## 會員系統實作報告

### 第一章 緒論

#### 研究背景與動機

本報告旨在詳盡闡述運用 C# 程式語言與 Windows Forms 框架建構會員管理系統之過程，並藉此展示基礎與進階程式設計概念之實際應用。當代應用程式之複雜度日益提升，對於高效能、安全且具備良好擴展性之用戶管理解決方案的需求亦隨之俱增。此類系統不僅對於確保資料之完整性、強化系統安全性至關重要，更為提供個人化使用者體驗奠定了堅實基礎。因此，本專案之動機，係緣於此一迫切需求，同時亦旨在示範於軟體開發領域中採行結構化程式設計典範之最佳實踐，以克服傳統開發方法所面臨之挑戰。

#### 研究目的

本研究之主要目標為建構一套功能完備之會員管理系統，其核心功能涵蓋用戶註冊、登入驗證、個人資料維護及管理員權限操作等關鍵環節。具體而言，此系統將實現安全的密碼儲存機制、精確的用戶身份驗證流程、彈性的基於角色之存取控制，以及響應式之使用者介面。此外，本專案亦著重於透過 Windows Forms 應用程式之開發，具體實踐物件導向程式設計（OOP）之核心原則，包括類別與物件之設計、繼承與介面之應用、方法過載與多型之展現、例外處理機制之建置、委派模式之運用以及多執行緒技術之整合。此舉旨在為學習者提供一個具體且實用之範例，以深入理解並掌握這些程式設計概念之實際應用。

#### 問題陳述

建構一套安全且高效之會員管理系統，需審慎考量多重面向之複雜性。首要之務在於確保資料之持久性與完整性，這要求採用可靠之資料儲存方案及嚴謹之資料驗證機制。其次，用戶身份之嚴謹驗證乃系統安全之基石，需有效防範未經授權之存取。再者，基於角色之存取控制機制，對於確保不同層級用戶僅能存取其權限範圍內之資源至關重要。同時，使用者介面之響應性亦為提升用戶體驗之關鍵要素，尤其於計算密集型操作期間，需確保介面不致凍結。其中所面臨之挑戰，包含密碼之安全管理（例如，如何有效防止暴力破解與字典攻擊）、防範常見之系統漏洞（如 SQL 注入、跨站腳本攻擊），以及確保於計算密集型操作期間仍能維持流暢之用戶體驗，避免因介面延遲而造成用戶之不滿。

#### 預期貢獻

本專案作為一份綜合性之教育資源，旨在闡明如何將多元程式設計概念有效整合於一連貫之應用系統中。其不僅為學習者提供一個實踐物件導向原則之平台，更為建構類似系統提供了一份清晰之藍圖。藉由運用設計模式與物件導向程式設計原則，本專案著重強調了系統之模組化、可維護性與可擴展性。此一藍圖可作為未來軟體開發專案之基礎，特別是針對需要用戶管理功能之應用程式。此外，本專案所展示之設計與實作原則，亦可為開發更為複雜之企業級應用程式提供寶貴之參考價值。

### 第二章 系統設計

#### 系統架構圖

本系統採行分層架構設計，其核心關注點區分為呈現層（表單）、業務邏輯層（模型、實用工具）及資料存取層（儲存庫）。此種分層設計之優勢在於提升系統之模組化程度、可維護性與可擴展性。各層之間透過明確定義之介面進行互動，有效降低了各層間之耦合度，從而使系統之各部分可獨立開發、測試及維護。

* **呈現層：** 由五個 Form.cs 檔案所構成：LoginForm、RegistrationForm、DashboardForm、ProfileUpdateForm 和 AdminPanelForm。此等表單負責處理使用者互動、接收用戶輸入，並將處理結果以視覺化方式呈現於使用者介面。
* **業務邏輯層：** 包含 Models 資料夾，其中之 Member.cs 檔案用於定義會員實體之資料結構與行為；以及 Utilities 資料夾，其內之 PasswordHasher.cs 檔案則專司密碼之安全處理，確保敏感資訊之加密與驗證。
* **資料存取層：** 則由 Repositories 資料夾進行管理，其內含 SQLiteMemberRepository.cs 檔案，該檔案實作了 IMemberRepository 介面，專責執行 SQLite 資料庫之相關操作，包括資料之增、刪、改、查，並將資料庫操作細節與業務邏輯層隔離。

