database.py、accounts.py、accounts\_server.py、accounts\_client.py的 **系統功能規格（Functional Specification**

**✅ 系統功能規格（Functional Specification）**

以下規格包含：

* Database 模組
* Accounts 模組
* Accounts Server
* Accounts Client

**✅ Database 功能規格**

**功能目標**

* 提供一個 SQLite 資料庫，儲存使用者帳戶、交易紀錄、以及市場價格資訊。
* 作為 Accounts 模組與 Server 模組的底層數據儲存。

**功能描述**

**1. 賬戶資料表 (accounts)**

* 欄位：
  + name (TEXT, Primary Key) → 使用者名稱（小寫）
  + account (TEXT) → 帳戶 JSON 資料
* 功能：
  + 新增或更新使用者帳戶資料。

**2. 日誌資料表 (logs)**

* 欄位：
  + id (INTEGER, PK, AUTOINCREMENT)
  + name (TEXT) → 使用者名稱
  + datetime (DATETIME) → 紀錄時間
  + type (TEXT) → 日誌類型（例如 account）
  + message (TEXT) → 訊息內容
* 功能：
  + 寫入使用者相關操作紀錄。
  + 查詢指定使用者最新 N 筆日誌。

**3. 市場資料表 (market)**

* 欄位：
  + date (TEXT, PK) → 日期
  + data (TEXT) → 當日市場資料（JSON）
* 功能：
  + 寫入或更新指定日期的市場行情數據。
  + 讀取指定日期的市場行情。

**✅ Accounts 功能規格**

**功能目標**

* 管理每個使用者的投資帳戶，實現資金管理、股票買賣、策略設定、以及報告產出。
* 提供一組可被外部服務呼叫的帳戶操作方法。

**功能描述**

**帳戶物件（Account Class）**

* 帳戶屬性：
  + name → 帳戶名稱
  + balance → 現金餘額
  + strategy → 投資策略描述
  + holdings → 持股明細（symbol → 數量）
  + transactions → 所有交易紀錄
  + portfolio\_value\_time\_series → 投資組合價值時間序列

**功能方法**

**1. 建立或讀取帳戶**

* Account.get(name)
  + 若帳戶存在 → 從 DB 讀取
  + 若帳戶不存在 → 建立初始帳戶（初始餘額 $10,000）

**2. 存入資金**

* deposit(amount)
  + 增加帳戶現金
  + 更新 DB

**3. 提領資金**

* withdraw(amount)
  + 減少帳戶現金
  + 更新 DB
  + 須檢查餘額不可負數

**4. 購買股票**

* buy\_shares(symbol, quantity, rationale)
  + 根據即時股價進行購買
  + 計算交易總額，檢查資金是否足夠
  + 更新持股與交易紀錄
  + 寫入 DB 與日誌

**5. 賣出股票**

* sell\_shares(symbol, quantity, rationale)
  + 檢查是否持有足夠股票
  + 進行賣出交易，更新現金
  + 更新持股與交易紀錄
  + 寫入 DB 與日誌

**6. 計算投資組合價值**

* calculate\_portfolio\_value()
  + 計算現金 + 持股市值

**7. 計算損益**

* calculate\_profit\_loss()
  + 計算帳戶累積損益

**8. 取得持股明細**

* get\_holdings()

**9. 取得策略描述**

* get\_strategy()

**10. 變更策略**

* change\_strategy(strategy)
  + 記錄策略更新

**11. 帳戶報告**

* report()
  + 回傳帳戶 JSON 詳細資料

**✅ Accounts Server 功能規格**

**功能目標**

* 將 Accounts 模組封裝成一組 MCP 工具與資源 (tools, resources)
* 讓外部系統（如 agents, clients）可透過 MCP 協定呼叫操作帳戶。

**功能描述**

**MCP 工具**

* get\_balance(name)
  + 回傳指定帳戶的現金餘額
* get\_holdings(name)
  + 回傳指定帳戶持股明細
* buy\_shares(name, symbol, quantity, rationale)
  + 呼叫帳戶買入股票流程
* sell\_shares(name, symbol, quantity, rationale)
  + 呼叫帳戶賣出股票流程
* change\_strategy(name, strategy)
  + 變更指定帳戶的投資策略

**MCP 資源**

* accounts://accounts\_server/{name}
  + 回傳完整帳戶報告 JSON
* accounts://strategy/{name}
  + 回傳帳戶策略描述

**運行模式**

* Server 預設執行 mcp.run(transport='stdio')
* 使用 STDIO 作為代理交互方式

**✅ Accounts Client 功能規格**

**功能目標**

* 建立一個 Client，可透過 MCP STDIO 協定呼叫 Accounts Server
* 支援列出工具、呼叫工具、讀取資源
* 適合作為 AI Agents 的接口

**功能描述**

**功能函式**

* **list\_accounts\_tools**()
  + 列出 **Server** 暴露的所有工具
* call\_accounts\_tool(tool\_name, tool\_args)
  + 呼叫 Server 端的 MCP 工具
* read\_accounts\_resource(name)
  + 讀取帳戶完整報告
* read\_strategy\_resource(name)
  + 讀取帳戶策略描述
* **get\_accounts\_tools\_openai**()
  + 將 MCP 工具轉換成 OpenAI Function Calling 格式，以便與 LLM 整合

**執行方式**

* 預設執行以下指令，啟動 Server：
* uv run accounts\_server.py
* 透過 STDIO channel 與 Server 溝通

**✅ 系統整合流程**

1. **使用者請求**  
   → 通過 Postman 或 Client 呼叫 Client API
2. **Client**  
   → 透過 MCP STDIO 傳送工具名稱 + 參數
3. **Server**  
   → 執行對應工具，呼叫 Accounts 模組
4. **Accounts 模組**  
   → 操作資料庫 + 計算邏輯 → 回傳結果
5. **Client 接收**  
   → 回傳 JSON 給外部系統或 Postman