.fontWeight(.bold)**

**Spacer()**

**Text(stock.marketType)**

**.font(.subheadline)**

**.foregroundColor(.white)**

**.padding(.horizontal, 10)**

**.padding(.vertical, 5)**

**.background(marketTypeColor(stock.marketType))**

**.cornerRadius(6)**

**}**

**if isLoadingPrice {**

**HStack {**

**Spacer()**

**ProgressView("載入價格中...")**

**Spacer()**

**}**

**} else if let price = currentPrice {**

**Text(formatCurrency(price))**

**.font(.system(size: 36, weight: .bold))**

**.padding(.vertical, 8)**

**}**

**HStack(spacing: 20) {**

**Button(action: {**

**showBuySheet = true**

**}) {**

**HStack {**

**Image(systemName: "arrow.down")**

**Text("買入")**

**.fontWeight(.semibold)**

**}**

**.frame(maxWidth: .infinity)**

**.padding()**

**.background(Color.blue)**

**.foregroundColor(.white)**

**.cornerRadius(10)**

**}**

**Button(action: {**

**showSellSheet = true**

**}) {**

**HStack {**

**Image(systemName: "arrow.up")**

**Text("賣出")**

**.fontWeight(.semibold)**

**}**

**.frame(maxWidth: .infinity)**

**.padding()**

**.background(Color.red)**

**.foregroundColor(.white)**

**.cornerRadius(10)**

**}**

**}**

**}**

**.padding()**

**.background(Color(.systemBackground))**

**.cornerRadius(12)**

**.shadow(color: Color.black.opacity(0.05), radius: 5, x: 0, y: 2)**

**// 歷史價格圖表**

**VStack(alignment: .leading, spacing: 12) {**

**Text("歷史價格")**

**.font(.headline)**

**.padding(.horizontal)**

**if isLoadingHistory {**

**HStack {**

**Spacer()**

**ProgressView("載入資料中...")**

**Spacer()**

**}**

**.padding()**

**} else if historicalData.isEmpty {**

**HStack {**

**Spacer()**

**Text("無歷史數據")**

**.foregroundColor(.gray)**

**Spacer()**

**}**

**.padding()**

**} else {**

**// 簡易圖表實現**

**// 真實應用中應使用專業圖表庫**

**GeometryReader { geometry in**

**StockPriceChartView(data: historicalData, size: geometry.size)**

**}**

**.frame(height: 200)**

**.padding(.horizontal)**

**}**

**}**

**.padding(.vertical)**

**.background(Color(.systemBackground))**

**.cornerRadius(12)**

**.shadow(color: Color.black.opacity(0.05), radius: 5, x: 0, y: 2)**

**}**

**.padding()**

**}**

**.navigationTitle(stock.symbol)**

**.navigationBarTitleDisplayMode(.inline)**

**.sheet(isPresented: $showBuySheet) {**

**TradeSheetView(**

**stock: stock,**

**currentPrice: currentPrice ?? 0,**

**isBuy: true,**

**quantity: $quantity**

**)**

**}**

**.sheet(isPresented: $showSellSheet) {**

**TradeSheetView(**

**stock: stock,**

**currentPrice: currentPrice ?? 0,**

**isBuy: false,**

**quantity: $quantity**

**)**

**}**

**.onAppear {**

**loadStockData()**

**}**

**}**

**private func loadStockData() {**

**loadCurrentPrice()**

**loadHistoricalData()**

**}**

**private func loadCurrentPrice() {**

**isLoadingPrice = true**

**// 應呼叫 StockDataService**

**StockDataService.shared.fetchCurrentPrice(symbol: stock.symbol)**

**.receive(on: DispatchQueue.main)**

**.sink(receiveCompletion: { completion in**

**if case .failure(let error) = completion {**

**print("Error loading price: \(error)")**

**}**

**isLoadingPrice = false**

**}, receiveValue: { price in**

**self.currentPrice = price**

**isLoadingPrice = false**

**})**

**.store(in: &cancellables)**

**}**

**private func loadHistoricalData() {**

**isLoadingHistory = true**

**// 應呼叫 StockDataService**

**StockDataService.shared.fetchHistoricalData(symbol: stock.symbol, period: "1mo", interval: "1d")**

**.receive(on: DispatchQueue.main)**

**.sink(receiveCompletion: { completion in**

**if case .failure(let error) = completion {**

**print("Error loading historical data: \(error)")**

**}**

**isLoadingHistory = false**

**}, receiveValue: { data in**

**self.historicalData = data**

**isLoadingHistory = false**

**})**

**.store(in: &cancellables)**

**}**

**private func formatCurrency(\_ value: Double) -> String {**

**let formatter = NumberFormatter()**

**formatter.numberStyle = .currency**

**formatter.locale = Locale(identifier: "zh\_TW")**

**return formatter.string(from: NSNumber(value: value)) ?? "NT$0"**

**}**

**private func marketTypeColor(\_ type: String) -> Color {**

**switch type {**

**case "上市":**

**return Color.blue**

**case "上櫃":**

**return Color.green**

**case "興櫃":**

**return Color.orange**

**default:**

**return Color.gray**

**}**

**}**

**// 用於管理 Combine 訂閱**

**private var cancellables = Set<AnyCancellable>()**

**}**

**struct StockPriceChartView: View {**

**let data: [HistoricalDataPoint]**

**let size: CGSize**

**private var minPrice: Double {**

**data.map { $0.low }.min() ?? 0**

**}**

**private var maxPrice: Double {**

**data.map { $0.high }.max() ?? 0**

**}**

**private var priceRange: Double {**

**maxPrice - minPrice**

**}**

**var body: some View {**

**ZStack(alignment: .bottomLeading) {**

**// Y軸參考線**

**VStack(spacing: 0) {**

**ForEach(0..<5) { i in**

**Spacer()**

**Rectangle()**

**.fill(Color(.systemGray5))**

**.frame(height: 1)**

**}**

**}**

**// 蠟燭圖表**

**HStack(spacing: 2) {**

**ForEach(0..<data.count, id: \.self) { index in**

**if index < data.count {**

**let point = data[index]**

**CandlestickView(**

**point: point,**

**minPrice: minPrice,**

**maxPrice: maxPrice,**

**availableHeight: size.height**

**)**

**}**

**}**

**}**

**// Y軸價格標籤**

**VStack(spacing: 0) {**

**ForEach(0..