#### 功能模組說明

* **登入模組（LoginForm.cs）：** 負責處理用戶身份驗證之核心流程。此模組接收用戶輸入之憑證，並透過與資料存取層之互動進行驗證。成功驗證後，系統將依據用戶之角色（普通用戶或管理員）導向至相應之儀表板介面。此模組整合了多執行緒技術以提升系統響應性，避免使用者介面在驗證過程中凍結，並運用委派機制實現模組間之通訊，確保登入成功後能通知主應用程式進行後續操作。
* **註冊模組（RegistrationForm.cs）：** 允許新用戶透過提供用戶名、密碼及電子郵件地址進行註冊。此模組具備嚴格之輸入驗證功能，以確保資料之有效性與完整性。同時，其與 PasswordHasher 模組緊密整合，確保密碼於儲存前經過安全哈希處理，有效降低資料洩漏之風險。
* **儀表板模組（DashboardForm.cs）：** 作為成功登入後之主要介面，此模組負責顯示用戶特定資訊，例如歡迎訊息及個人詳細資料。其設計考量了用戶角色之差異，並提供個人資料更新之存取權限；對於具備管理員權限之用戶，亦將動態顯示管理員面板之存取途徑，實現基於角色之介面呈現。
* **個人資料更新模組（ProfileUpdateForm.cs）：** 允許已登入會員更新其電子郵件地址及密碼。此模組包含資料驗證功能，以確保更新資訊之正確性。同時，其透過委派機制通知儀表板相關變更，確保使用者介面能即時反映最新之個人資料。
* **管理員面板模組（AdminPanelForm.cs）：** 此模組僅限具備管理員權限之用戶存取，其功能涵蓋會員資料之檢視、新增與刪除。會員資料將於 DataGridView 中進行清晰呈現，使管理員能夠便捷地執行用戶管理操作，例如移除違規用戶或新增特定權限之帳戶。

#### 資料庫設計

本系統採用輕量級之 SQLite 資料庫，其中包含一個名為 Members 之資料表。選擇 SQLite 之原因在於其無需獨立之伺服器進程，可直接嵌入應用程式中，極大地簡化了部署與管理。

* **資料表名稱：** Members
* **欄位：**
  + Id (INTEGER PRIMARY KEY AUTOINCREMENT)：作為會員之唯一識別符，自動遞增以確保每位會員皆擁有獨特之 ID，且為資料庫之主鍵，保障資料之完整性。
  + Username (TEXT UNIQUE NOT NULL)：儲存會員之用戶名，此欄位設定為唯一且不可為空，以確保每個用戶名之獨特性，防止重複註冊。
  + PasswordHash (TEXT NOT NULL)：儲存經過哈希處理之密碼。此欄位不可為空，且僅儲存密碼之哈希值而非明文，大幅提升了密碼安全性。
  + Email (TEXT)：儲存會員之電子郵件地址，此欄位允許為空，提供彈性。
  + Role (TEXT NOT NULL)：儲存會員之角色，例如「User」或「Admin」。此欄位不可為空，用於實現基於角色之存取控制及功能區分。

