# 銀行系統專題報告

## 第一章 緒論

### 研究背景與動機

隨著科技的飛速發展，金融服務業正經歷前所未有的數位轉型。銀行系統作為金融機構的核心，其安全性、效率與使用者體驗直接影響銀行營運及客戶滿意度。傳統銀行業務面臨線上化、自動化的挑戰，同時也需要因應日益嚴峻的資安威脅。本專題旨在開發一個桌面銀行系統應用程式，以滿足銀行內部員工日常營運管理任務，並提供客戶便捷的帳戶查詢與基本金融交易服務。

### 研究目的

1. 設計並實作一個安全、高效且直觀的桌面銀行系統，支援多角色使用者認證與授權。
2. 實現核心銀行業務功能，包括客戶帳戶管理（存款、取款、新增客戶）、銀行內部員工管理及銀行營運資金管理。
3. 建立詳盡的交易記錄與審計日誌追蹤機制，確保資料可追溯性與合規性。
4. 評估所開發系統的效能、安全性、可用性與可靠性，並提出未來改進方向。

### 問題陳述

現有銀行系統可能存在操作複雜、介面不直觀、權限管理不夠細緻等問題，或在安全性方面仍有提升空間。對於中小型銀行或特定業務場景，尋求輕量級、易於部署且功能齊全的桌面應用程式，以簡化日常業務流程並提高效率，是一個實際的需求。本專題將嘗試解決如何在有限資源下，打造一個兼顧功能、效能與安全性的銀行系統。

這個專題設計此銀行系統，主要是針對中小型銀行或特定業務場景的需求，提供一個輕量級、易於部署且功能齊全的桌面應用程式，以簡化日常業務流程並提高效率 。

現有大型銀行系統可能存在以下問題，使得此專題系統有其設計情境和意義：

* **操作複雜性與介面直觀性不足**：大型銀行系統可能操作複雜，介面不夠直觀，導致員工需要較長時間培訓才能上手，且可能增加人為錯誤 。
* **權限管理不夠細緻**：大型系統的權限管理可能無法滿足某些中小型銀行對輕量級、細緻權限控制的需求 。
* **部署與資源限制**：大型銀行系統通常需要龐大的基礎設施和資源投入，對於資源有限的中小型銀行而言，部署和維護成本高昂。此專題系統採用 SQLite 輕量級資料庫，更易於部署和管理 。
* **示範與參考價值**：此專題開發的桌面銀行系統應用程式，可作為運用 C# 和 SQLite 技術實現銀行核心業務功能的示範，為未來的銀行系統開發專案提供參考設計與實作經驗 。

因此，此系統的設計情境是為了解決特定規模或業務需求下的銀行，對於輕量級、高效率、易於部署和操作的桌面應用程式的實際需求 。

### 預期貢獻

本專題的成果預期能提供一個可作為示範的桌面銀行系統應用程式，展現如何運用 C# 和 SQLite 技術實現銀行核心業務功能。這將有助於：

1. 為未來的銀行系統開發專案提供參考設計與實作經驗。
2. 提升內部員工的操作效率，降低人為錯誤。
3. 為客戶提供更便捷的自助服務管道。
4. 為銀行提供一套基礎的營運監控與審計工具。

## 第二章 系統設計

### 系統架構圖

本銀行系統採用分層架構設計，主要分為三個層次：

* **使用者介面層 (UI Layer)**：負責所有使用者介面的展示與使用者輸入的接收。
* **業務邏輯層 (BLL - Business Logic Layer)**：包含應用程式的核心業務規則和處理邏輯，負責處理 UI 層的請求並協調資料存取層的操作。
* **資料存取層 (DAL - Data Access Layer)**：負責與資料庫互動，執行資料的讀取、寫入、更新和刪除操作。

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述一張含有 文字, 圖表, 行, 字型 的圖片

自動產生的描述

### 功能模組說明

本系統依據功能需求劃分為以下主要模組：

1. **使用者認證模組**：處理登入、登出、角色驗證及權限導向。
2. **客戶帳戶管理模組**：包括帳戶概覽、存款、取款、新增客戶、刪除客戶、調整客戶餘額。
3. **銀行內部員工管理模組**：負責新增、刪除、修改職員及副行長帳號。
4. **銀行總覽與資產查詢模組**：顯示所有客戶帳戶列表、銀行營運資金及銀行總資產。
5. **銀行營運資金調整模組**：允許行長調整營運資金並進行雙重驗證。
6. **審計日誌查詢模組**：記錄並顯示所有敏感操作的詳細日誌。

### 資料庫設計

本系統採用 SQLite 作為資料儲存解決方案。資料庫設計主要包含以下實體：

* **Users (使用者)**：儲存所有系統用戶（客戶、職員、副行長、行長）的帳號、密碼雜湊值、姓名和角色等資訊。

一張含有 文字, 螢幕擷取畫面, 數字, 字型 的圖片

自動產生的描述

* **Accounts (帳戶)**：儲存客戶的銀行帳戶資訊，包含帳號、所屬客戶 ID 和當前餘額。一張含有 文字, 字型, 行, 螢幕擷取畫面 的圖片

  自動產生的描述
* **Transactions (交易)**：記錄所有客戶帳戶的金融交易，包含交易 ID、帳戶 ID、交易類型、金額、時間和描述。

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述

* **AuditLogs (審計日誌)**：記錄所有關鍵系統操作，包含日誌類型、操作者、操作時間、詳細描述和相關資訊。

一張含有 文字, 螢幕擷取畫面, 數字, 字型 的圖片

自動產生的描述

* **BankFunds (銀行營運資金)**：單一記錄，儲存銀行目前的營運資金金額。

一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述

## 第三章 系統實現

### 開發環境

* **作業系統**：Microsoft Windows 10 或更高版本
* **整合開發環境 (IDE)**：Microsoft Visual Studio (建議具體版本，例如 Visual Studio 2022)
* **程式語言**：C#
* **.NET 版本**：.NET Framework (建議具體版本，例如 4.8) 或 .NET Core (建議具體版本，例如 .NET 6)
* **資料庫**：SQLite (透過 ADO.NET 或 Entity Framework Core 進行互動)