<5) { i in**

**let price = minPrice + priceRange \* (Double(4 - i) / 4)**

**Text(formatPrice(price))**

**.font(.caption2)**

**.foregroundColor(.gray)**

**Spacer()**

**}**

**}**

**.padding(.trailing, size.width - 40)**

**}**

**}**

**private func formatPrice(\_ price: Double) -> String {**

**let formatter = NumberFormatter()**

**formatter.numberStyle = .decimal**

**formatter.maximumFractionDigits = 2**

**return formatter.string(from: NSNumber(value: price)) ?? "0"**

**}**

**}**

**struct CandlestickView: View {**

**let point: HistoricalDataPoint**

**let minPrice: Double**

**let maxPrice: Double**

**let availableHeight: CGFloat**

**private var isUp: Bool {**

**point.close >= point.open**

**}**

**private func yPosition(for price: Double) -> CGFloat {**

**guard maxPrice > minPrice else { return 0 }**

**let percentage = 1 - ((price - minPrice) / (maxPrice - minPrice))**

**return percentage \* availableHeight**

**}**

**var body: some View {**

**VStack(spacing: 0) {**

**// 上影線**

**Rectangle()**

**.fill(isUp ? Color.green : Color.red)**

**.frame(width: 1, height: yPosition(for: point.high) - yPosition(for: max(point.open, point.close)))**

**// 實體**

**Rectangle()**

**.fill(isUp ? Color.green : Color.red)**

**.frame(**

**width: 8,**

**height: max(1, abs(yPosition(for: point.open) - yPosition(for: point.close)))**

**)**

**// 下影線**

**Rectangle()**

**.fill(isUp ? Color.green : Color.red)**

**.frame(width: 1, height: yPosition(for: min(point.open, point.close)) - yPosition(for: point.low))**

**}**

**}**

**}**

**struct TradeSheetView: View {**

**let stock: StockInfo**

**let currentPrice: Double**

**let isBuy: Bool**

**@Binding var quantity: String**

**@State private var isProcessing = false**

**@State private var showResult = false**

**@State private var transactionResult: Transaction?**

**@State private var errorMessage: String?**

**@Environment(\.presentationMode) var presentationMode**

**private var totalAmount: Double {**

**let qty = Int(quantity) ?? 0**

**return Double(qty) \* currentPrice**

**}**

**private var fee: Double {**

**totalAmount \* 0.001425 // 台股手續費率**

**}**

**private var finalAmount: Double {**

**totalAmount + fee**

**}**

**var body: some View {**

**NavigationView {**

**Form {**

**Section(header: Text("交易資訊")) {**

**HStack {**

**Text("股票代號")**

**Spacer()**

**Text(stock.symbol)**

**.foregroundColor(.gray)**

**}**

**HStack {**

**Text("股票名稱")**

**Spacer()**

**Text(stock.name)**

**.foregroundColor(.gray)**

**}**

**HStack {**

**Text("市場別")**

**Spacer()**

**Text(stock.marketType)**

**.foregroundColor(.gray)**

**}**

**HStack {**

**Text("現價")**

**Spacer()**

**Text(formatCurrency(currentPrice))**

**.foregroundColor(.gray)**

**}**

**}**

**Section(header: Text("交易數量")) {**

**HStack {**

**Text("數量 (股)")**

**Spacer()**

**TextField("輸入股數", text: $quantity)**

**.keyboardType(.numberPad)**

**.multilineTextAlignment(.trailing)**

**}**

**HStack {**

**Text("張數")**

**Spacer()**

**Text("\((Int(quantity) ?? 0) / 1000) 張 \((Int(quantity) ?? 0) % 1000) 股")**

**.foregroundColor(.gray)**

**}**

**}**

**Section(header: Text("交易金額")) {**

**HStack {**

**Text("總金額")**

**Spacer()**

**Text(formatCurrency(totalAmount))**

**.foregroundColor(.gray)**

**}**

**HStack {**

**Text("手續費")**

**Spacer()**

**Text(formatCurrency(fee))**

**.foregroundColor(.gray)**

**}**

**HStack {**

**Text("應付總額")**

**Spacer()**

**Text(formatCurrency(finalAmount))**

**.fontWeight(.bold)**

**}**

**}**

**Section {**

**Button(action: executeTrade) {**

**HStack {**

**Spacer()**

**if isProcessing {**

**ProgressView()**

**.padding(.trailing, 10)**

**}**

**Text(isBuy ? "買入" : "賣出")**

**.fontWeight(.semibold)**

**Spacer()**

**}**

**}**

**.disabled(isProcessing || (Int(quantity) ?? 0) <= 0)**

**.foregroundColor(.white)**

**.listRowBackground(**

**(isBuy ? Color.blue : Color.red)**

**.opacity(isProcessing || (Int(quantity) ?? 0) <= 0 ? 0.5 : 1)**

**)**

**}**

**}**

**.navigationTitle(isBuy ? "買入股票" : "賣出股票")**

**.navigationBarTitleDisplayMode(.inline)**

**.navigationBarItems(trailing: Button("取消") {**

**presentationMode.wrappedValue.dismiss()**

**})**

**.alert(isPresented: $showResult) {**

**if let error = errorMessage {**

**return Alert(**

**title: Text("交易失敗"),**

**message: Text(error),**

**dismissButton: .default(Text("確定"))**

**)**

**} else {**

**return Alert(**

**title: Text("交易成功"),**

**message: Text("\(isBuy ? "買入" : "賣出") \(quantity) 股 \(stock.name) 成功！"),**

**dismissButton: .default(Text("確定")) {**

**presentationMode.wrappedValue.dismiss()**

**}**

**)**

**}**

**}**

**}**

**}**

**private func executeTrade() {**

**guard let qty = Int(quantity), qty > 0 else { return }**

**isProcessing = true**

**let userId = UserDefaults.standard.string(forKey: "userId") ?? ""**

**// 呼叫交易服務**

**let publisher = isBuy ?**

**TradingService.shared.buyStock(userId: userId, symbol: stock.symbol, quantity: qty, price: currentPrice) :**

**TradingService.shared.sellStock(userId: userId, symbol: stock.symbol, quantity: qty, price: currentPrice)**

**publisher**

**.receive(on: DispatchQueue.main)**

**.sink(receiveCompletion: { completion in**

**if case .failure(let error) = completion {**

**errorMessage = error.localizedDescription**

**showResult = true**

**}**

**isProcessing = false**

**}, receiveValue: { transaction in**

**transactionResult = transaction**

**errorMessage = nil**

**showResult = true**

**})**

**.store(in: &cancellables)**

**}**

**private func formatCurrency(\_ value: Double) -> String {**

**let formatter = NumberFormatter()**

**formatter.numberStyle = .currency**

**formatter.locale = Locale(identifier: "zh\_TW")**

**return formatter.string(from: NSNumber(value: value)) ?? "NT$0"**

**}**

**// 用於管理 Combine 訂閱**

**private var cancellables = Set<AnyCancellable>()**

**}  
  
import SwiftUI**