#### 類別圖、循序圖等 UML 圖表

**類別圖（概念性描述）：**

* **Member 類別：** 作為系統中所有會員實體之基礎抽象，其具備 Id、Username、PasswordHash、Email 及 Role 等核心屬性。該類別擁有多載之建構函數，提供多種初始化物件之方式，並包含一個虛擬的 DisplayInfo() 方法，旨在定義會員資訊之通用呈現方式，為子類別之特化提供基礎。
* **AdminMember 類別：** 繼承自 Member 類別，代表具備管理員權限之會員。其覆寫了 DisplayInfo() 方法，以提供管理員專屬之資訊呈現格式，例如額外標示其管理員身份，此乃多型之典型應用。
* MemberRole **列舉：** 定義了系統中所有可能之會員角色，目前包含 User 與 Admin 兩種，此舉有助於程式碼之清晰度與可維護性。
* IMemberRepository **介面：** 定義了資料庫操作之標準契約，其包含 GetMemberByUsername、GetAllMembers、AddMember、UpdateMember、DeleteMember、IsUsernameTaken 和 InitializeDatabase 等方法。此介面之存在，確保了資料存取層之抽象化，使得底層資料庫實作可隨時替換而無需修改上層業務邏輯。
* **SQLiteMemberRepository 類別：** 實作 IMemberRepository 介面，負責處理 SQLite 資料庫之具體互動。該類別內部整合了例外處理機制，以穩健地應對資料庫操作中可能發生之錯誤。
* **PasswordHasher 類別：** 此為一靜態工具類別，其提供運用 SHA256 演算法之 HashPassword 及 VerifyPassword 方法。此類別之設計旨在將密碼哈希邏輯封裝，確保密碼處理之安全性與一致性。
* **表單類別（LoginForm、RegistrationForm、DashboardForm、ProfileUpdateForm、AdminPanelForm）：** 各表單類別皆與 SQLiteMemberRepository 進行互動，以執行資料之存取與操作。同時，這些表單亦可能透過委派機制與其他表單進行事件驅動之通訊，實現模組間之鬆散耦合。

**循序圖（登入過程 - 高層次描述）：**

1. **使用者：** 於 LoginForm 之相應輸入欄位中鍵入其用戶名與密碼。
2. LoginForm**：** 使用者點擊 btnLogin 按鈕後，系統將立即禁用該按鈕以防止重複提交，並於使用者介面顯示載入指示器，提示使用者系統正在處理請求。
3. LoginForm**：** 啟動一個背景執行緒（透過 ThreadPool.QueueUserWorkItem 實作），將耗時之登入驗證程序卸載至此執行緒執行，以避免使用者介面主執行緒阻塞。
4. **背景執行緒：** 呼叫 \_memberRepository.GetMemberByUsername(username) 方法，向資料存取層請求檢索與所提供用戶名相符之會員資料。
5. **SQLiteMemberRepository：** 執行資料庫查詢操作，以檢索指定用戶名之會員記錄。
6. **背景執行緒：** 若成功檢索到會員資料，則進一步呼叫 PasswordHasher.VerifyPassword(password, member.PasswordHash) 方法，將用戶輸入之密碼進行哈希處理後與資料庫中儲存之哈希值進行比對，以驗證密碼之正確性。
7. **背景執行緒：** 若密碼驗證成功，則透過 LoginForm.Invoke 方法將控制權交還給使用者介面執行緒，以安全地更新使用者介面。此步驟對於跨執行緒操作至關重要，可避免潛在之執行緒安全問題。
8. LoginForm**（UI 執行緒）：** 顯示登入成功訊息，隨後觸發 OnLoginSuccess 事件，並隱藏當前 LoginForm。
9. **Program.cs：** 訂閱 OnLoginSuccess 事件，一旦接收到登入成功通知，隨即隱藏 LoginForm 並顯示 DashboardForm，完成應用程式介面之切換。
10. LoginForm**（UI 執行緒）：** 若登入程序失敗（例如，用戶名或密碼不正確），則顯示失敗訊息，並重新啟用相關按鈕，允許使用者再次嘗試登入。

### 第三章 系統實現

本章節將詳述系統之實現細節，包含開發環境之配置、核心功能之程式碼解析，以及關鍵技術之實作說明。

#### 開發環境

* **Visual Studio：** 本專案採用 Visual Studio 作為整合開發環境（IDE），其提供豐富之開發工具與調試功能，極大地提升了開發效率與程式碼品質。建議使用最新穩定版本，例如 Visual Studio 2022，以獲得最佳之開發體驗與功能支援。
* **.NET Framework：** 系統基於 .NET Framework 平台開發，該框架為 Windows 桌面應用程式提供了穩健之運行環境與豐富之類別庫。建議使用 .NET Framework 4.8 或更高版本，以確保應用程式之兼容性與效能。
* **NuGet 套件：** System.Data.SQLite 套件被選用作為資料庫連接之解決方案。此套件為 C# 應用程式提供了與 SQLite 資料庫互動之必要接口，其輕量級與嵌入式之特性，使其成為小型至中型應用程式之理想選擇。