### 核心功能代碼解析

本章將深入分析幾個核心功能的代碼實現，例如：

1. **使用者登入認證**：
   * 如何使用雜湊演算法（如 SHA256）儲存和驗證密碼。
   * 如何根據用戶角色導向不同主介面。
2. **客戶帳戶存款/取款**：
   * 如何實現金額有效性驗證和餘額檢查。
   * 如何原子性地更新帳戶餘額並記錄交易。
3. **審計日誌記錄**：
   * 如何在關鍵操作（如新增用戶、資金調整）發生時自動記錄日誌。
   * 日誌結構和儲存方式。

### 關鍵技術實現說明

1. **資料庫互動**：說明如何使用 ADO.NET 或 ORM (如 Dapper 或 Entity Framework Core) 與 SQLite 資料庫進行連接、查詢、插入、更新和刪除操作，特別強調參數化查詢以防止 SQL 注入。
2. **權限控制**：詳細解釋如何實作基於角色的存取控制 (RBAC)，確保不同角色只能存取其權限範圍內的功能。
3. **錯誤處理與日誌記錄**：說明如何建構健全的錯誤處理機制，捕獲和處理異常，並將重要的錯誤資訊記錄到系統日誌中。
4. **UI/UX 設計原則**：簡要介紹在介面設計中所採用的直觀、簡潔和即時回饋原則，以提升使用者體驗。

### 界面設計與使用者操作流程

* **登入介面**：清晰的帳號/密碼輸入框、登入按鈕，以及錯誤提示區塊。
* **客戶主介面**：顯示帳戶餘額、交易記錄列表，以及存款/取款按鈕。
* **職員/副行長/行長主介面**：根據角色顯示不同的功能選單或按鈕，例如客戶管理、員工管理、銀行總覽等。
* **操作流程**：例如「行長調整營運資金」的操作步驟（進入介面 -> 輸入金額 -> 雙重驗證 -> 輸入說明 -> 確認調整）。

## 第四章 測試與驗證

### 單元測試案例

針對業務邏輯層 (BLL) 和資料存取層 (DAL) 中的核心方法撰寫單元測試。例如：

* AuthenticationService.ValidateUser()：測試正確和錯誤的帳號密碼。
* AccountService.Deposit()：測試正數金額、負數金額存款。
* AccountService.Withdraw()：測試足夠餘額、餘額不足取款。
* UserManager.CreateClerk()：測試帳號唯一性。

### 系統整合測試

測試不同模組之間的協同工作，例如：

* **登入流程**：測試從 UI 輸入帳號密碼到成功登入並導向正確介面的完整流程。
* **新增客戶並存款**：測試職員新增客戶後，客戶登入並成功存款的流程。
* **權限測試**：測試不同角色是否能正確存取或被阻止存取特定功能。

### 效能評估（如有）

* **響應時間測試**：測量關鍵操作（如登入、查詢交易記錄、加載客戶列表）的響應時間，並與非功能性需求中的 NFR-1.1 和 NFR-1.2 進行比對。
* **資料庫操作效率**：評估在大量資料下，資料庫查詢和寫入操作的效率。

## 第五章 結論與未來工作

### 研究成果總結

本專題成功開發了一個基於 C# 和 SQLite 的桌面銀行系統應用程式，實現了多角色使用者認證、客戶帳戶管理、內部員工管理、銀行總資產查詢以及審計日誌追蹤等核心功能。系統設計採用分層架構，提高了程式碼的可維護性和可測試性。通過單元測試和整合測試，驗證了系統功能的正確性和穩定性。

### 專題限制

1. **SQLite 資料庫限制**：SQLite 屬於檔案型資料庫，不適用於高併發或分散式環境，限制了系統的擴展性。
2. **安全性未臻完善**：儘管實作了密碼雜湊，但目前未實作多因素認證 (MFA) 或入侵檢測系統 (IDS) 等進階安全機制。
3. **桌面應用限制**：作為桌面應用，系統的擴展性受限，不適合直接擴展為大型企業級或 Web 應用。
4. **即時性**：受限於桌面應用和 SQLite 特性，系統即時性可能無法達到超低延遲。

### 未來改進方向

1. **資料庫升級**：考慮將資料庫升級至支援高併發和分散式部署的關聯式資料庫（如 SQL Server、PostgreSQL 或 MySQL），以應對未來可能的擴展需求。
2. **增強安全性**：
   * 導入多因素認證 (MFA)，如簡訊驗證碼或生物辨識。
   * 實作更細粒度的存取控制和日誌監控，並整合入侵檢測機制。
   * 對敏感資料進行加密儲存。
3. **介面優化與使用者體驗提升**：
   * 採用更現代化的 UI 框架或設計模式，提升視覺美感和互動性。
   * 加入資料匯出（如 Excel、PDF）功能。
   * 優化報表生成和視覺化呈現，提供更直觀的數據分析。
4. **模組化與擴展性**：
   * 進一步將業務邏輯抽象為服務，方便未來功能擴展和單元測試。
   * 考慮微服務架構，若未來需要將部分功能雲端化或對外開放 API。
5. **性能優化**：對資料庫查詢進行索引優化，並對頻繁操作進行緩存處理，以提升系統在處理大量數據時的性能。
6. **自動化測試**：增加更多的自動化測試案例，包括 UI 自動化測試，以提升測試效率和系統品質。