**@main**

**struct InvestSimulatorApp: App {**

**var body: some Scene {**

**WindowGroup {**

**ContentView()**

**}**

**}**

**}**

**struct ContentView: View {**

**@State private var isLoggedIn = false**

**var body: some View {**

**Group {**

**if isLoggedIn {**

**DashboardView()**

**} else {**

**LoginView(isLoggedIn: $isLoggedIn)**

**}**

**}**

**}**

**}  
  
 .store(in: &cancellables)**

**}**

**import SwiftUI**

**struct LoginView: View {**

**@Binding var isLoggedIn: Bool**

**@State private var phoneNumber = ""**

**@State private var verificationCode = ""**

**@State private var isRegistering = false**

**@State private var username = ""**

**@State private var isVerificationSent = false**

**@State private var isVerifying = false**

**@State private var errorMessage: String?**

**var body: some View {**

**VStack(spacing: 30) {**

**// Logo 和標題**

**VStack(spacing: 16) {**

**Image(systemName: "chart.line.uptrend.xyaxis.circle.fill")**

**.font(.system(size: 80))**

**.foregroundColor(.blue)**

**Text("投資模擬平台")**

**.font(.largeTitle)**

**.fontWeight(.bold)**

**Text("體驗真實交易，提升投資技巧")**

**.font(.headline)**

**.foregroundColor(.gray)**

**.multilineTextAlignment(.center)**

**.padding(.horizontal)**

**}**

**// 登入表單**

**VStack(spacing: 20) {**

**if isRegistering {**

**TextField("用戶名稱", text: $username)**

**.padding()**

**.background(Color(.systemGray6))**

**.cornerRadius(10)**

**.autocapitalization(.none)**

**.disableAutocorrection(true)**

**}**

**TextField("手機號碼", text: $phoneNumber)**

**.padding()**

**.background(Color(.systemGray6))**

**.cornerRadius(10)**

**.keyboardType(.phonePad)**

**if isVerificationSent {**

**TextField("驗證碼", text: $verificationCode)**

**.padding()**

**.background(Color(.systemGray6))**

**.cornerRadius(10)**

**.keyboardType(.numberPad)**

**}**

**if let errorText = errorMessage {**

**Text(errorText)**

**.foregroundColor(.red)**

**.font(.footnote)**

**}**

**if isVerificationSent {**

**Button(action: verifyCode) {**

**HStack {**

**if isVerifying {**

**ProgressView()**

**.padding(.trailing, 10)**

**}**

**Text("驗證")**

**.fontWeight(.bold)**

**.foregroundColor(.white)**

**}**

**.frame(maxWidth: .infinity)**

**.padding()**

**.background(Color.blue)**

**.cornerRadius(10)**

**}**

**.disabled(isVerifying || verificationCode.isEmpty)**

**.opacity(isVerifying || verificationCode.isEmpty ? 0.7 : 1)**

**} else {**

**Button(action: sendVerificationCode) {**

**HStack {**

**if isVerifying {**

**ProgressView()**

**.padding(.trailing, 10)**

**}**

**Text(isRegistering ? "註冊" : "登入")**

**.fontWeight(.bold)**

**.foregroundColor(.white)**

**}**

**.frame(maxWidth: .infinity)**

**.padding()**

**.background(Color.blue)**

**.cornerRadius(10)**

**}**

**.disabled(isVerifying || (isRegistering && username.isEmpty) || phoneNumber.isEmpty)**

**.opacity(isVerifying || (isRegistering && username.isEmpty) || phoneNumber.isEmpty ? 0.7 : 1)**

**}**

**if !isVerificationSent {**

**Button(action: {**

**isRegistering.toggle()**

**}) {**

**Text(isRegistering ? "已有帳號？登入" : "還沒有帳號？註冊")**

**.foregroundColor(.blue)**

**}**

**}**

**}**

**.padding(.horizontal)**

**// 免責聲明**

**VStack(spacing: 4) {**

**Text("註冊即表示您同意我們的")**

**.font(.caption)**

**.foregroundColor(.gray)**

**HStack {**

**Button(action: {}) {**

**Text("使用條款")**

**.font(.caption)**

**.foregroundColor(.blue)**

**}**

**Text("與")**

**.font(.caption)**

**.foregroundColor(.gray)**

**Button(action: {}) {**

**Text("隱私政策")**

**.font(.caption)**

**.foregroundColor(.blue)**

**}**

**}**

**}**

**}**

**.padding(.horizontal)**

**}**

**private func sendVerificationCode() {**

**guard validatePhoneNumber() else {**

**errorMessage = "請輸入有效的台灣手機號碼"**

**return**

**}**

**isVerifying = true**

**errorMessage = nil**

**// 應呼叫 UserService**

**UserService.shared.sendVerificationCode(phoneNumber: phoneNumber)**

**.receive(on: DispatchQueue.main)**

**.sink(receiveCompletion: { completion in**

**if case .failure(let error) = completion {**

**errorMessage = "發送驗證碼失敗：\(error.localizedDescription)"**

**}**

**isVerifying = false**

**}, receiveValue: { response in**

**if response.success {**

**isVerificationSent = true**

**} else {**

**errorMessage = response.message**

**}**

**isVerifying = false**

**})**

**.store(in: &cancellables)**

**}**

**private func verifyCode() {**

**isVerifying = true**

**errorMessage = nil**

**// 應呼叫 UserService**

**UserService.shared.verifyCode(phoneNumber: phoneNumber, code: verificationCode)**

**.receive(on: DispatchQueue.main)**

**.sink(receiveCompletion: { completion in**

**if case .failure(let error) = completion {**

**errorMessage = "驗證失敗：\(error.localizedDescription)"**

**}**

**isVerifying = false**

**}, receiveValue: { response in**

**if response.verified, let userId = response.userId, let token = response.token {**

**// 儲存用戶資訊**

**UserDefaults.standard.set(userId, forKey: "userId")**

**UserDefaults.standard.set(token, forKey: "userToken")**

**isLoggedIn = true**

**} else {**

**errorMessage = "驗證碼無效，請重試"**

**}**

**isVerifying = false**

**})**

**.store(in  
 private func validatePhoneNumber() -> Bool {**