#### 核心功能代碼解析

* **Member.cs（類別與物件、過載、多型）：**
  + 此檔案定義了 Member 類別，作為系統中所有會員實體之核心抽象。其具備 Id、Username、PasswordHash、Email 及 Role 等關鍵屬性，精確地描述了會員之基本資訊。
  + Member 類別擁有多個建構函數，此為方法過載（Overloading）之典型應用，允許開發者依據不同之初始參數集來建立 Member 物件，提供了極大之彈性。例如，可以僅提供用戶名、密碼和電子郵件來創建新會員，或從資料庫讀取時包含 ID。
  + virtual DisplayInfo() 方法的設計，實現了多型（Polymorphism）特性。此方法定義了會員資訊之通用呈現方式，並允許其子類別（如 AdminMember）覆寫此行為，以提供更為特化之資訊呈現。
  + AdminMember 類別繼承自 Member 類別，此舉明確展示了繼承（Inheritance）之概念。AdminMember 不僅繼承了 Member 的所有屬性與方法，更覆寫了 DisplayInfo() 方法，以在輸出中明確標示其管理員身份，進一步彰顯了多型之應用。
* **IMemberRepository.cs（介面）：**
  + 此檔案定義了 IMemberRepository 介面，其建立了資料存取操作之標準契約。此介面之存在，旨在確保程式碼之一致性，並啟用依賴反轉原則（Dependency Inversion Principle），使得業務邏輯層無需直接依賴於具體之資料庫實作。此設計模式極大地提升了系統之可測試性與可擴展性。
* **SQLiteMemberRepository.cs（類別、實作介面、例外處理、多型）：**
  + 此類別實作了 IMemberRepository 介面，提供了與 SQLite 資料庫互動之具體方法。其內部包含 InitializeDatabase() 方法，該方法負責在資料庫檔案及結構尚未存在時進行自動建立，確保系統首次運行時之順暢。
  + 在資料庫操作中（例如，AddMember 捕捉唯一約束違規之情形，其錯誤碼為 19），廣泛運用 try-catch 區塊以實現強健之例外處理機制。此舉可有效捕捉並處理資料庫操作中可能發生之各種錯誤，提升系統之穩定性與容錯能力。
  + 在 GetMemberByUsername 和 GetAllMembers 方法中，程式碼根據資料庫中之 Role 欄位值，動態實例化 Member 或 AdminMember 物件，此亦為多型特性之體現，確保了資料讀取之靈活性與正確性。
* **PasswordHasher.cs（實用工具）：**
  + 此為一靜態類別，其提供運用 SHA256 演算法之 HashPassword 及 VerifyPassword 方法。HashPassword 方法將明文密碼轉換為不可逆之哈希值，以確保密碼於儲存時之安全性。VerifyPassword 方法則用於驗證用戶輸入之密碼是否與儲存之哈希值匹配，而無需直接處理明文密碼。此類別之設計旨在將密碼哈希邏輯封裝，確保密碼處理之安全性與一致性，有效防範密碼洩漏風險。

#### 關鍵技術實現說明

* **委派與事件（LoginForm、RegistrationForm、ProfileUpdateForm、Program.cs）：**
  + 委派（Delegates），如 LoginSuccessEventHandler 及 RegistrationSuccessEventHandler，定義了事件之簽名，即事件處理方法所需之參數類型與返回類型。
  + 事件（Events），如 OnLoginSuccess、OnRegistrationSuccess 及 OnProfileUpdated，則基於委派實現，促進了表單間之鬆散耦合通訊。此機制使得一個表單能夠在特定操作完成後，通知其他表單進行相應之處理（例如，登入成功後通知主程式顯示儀表板），而無需直接依賴於具體之表單實例，極大地提升了程式碼之模組化與可維護性。
* **執行緒（LoginForm）：**
  + ThreadPool.QueueUserWorkItem 於 btnLogin\_Click 事件中被運用，此乃多執行緒技術之關鍵應用。其目的在於將潛在耗時之登入驗證程序卸載至背景執行緒執行，從而避免使用者介面（UI）主執行緒因長時間阻塞而導致應用程式介面凍結，顯著提升了用戶體驗。
  + this.Invoke((MethodInvoker)delegate { ... }); 對於從非使用者介面執行緒安全地更新使用者介面元素至關重要。由於 Windows Forms 控制項並非執行緒安全，直接從背景執行緒操作使用者介面元素將引發跨執行緒操作異常。Invoke 方法確保所有使用者介面更新操作皆於主使用者介面執行緒上執行，從而有效預防此類問題，保障應用程式之穩定性。

#### 界面設計與使用者操作流程

1. **應用程式啟動：** 系統啟動時，Program.cs 作為應用程式之入口點，負責初始化並啟動 LoginForm。此表單將作為使用者與系統互動之首個介面。
2. **登入/註冊：**
   * 使用者可於 LoginForm 中之指定輸入欄位鍵入其用戶名與密碼，以嘗試登入。系統將提供視覺回饋，例如載入指示器，以告知使用者登入請求正在處理中。
   * 抑或，使用者亦可點擊「註冊」按鈕，此操作將開啟 RegistrationForm。該表單引導使用者完成新帳戶之創建，包括用戶名、密碼及電子郵件之輸入。
   * 成功註冊後，RegistrationForm 將自動關閉，並將新註冊之用戶名自動填入 LoginForm 之用戶名欄位，隨後重新顯示 LoginForm，便捷使用者進行後續登入。
3. **儀表板存取：**
   * 成功登入後，LoginForm 將自動隱藏，並顯示 DashboardForm。
   * DashboardForm 將動態呈現用戶專屬之資訊，例如歡迎訊息及個人詳細資料。此處運用多型特性，透過呼叫 \_loggedInMember.DisplayInfo() 方法，依據登入會員之實際類型（普通會員或管理員）顯示不同之資訊格式。
   * 若登入用戶具備管理員權限，系統將條件性地顯示「管理員面板」按鈕，提供管理員功能之存取途徑，實現基於角色之介面呈現。
4. **個人資料管理：**
   * 使用者可自 DashboardForm 點擊「更新個人資料」選項，此操作將開啟 ProfileUpdateForm。
   * ProfileUpdateForm 允許用戶更新其電子郵件地址及更改密碼。此模組包含嚴格之資料驗證功能，以確保更新資訊之正確性與安全性。同時，其透過委派機制於成功更新後通知 DashboardForm，確保使用者介面能即時反映最新之個人資料，提供流暢之用戶體驗。
5. **管理員功能：**
   * 若登入用戶具備管理員權限，其可自 DashboardForm 存取 AdminPanelForm。
   * AdminPanelForm 於 DataGridView 中清晰呈現所有會員之詳細資料，使管理員能夠便捷地執行用戶管理操作，包括刷新會員列表、刪除選定之會員，以及新增會員（此功能重用了 RegistrationForm 之邏輯）。此模組為系統管理提供了必要之工具。
6. **登出：**
   * 點擊 DashboardForm 上之「登出」按鈕將執行登出操作。此操作將關閉當前表單，並重新開啟 LoginForm，使應用程式返回至初始登入介面，完成用戶會話之結束。

### 第四章 測試與驗證

本章節將闡述系統之測試與驗證策略，涵蓋單元測試案例、系統整合測試及效能評估，以確保系統之功能性、穩定性與效能。

#### 單元測試案例

* **Member 類別：**
  + 測試建構函數之過載，以確保 Id、Username、PasswordHash、Email 及 Role 等屬性於不同初始化情境下均能正確賦值。
  + 測試 Member 與 AdminMember 之 DisplayInfo() 方法，以驗證多型特性之正確實作，即 AdminMember 應顯示其特有之管理員標識。
* **PasswordHasher 類別：**
  + 測試 HashPassword() 方法，以確保其對於相同輸入能產生一致且不可逆之哈希值。應測試不同長度與內容之密碼。
  + 測試 VerifyPassword() 方法，透過使用正確與不正確之密碼與儲存之哈希值進行比對，驗證其布林返回值之正確性。
* **SQLiteMemberRepository 類別：**
  + 測試 InitializeDatabase() 方法，以確認資料庫檔案及 Members 資料表之成功建立，並驗證其結構與預期相符。
  + 測試 AddMember() 方法，以驗證成功插入新會員資料之能力，並特別測試處理唯一用戶名約束（例如，當嘗試新增重複用戶名時，應捕捉到 SQLiteException 錯誤碼 19）之情境。
  + 測試 GetMemberByUsername() 方法，以檢驗其對於現有及不存在用戶之處理能力，並根據資料庫中之 Role 欄位正確返回 Member 或 AdminMember 實例。
  + 測試 GetAllMembers() 方法，以確保其能準確檢索所有會員資料，並正確實例化 Member 或 AdminMember 物件。
  + 測試 UpdateMember() 方法，以驗證會員資料（如電子郵件、密碼哈希）更改後之持久性。
  + 測試 DeleteMember() 方法，以驗證成功刪除指定會員之功能，並確認其不再出現於會員列表中。
  + 測試 IsUsernameTaken() 方法，以檢驗其在用戶名已被佔用與未被佔用兩種情況下之布林返回值之正確性。
  + 測試各種錯誤條件（例如，資料庫連接中斷、非法 SQL 語句）下之所有方法，並驗證其例外處理機制是否能穩健地捕捉並報告錯誤。