**// 簡易台灣手機號碼驗證**

**let phoneRegex = "^09[0-9]{8}$"**

**let phoneTest = NSPredicate(format: "SELF MATCHES %@", phoneRegex)**

**return phoneTest.evaluate(with: phoneNumber)**

**}**

**// 用於管理 Combine 訂閱**

**private var cancellables = Set<AnyCancellable>()**

**}  
from fastapi import FastAPI, Request, HTTPException**

**from fastapi.middleware.cors import CORSMiddleware**

**from fastapi.responses import JSONResponse**

**import uvicorn**

**import os**

**from datetime import datetime**

**import logging**

**# 導入各個模塊**

**from stock\_api import app as stock\_app**

**from trading\_api import app as trading\_app**

**from ai\_analysis import app as ai\_app**

**from user\_management import app as user\_app**

**# 創建主應用**

**app = FastAPI(**

**title="投資模擬平台 API",**

**description="虛擬股票投資模擬平台的後端 API",**

**version="1.0.0"**

**)**

**# 設定 CORS**

**app.add\_middleware(**

**CORSMiddleware,**

**allow\_origins=["\*"], # 生產環境中應該設定特定的域名**

**allow\_credentials=True,**

**allow\_methods=["\*"],**

**allow\_headers=["\*"],**

**)**

**# 全域異常處理**

**@app.exception\_handler(Exception)**

**async def global\_exception\_handler(request: Request, exc: Exception):**

**logging.error(f"Global exception: {exc}")**

**return JSONResponse(**

**status\_code=500,**

**content={"detail": "Internal server error"}**

**)**

**# 健康檢查**

**@app.get("/health")**

**async def health\_check():**

**return {"status": "healthy", "timestamp": datetime.now().isoformat()}**

**# 掛載子應用**

**app.mount("/stock", stock\_app)**

**app.mount("/trading", trading\_app)**

**app.mount("/ai", ai\_app)**

**app.mount("/users", user\_app)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**uvicorn.run(**

**"main:app",**

**host="0.0.0.0",**

**port=int(os.getenv("PORT", 8000)),**

**reload=True**

**)  
  
from fastapi import FastAPI, HTTPException, Depends**

**from pydantic import BaseModel**

**from typing import Optional**

**import uuid**

**import os**

**import random**

**import string**

**from datetime import datetime, timedelta**

**from supabase import create\_client, Client**

**import hashlib**

**import jwt**

**import requests**

**app = FastAPI()**

**# Supabase 連接設定**

**SUPABASE\_URL = os.getenv("SUPABASE\_URL")**

**SUPABASE\_KEY = os.getenv("SUPABASE\_KEY")**

**supabase: Client = create\_client(SUPABASE\_URL, SUPABASE\_KEY)**

**# SMS API 設定**

**SMS\_API\_URL = os.getenv("SMS\_API\_URL")**

**SMS\_API\_KEY = os.getenv("SMS\_API\_KEY")**

**# JWT 設定**

**JWT\_SECRET = os.getenv("JWT\_SECRET", "your-secret-key")**

**JWT\_ALGORITHM = "HS256"**

**# 請求模型**

**class RegistrationRequest(BaseModel):**

**username: str**

**phoneNumber: str**

**class SMSRequest(BaseModel):**

**phoneNumber: str**

**class VerificationRequest(BaseModel):**

**phoneNumber: str**

**code: str**

**class InvitationRequest(BaseModel):**

**userId: str**

**invitationCode: str**

**class GroupRequest(BaseModel):**

**name: str**

**ownerId: str**

**description: str**

**# 用戶註冊**

**@app.post("/register")**

**async def register\_user(request: RegistrationRequest):**

**try:**

**# 檢查手機號碼是否已存在**

**existing\_user = supabase.table("users").select("\*").eq("phoneNumber", request.phoneNumber).execute()**

**if existing\_user.data:**

**raise HTTPException(status\_code=400, detail="Phone number already registered")**

**# 生成邀請碼**

**invitation\_code = generate\_invitation\_code()**

**# 創建用戶**

**user\_id = str(uuid.uuid4())**

**user\_data = {**

**"id": user\_id,**

**"username": request.username,**

**"phoneNumber": request.phoneNumber,**

**"initialCapital": 1000000, # 100萬模擬金**

**"availableCash": 1000000,**

**"invitationCode": invitation\_code,**

**"createdAt": datetime.now().isoformat()**

**}**

**result = supabase.table("users").insert(user\_data).execute()**

**return {**

**"userId": user\_id,**

**"username": request.username,**

**"initialCapital": 1000000,**

**"invitationCode": invitation\_code**

**}**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))**

**# 發送SMS驗證碼**

**@app.post("/send-verification")**

**async def send\_verification\_code(request: SMSRequest):**

**try:**

**# 生成6位數驗證碼**

**verification\_code = str(random.randint(100000, 999999))**

**# 儲存驗證碼到資料庫（有效期10分鐘）**

**verification\_data = {**

**"id": str(uuid.uuid4()),**

**"phoneNumber": request.phoneNumber,**

**"code": verification\_code,**

**"expiresAt": (datetime.now() + timedelta(minutes=10)).isoformat(),**

**"used": False**

**}**

**supabase.table("verification\_codes").insert(verification\_data).execute()**

**# 發送SMS**

**sms\_result = send\_sms(request.phoneNumber, f"您的驗證碼是：{verification\_code}，有效期10分鐘。")**

**if sms\_result:**

**return {"success": True, "message": "驗證碼已發送"}**

**else:**

**raise HTTPException(status\_code=500, detail="Failed to send SMS")**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))**

**# 驗證SMS驗證碼**

**@app.post("/verify-code")**

**async def verify\_code(request: VerificationRequest):**

**try:**

**# 查詢驗證碼**

**verification\_data = supabase.table("verification\_codes").select("\*").eq("phoneNumber", request.phoneNumber).eq("code", request.code).eq("used", False).execute()**