#### 系統整合測試

* **登入流程：** 驗證輸入正確憑證後，系統是否能根據用戶角色（普通用戶或管理員）正確導向至相應之儀表板介面。同時，需測試輸入不正確之憑證、空欄位輸入及並發登入嘗試之情境，並驗證系統之錯誤提示與處理機制。
* **註冊流程：** 測試成功註冊之情境，包括用戶名、密碼及電子郵件之有效性驗證。此外，需測試使用現有用戶名進行註冊、使用不匹配之密碼或強度不足之密碼進行註冊之情境，並驗證系統之錯誤提示與拒絕註冊行為。
* **個人資料更新流程：** 測試更新電子郵件地址、更改密碼，以及密碼確認不匹配時之行為。驗證相關更改是否已正確反映於儀表板上，並確保密碼更新後仍能正常登入。
* **管理員面板流程：** 測試會員資料之載入、選定會員之刪除，以及從管理員面板新增會員之功能。驗證資料庫之變更是否已正確反映於介面中，並測試管理員操作之權限限制。
* **表單導航：** 確保 LoginForm、RegistrationForm、DashboardForm、ProfileUpdateForm 及 AdminPanelForm 之間之轉換流暢無礙，無介面卡頓或異常關閉之情形。
* **執行緒安全：** 驗證於背景登入操作期間，使用者介面是否能維持響應性，避免「無響應」狀態。同時，確認來自背景執行緒之使用者介面更新是否透過 Invoke 方法正確處理，以防止跨執行緒操作所引發之錯誤。

#### 效能評估

* **登入/註冊響應時間：** 此項評估旨在測量登入與註冊程序從用戶提交請求至系統響應所需之時間。尤其著重於模擬網路延遲（例如於 LoginForm 中運用 Thread.Sleep 實作）之情境下，對其響應時間進行量化評估。此數據有助於識別潛在之效能瓶頸。
* **資料庫操作速度：** 此項評估旨在衡量 AddMember、GetMemberByUsername、GetAllMembers、UpdateMember 及 DeleteMember 等核心資料庫操作之執行速度。特別考量於資料庫中會員數量持續增加之情境，對其查詢與操作效率進行基準測試，以評估其在不同負載下之表現。
* **使用者介面響應性：** 於大量資料庫操作或背景任務執行期間，監控使用者介面之響應性，包括介面刷新率、按鈕點擊響應延遲等指標。此項評估旨在確認多執行緒技術能有效防止使用者介面凍結，確保應用程式提供流暢之用戶體驗。

### 第五章 結論與未來工作

#### 研究成果總結

本專案成功展示了運用 C# 程式語言、Windows Forms 框架及 SQLite 資料庫實作一套全面性會員管理系統之能力。其有效地整合了物件導向程式設計（OOP）之核心概念，例如類別與物件（Member、AdminMember），清晰地定義了系統中實體之結構與行為；繼承（AdminMember 繼承自 Member 類別），實現了程式碼之重用與擴展；以及介面（IMemberRepository 由 SQLiteMemberRepository 實作），確保了資料存取層之抽象化與模組化。諸如方法過載（體現於 Member 建構函數之多樣性）與多型（展現於 DisplayInfo() 方法及 Member 物件建立過程）等關鍵進階概念，亦於本專案中獲得清晰之演示，彰顯了物件導向設計之彈性與強大。此外，在資料存取層及用戶輸入驗證環節，皆廣泛應用了強健之例外處理機制，顯著提升了系統之穩定性與容錯能力。委派之運用促進了表單間鬆散耦合之事件驅動通訊，有效降低了模組間之依賴性，而多執行緒技術則確保了使用者介面於背景操作期間仍能維持響應性，顯著提升了用戶體驗。