**if not verification\_data.data:**

**raise HTTPException(status\_code=400, detail="Invalid verification code")**

**verification = verification\_data.data[0]**

**# 檢查是否過期**

**expires\_at = datetime.fromisoformat(verification["expiresAt"].replace("Z", "+00:00"))**

**if datetime.now() > expires\_at:**

**raise HTTPException(status\_code=400, detail="Verification code expired")**

**# 標記驗證碼為已使用**

**supabase.table("verification\_codes").update({"used": True}).eq("id", verification["id"]).execute()**

**# 查詢或創建用戶**

**user\_data = supabase.table("users").select("\*").eq("phoneNumber", request.phoneNumber).execute()**

**if user\_data.data:**

**user = user\_data.data[0]**

**user\_id = user["id"]**

**else:**

**# 創建新用戶**

**user\_id = str(uuid.uuid4())**

**invitation\_code = generate\_invitation\_code()**

**new\_user\_data = {**

**"id": user\_id,**

**"username": f"用戶{user\_id[:8]}",**

**"phoneNumber": request.phoneNumber,**

**"initialCapital": 1000000,**

**"availableCash": 1000000,**

**"invitationCode": invitation\_code,**

**"createdAt": datetime.now().isoformat()**

**}**

**supabase.table("users").insert(new\_user\_data).execute()**

**# 生成JWT token**

**token = generate\_jwt\_token(user\_id)**

**return {**

**"verified": True,**

**"userId": user\_id,**

**"token": token**

**}**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))**

**# 使用邀請碼**

**@app.post("/use-invitation")**

**async def use\_invitation\_code(request: InvitationRequest):**

**try:**

**# 查詢邀請碼擁有者**

**inviter\_data = supabase.table("users").select("\*").eq("invitationCode", request.invitationCode).execute()**

**if not inviter\_data.data:**

**raise HTTPException(status\_code=400, detail="Invalid invitation code")**

**inviter = inviter\_data.data[0]**

**# 檢查用戶是否存在**

**user\_data = supabase.table("users").select("\*").eq("id", request.userId).execute()**

**if not user\_data.data:**

**raise HTTPException(status\_code=400, detail="User not found")**

**user = user\_data.data[0]**

**# 檢查是否已使用過邀請碼**

**invitation\_usage = supabase.table("invitation\_usage").select("\*").eq("userId", request.userId).execute()**

**if invitation\_usage.data:**

**raise HTTPException(status\_code=400, detail="You have already used an invitation code")**

**# 給雙方增加獎勵資金**

**bonus\_amount = 50000 # 5萬模擬金獎勵**

**# 更新邀請者資金**

**new\_inviter\_cash = inviter["availableCash"] + bonus\_amount**

**supabase.table("users").update({"availableCash": new\_inviter\_cash}).eq("id", inviter["id"]).execute()**

**# 更新用戶資金**

**new\_user\_cash = user["availableCash"] + bonus\_amount**

**supabase.table("users").update({"availableCash": new\_user\_cash}).eq("id", request.userId).execute()**

**# 記錄邀請碼使用**

**usage\_data = {**

**"id": str(uuid.uuid4()),**

**"userId": request.userId,**

**"inviterId": inviter["id"],**

**"invitationCode": request.invitationCode,**

**"bonusAmount": bonus\_amount,**

**"usedAt": datetime.now().isoformat()**

**}**

**supabase.table("invitation\_usage").insert(usage\_data).execute()**

**return {**

**"success": True,**

**"bonusAmount": bonus\_amount,**

**"message": f"成功使用邀請碼，您和邀請者各獲得 NT${bonus\_amount:,} 模擬金！"**

**}**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))**

**# 創建投資群組**

**@app.post("/groups")**

**async def create\_investment\_group(request: GroupRequest):**

**try:**

**# 檢查用戶是否存在**

**user\_data = supabase.table("users").select("\*").eq("id", request.ownerId).execute()**

**if not user\_data.data:**

**raise HTTPException(status\_code=400, detail="User not found")**

**# 創建群組**

**group\_id = str(uuid.uuid4())**

**group\_data = {**

**"id": group\_id,**

**"name": request.name,**

**"ownerId": request.ownerId,**

**"description": request.description,**

**"createdAt": datetime.now().isoformat(),**

**"members": [request.ownerId] # 創建者自動成為成員**

**}**

**result = supabase.table("investment\_groups").insert(group\_data).execute()**

**return result.data[0]**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))**

**# 獲取用戶群組**

**@app.get("/groups")**

**async def get\_user\_groups(userId: str):**

**try:**

**# 查詢用戶參與的群組**

**groups\_data = supabase.table("investment\_groups").select("\*").contains("members", [userId]).execute()**

**return {"groups": groups\_data.data}**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))**

**# 加入群組**

**@app.post("/groups/{group\_id}/join")**

**async def join\_group(group\_id: str, userId: str):**

**try:**

**# 檢查群組是否存在**

**group\_data = supabase.table("investment\_groups").select("\*").eq("id", group\_id).execute()**

**if not group\_data.data:**

**raise HTTPException(status\_code=404, detail="Group not found")**

**group = group\_data.data[0]**

**# 檢查用戶是否已是成員**

**if userId in group["members"]:**

**raise HTTPException(status\_code=400, detail="User is already a member")**

**# 添加用戶到群組**

**updated\_members = group["members"] + [userId]**

**supabase.table("investment\_groups").update({"members": updated\_members}).eq("id", group\_id).execute()**

**return {"success": True, "message": "Successfully joined group"}**

**except Exception as e:**

**raise HTTPException(status\_code=500, detail=str(e))**

**# 輔助函數**

**def generate\_invitation\_code():**

**"""生成6位數邀請碼"""**

**return ''.join(random.choices(string.ascii\_uppercase + string.digits, k=6))**

**def send\_sms(phone\_number: str, message: str):**

**"""發送SMS (需要實際的SMS服務商API)"""**

**try:**

**# 這裡應該整合實際的SMS服務商API**

**# 例如：台灣大哥大、中華電信等SMS服務**

**# 模擬SMS發送**

**if SMS\_API\_URL and SMS\_API\_KEY:**

**# 實際API調用**

**response = requests.post(**

**SMS\_API\_URL,**

**headers={**

**"Authorization": f"Bearer {SMS\_API\_KEY}",**

**"Content-Type": "application/json"**

**},**

**json={**

**"to": phone\_number,**

**"message": message**

**}**

**)**

**return response.status\_code == 200**

**else:**

**# 開發環境模擬**

**print(f"SMS to {phone\_number}: {message}")**

**return True**

**except Exception as e:**

**print(f"SMS sending failed: {e}")**

**return False**

**def generate\_jwt\_token(user\_id: str):**

**"""生成JWT token"""**

**payload = {**

**"user\_id": user\_id,**

**"exp": datetime.now() + timedelta(days=30) # 30天有效期**

**}**

**return jwt.encode(payload, JWT\_SECRET, algorithm=JWT\_ALGORITHM)**

**def verify\_jwt\_token(token: str):**

**"""驗證JWT token"""**

**try:**

**payload = jwt.decode(token, JWT\_SECRET, algorithms=[JWT\_ALGORITHM])**

**return payload["user\_id"]**

**except jwt.ExpiredSignatureError:**

**raise HTTPException(status\_code=401, detail="Token expired")**

**except jwt.InvalidTokenError:**

**raise HTTPException(status\_code=401, detail="Invalid token")  
  
-- 用戶表**

**CREATE TABLE users (**

**id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),**

**username VARCHAR(50) NOT NULL,**

**phone\_number VARCHAR(20) UNIQUE NOT NULL,**

**initial\_capital DECIMAL(15, 2) DEFAULT 1000000,**

**available\_cash DECIMAL(15, 2) DEFAULT 1000000,**

**invitation\_code VARCHAR(10) UNIQUE NOT NULL,**

**created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),**

**updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()**

**);**

**-- 投資群組表**

**CREATE TABLE investment\_groups (**

**id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),**

**name VARCHAR(100) NOT NULL,**

**owner\_id UUID REFERENCES users(id),**

**description TEXT,**

**members UUID[] DEFAULT '{}',**

**created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()**

**);**

**-- 持倉表**

**CREATE TABLE positions (**

**id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),**

**user\_id UUID REFERENCES users(id),**

**symbol VARCHAR(10) NOT NULL,**

**quantity INTEGER NOT NULL,**

**average\_cost DECIMAL(10, 2) NOT NULL,**

**purchase\_date TIMESTAMP WITH TIME ZONE DEFAULT NOW(),**

**market\_type VARCHAR(10) NOT NULL,**

**created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),**

**updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()**

**);**

**-- 交易記錄表**

**CREATE TABLE transactions (**

**id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),**

**user\_id UUID REFERENCES users(id),**

**symbol VARCHAR(10) NOT NULL,**

**action VARCHAR(4) NOT NULL CHECK (action IN ('buy', 'sell')),**

**quantity INTEGER NOT NULL,**

**price DECIMAL(10, 2) NOT NULL,**

**fee DECIMAL(10, 2) NOT NULL,**

**total\_amount DECIMAL(15, 2) NOT NULL,**

**profit DECIMAL(15, 2),**

**timestamp TIMESTAMP WITH TIME ZONE DEFAULT NOW()**

**);**

**-- 績效快照表**

**CREATE TABLE performance\_snapshots (**

**id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),**

**user\_id UUID REFERENCES users(id),**

**portfolio\_value DECIMAL(15, 2) NOT NULL,**

**return\_rate DECIMAL(5, 2) NOT NULL,**

**period VARCHAR(10) NOT NULL CHECK (period IN ('week', 'month', 'year')),**

**timestamp TIMESTAMP WITH TIME ZONE DEFAULT NOW()**

**);**

**-- 驗證碼表**

**CREATE TABLE verification\_codes (**

**id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),**

**phone\_number VARCHAR(20) NOT NULL,**

**code VARCHAR(6) NOT NULL,**

**expires\_at TIMESTAMP WITH TIME ZONE NOT NULL,**

**used BOOLEAN DEFAULT FALSE,**

**created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()**

**);**

**-- 邀請碼使用記錄表**

**CREATE TABLE invitation\_usage (**

**id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),**

**user\_id UUID REFERENCES users(id),**

**inviter\_id UUID REFERENCES users(id),**

**invitation\_code VARCHAR(10) NOT NULL,**

**bonus\_amount DECIMAL(15, 2) NOT NULL,**

**used\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()**

**);**

**-- 創建索引**

**CREATE INDEX idx\_users\_phone ON users(phone\_number);**

**CREATE INDEX idx\_positions\_user\_symbol ON positions(user\_id, symbol);**

**CREATE INDEX idx\_transactions\_user\_timestamp ON transactions(user\_id, timestamp DESC);**

**CREATE INDEX idx\_performance\_snapshots\_user\_period ON performance\_snapshots(user\_id, period);**

**CREATE INDEX idx\_verification\_codes\_phone\_code ON verification\_codes(phone\_number, code);**

**-- 創建更新時間觸發器**

**CREATE OR REPLACE FUNCTION update\_updated\_at\_column()**

**RETURNS TRIGGER AS $$**

**BEGIN**

**NEW.updated\_at = NOW();**

**RETURN NEW;**

**END;**

**$$ language 'plpgsql';**

**CREATE TRIGGER update\_users\_updated\_at BEFORE UPDATE ON users**

**FOR EACH ROW EXECUTE FUNCTION update\_updated\_at\_column();**

**CREATE TRIGGER update\_positions\_updated\_at BEFORE UPDATE ON positions**

**FOR EACH ROW EXECUTE FUNCTION update\_updated\_at\_column();  
import schedule**

**import time**

**from datetime import datetime, timedelta**

**import yfinance as yf**

**from supabase import create\_client, Client**

**import os**

**import logging**

**# 設定日誌**

**logging.basicConfig(level=logging.INFO)**

**logger = logging.getLogger(\_\_name\_\_)**

**# Supabase 連接設定**

**SUPABASE\_URL = os.getenv("SUPABASE\_URL")**

**SUPABASE\_KEY = os.getenv("SUPABASE\_KEY")**

**supabase: Client = create\_client(SUPABASE\_URL, SUPABASE\_KEY)**

**def update\_performance\_snapshots():**

**"""更新績效快照 - 每天執行"""**

**try:**

**logger.info("開始更新績效快照...")**

**# 獲取所有用戶**

**users\_data = supabase.table("users").select("\*").execute()**

**users = users\_data.data**

**for user in users:**

**user\_id = user["id"]**

**# 計算當前投資組合價值**

**portfolio\_value = calculate\_portfolio\_value(user\_id)**

**total\_assets = user["availableCash"] + portfolio\_value**

**# 計算報酬率**

**return\_rate = ((total\_assets - user["initialCapital"]) / user["initialCapital"]) \* 100**

**# 更新週績效快照**

**update\_period\_snapshot(user\_id, "week", total\_assets, return\_rate)**

**# 更新月績效快照**

**update\_period\_snapshot(user\_id, "month", total\_assets, return\_rate)**

**# 更新年績效快照**

**update\_period\_snapshot(user\_id, "year", total\_assets, return\_rate)**

**logger.info("績效快照更新完成")**

**except Exception as e:**

**logger.error(f"更新績效快照失敗: {e}")**

**def calculate\_portfolio\_value(user\_id: str) -> float:**

**"""計算用戶投資組合價值"""**

**try:**

**# 獲取用戶持倉**

**positions\_data = supabase.table("positions").select("\*").eq("userId", user\_id).execute()**

**positions = positions\_data.data**

**total\_value = 0**

**for position in positions:**

**symbol = position["symbol"]**

**quantity = position["quantity"]**

**# 獲取即時價格**

**try:**

**ticker = yf.Ticker(f"{symbol}.TW")**

**data = ticker.history(period="1d")**

**if not data.empty:**

**current\_price = data['Close'].iloc[-1]**

**position\_value = quantity \* current\_price**

**total\_value += position\_value**

**else:**

**# 如果無法獲取即時價格，使用平均成本**

**position\_value = quantity \* position["averageCost"]**

**total\_value += position\_value**

**except Exception as e:**

**logger.warning(f"無法獲取 {symbol} 的即時價格，使用平均成本: {e}")**

**position\_value = quantity \* position["averageCost"]**

**total\_value += position\_value**

**return total\_value**

**except Exception as e:**

**logger.error(f"計算投資組合價值失敗: {e}")**

**return 0**

**def update\_period\_snapshot(user\_id: str, period: str, portfolio\_value: float, return\_rate: float):**

**"""更新指定期間的績效快照"""**

**try:**

**# 檢查是否已有今日的快照**

**today = datetime.now().date()**

**existing\_snapshot = supabase.table("performance\_snapshots").select("\*").eq("userId", user\_id).eq("period", period).gte("timestamp", today).execute()**

**if existing\_snapshot.data:**

**# 更新現有快照**

**snapshot\_id = existing\_snapshot.data[0]["id"]**

**supabase.table("performance\_snapshots").update({**

**"portfolioValue": portfolio\_value,**

**"returnRate": return\_rate,**

**"timestamp": datetime.now().isoformat()**

**}).eq("id", snapshot\_id).execute()**

**else:**

**# 創建新快照**

**snapshot\_data = {**

**"userId": user\_id,**

**"portfolioValue": portfolio\_value,**

**"returnRate": return\_rate,**

**"period": period,**

**"timestamp": datetime.now().isoformat()**

**}**

**supabase.table("performance\_snapshots").insert(snapshot\_data).execute()**

**except Exception as e:**

**logger.error(f"更新 {period} 快照失敗: {e}")**

**def cleanup\_old\_verification\_codes():**

**"""清理過期的驗證碼"""**

**try:**

**logger.info("清理過期驗證碼...")**

**# 刪除1小時前的驗證碼**

**cutoff\_time = datetime.now() - timedelta(hours=1)**

**supabase.table("verification\_codes").delete().lt("expiresAt", cutoff\_time.isoformat()).execute()**

**logger.info("過期驗證碼清理完成")**

**except Exception as e:**

**logger.error(f"清理驗證碼失敗: {e}")**

**def backup\_daily\_data():**

**"""每日數據備份"""**

**try:**

**logger.info("開始每日數據備份...")**

**# 備份關鍵表格數據**

**tables\_to\_backup = ["users", "positions", "transactions", "performance\_snapshots"]**

**for table in tables\_to\_backup:**

**data = supabase.table(table).select("\*").execute()**

**# 這裡可以實現將數據備份到檔案或其他儲存服務**

**# 例如：AWS S3、Google Cloud Storage等**

**logger.info(f"備份 {table} 表格: {len(data.data)} 筆記錄")**

**logger.info("每日數據備份完成")**

**except Exception as e:**

**logger.error(f"數據備份失敗: {e}")**

**# 設定排程任務**

**def setup\_scheduled\_tasks():**

**"""設定排程任務"""**

**# 每天凌晨2點更新績效快照**

**schedule.every().day.at("02:00").do(update\_performance\_snapshots)**

**# 每小時清理過期驗證碼**

**schedule.every().hour.do(cleanup\_old\_verification\_codes)**

**# 每天凌晨3點備份數據**

**schedule.every().day.at("03:00").do(backup\_daily\_data)**

**logger.info("排程任務設定完成")**

**def run\_scheduler():**

**"""執行排程器"""**

**setup\_scheduled\_tasks()**

**logger.info("排程器開始運行...")**

**while True:**

**schedule.run\_pending()**

**time.sleep(60) # 每分鐘檢查一次**

**if \_\_name\_\_ == "\_\_main\_\_":**

**run\_scheduler()  
  
version: '3.8'**

**services:**

**# 後端API服務**

**backend:**

**build: ./backend**

**ports:**

**- "8000:8000"**

**environment:**

**- SUPABASE\_URL=${SUPABASE\_URL}**

**- SUPABASE\_KEY=${SUPABASE\_KEY}**

**- SMS\_API\_URL=${SMS\_API\_URL}**

**- SMS\_API\_KEY=${SMS\_API\_KEY}**

**- JWT\_SECRET=${JWT\_SECRET}**

**depends\_on:**

**- redis**

**volumes:**

**- ./backend:/app**

**command: uvicorn main:app --host 0.0.0.0 --port 8000 --reload**

**# 排程任務服務**

**scheduler:**

**build: ./backend**

**environment:**

**- SUPABASE\_URL=${SUPABASE\_URL}**

**- SUPABASE\_KEY=${SUPABASE\_KEY}**

**volumes:**

**- ./backend:/app**

**command: python scheduler.py**

**# Redis 快取服務**

**redis:**

**image: redis:7-alpine**

**ports:**

**- "6379:6379"**

**volumes:**

**- redis\_data:/data**

**# Nginx 反向代理**

**nginx:**

**image: nginx:alpine**

**ports:**

**- "80:80"**

**- "443:443"**

**volumes:**

**- ./nginx.conf:/etc/nginx/nginx.conf**

**- ./ssl:/etc/nginx/ssl**

**depends\_on:**

**- backend**

**volumes:**

**redis\_data:  
  
FROM python:3.11-slim**

**# 設定工作目錄**

**WORKDIR /app**

**# 複製需求檔案**

**COPY requirements.txt .**

**# 安裝依賴**

**RUN pip install --no-cache-dir -r requirements.txt**

**# 複製應用程式檔案**

**COPY . .**

**# 暴露端口**

**EXPOSE 8000**

**# 啟動命令**

**CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]  
  
fastapi==0.104.1**

**uvicorn[standard]==0.24.0**

**supabase==2.0.0**

**yfinance==0.2.28**

**pandas==2.1.3**

**numpy==1.25.2**

**requests==2.31.0**

**python-jose[cryptography]==3.3.0**

**python-multipart==0.0.6**

**schedule==1.2.0**

**redis==5.0.1**

**pydantic==2.4.2**

**python-dotenv==1.0.0  
  
events {**

**worker\_connections 1024;**

**}**

**http {**

**upstream backend {**

**server backend:8000;**

**}**

**server {**

**listen 80;**

**server\_name your-domain.com;**

**# API 路由**

**location /api/ {**

**proxy\_pass http://backend/;**

**proxy\_set\_header Host $host;**

**proxy\_set\_header X-Real-IP $remote\_addr;**

**proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;**

**proxy\_set\_header X-Forwarded-Proto $scheme;**

**}**

**# 健康檢查**

**location /health {**

**proxy\_pass http://backend/health;**

**}**

**# 根路由重定向**

**location / {**

**return 301 https://apps.apple.com/app/your-app-id;**

**}**

**}**

**# HTTPS 設定 (如果需要)**

**server {**

**listen 443 ssl;**

**server\_name your-domain.com;**

**ssl\_certificate /etc/nginx/ssl/cert.pem;**

**ssl\_certificate\_key /etc/nginx/ssl/key.pem;**

**location /api/ {**

**proxy\_pass http://backend/;**

**proxy\_set\_header Host $host;**

**proxy\_set\_header X-Real-IP $remote\_addr;**

**proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;**

**proxy\_set\_header X-Forwarded-Proto $scheme;**

**}**

**}**

**}  
  
# Supabase 設定**

**SUPABASE\_URL=https://your-project.supabase.co**

**SUPABASE\_KEY=your-anon-key**

**# SMS API 設定**

**SMS\_API\_URL=https://sms-provider-api.com/send**

**SMS\_API\_KEY=your-sms-api-key**

**# JWT 設定**

**JWT\_SECRET=your-super-secret-jwt-key**

**# 環境設定**

**ENVIRONMENT=development**

**DEBUG=true**

**# 資料庫設定 (如果使用自建資料庫)**

**DATABASE\_URL=postgresql://user:password@localhost:5432/invest\_simulator**

**# Redis 設定**

**REDIS\_URL=redis://localhost:6379  
  
融合chatgpt deepthink  
給出得結果**好的，我會為你規劃一份完整的技術開發計劃書，詳列如何建立你的跨前後端模擬投資 App，包含 Swift 前端、Python Flask 後端、yfinance 串接、交易記錄與損益計算機制，以及 Supabase 時程任務等，格式會方便你直接交給 Claude 實作。完成後會通知你。