#### 專題限制

* **安全性：** 儘管已實作密碼哈希機制以保護用戶憑證，然仍可考量進一步之安全強化措施。例如，引入加鹽處理（Salting）以防範彩虹表攻擊，並採用更為強健之金鑰派生函數（Key Derivation Functions），諸如 PBKDF2 或 Argon2，以增加破解密碼之難度。
* **使用者介面/使用者體驗（UI/UX）：** 本系統之使用者介面功能完備，然其設計相對基礎。未來可應用更為進階之使用者介面/使用者體驗（UI/UX）設計原則，例如採用現代化之視覺風格、動畫效果及更直觀之導航流程，以實現更為精緻之視覺呈現及提升整體可用性。
* **可擴展性：** SQLite 資料庫因其輕量級與嵌入式之特性，適用於中小型應用程式之需求。然而，對於大規模部署或高並發存取之情境，其性能可能受限。此時，則需採用更為強大之關聯式資料庫系統（例如，SQL Server、PostgreSQL）或非關聯式資料庫（NoSQL）；儘管如此，IMemberRepository 介面之設計特性仍允許便捷地替換底層資料庫，無需大幅修改業務邏輯層。
* **錯誤報告：** 儘管已實作例外處理機制以捕捉運行時錯誤，然仍可進一步實作更為詳盡之日誌記錄功能，將系統運行狀態、錯誤訊息及用戶操作記錄至日誌檔案中，以便於問題追蹤與分析。同時，提供對用戶更為友善之錯誤訊息，避免直接顯示技術性錯誤，提升用戶體驗。
* **管理員功能：** AdminPanelForm 提供了基本之增刪改查（CRUD）操作。未來可增補更為複雜之管理功能，例如用戶搜尋與篩選、精細化之角色管理（允許自定義角色及權限）、審計日誌（記錄管理員操作）以及用戶活動監控等，以滿足更為複雜之管理需求。

#### 未來改進方向

* **安全性強化：** 建議實作密碼加鹽機制，並採用更為強健之哈希演算法，如 bcrypt 或 scrypt，以提升密碼儲存之安全性。此外，可考慮導入雙因素身份驗證（Two-Factor Authentication, 2FA）機制，為用戶帳戶提供額外之安全層級。
* **進階使用者介面框架：** 建議探索運用 C# 之現代使用者介面框架，例如 Windows Presentation Foundation (WPF)、Avalonia UI 或 .NET Multi-platform App UI (MAUI)，以期實現更為豐富之介面設計、更佳之視覺效果及提升跨平台兼容性，從而擴大應用程式之潛在用戶群。
* **非同步操作：** 建議擴展 async/await 語法之應用範圍，使其涵蓋所有潛在之阻塞性輸入/輸出（I/O）操作，例如資料庫查詢、網路請求等。此舉將確保應用程式於執行期間，使用者介面能持續維持響應性，提供更流暢之用戶體驗。
* **日誌記錄：** 建議整合專業日誌框架，例如 NLog 或 Serilog，以實現更為完善之錯誤追蹤及應用程式監控功能。透過結構化日誌記錄，可更有效地分析系統行為、診斷問題並進行效能優化。
* **依賴注入：** 建議實作依賴注入（Dependency Injection, DI）容器，例如 Autofac 或 Microsoft.Extensions.DependencyInjection，以更有效地管理依賴項（例如，IMemberRepository 之實例）。此舉將進一步提升程式碼之可測試性、模組化程度及可維護性。
* **單元測試框架：** 建議整合專用之單元測試框架，例如 NUnit 或 xUnit，以實現更具結構性及自動化之測試流程。透過編寫全面的單元測試，可確保程式碼之品質與穩定性，並於開發過程中及早發現並修正錯誤。
* **基於角色之存取控制（RBAC）：** 建議實作更為細緻之權限管理機制，允許系統管理員定義多種角色，並為每個角色分配特定之操作權限，以區分「用戶」與「管理員」之外之不同層級存取權限。
* **使用者體驗（UX）增強：** 建議增補諸如「忘記密碼」流程、自動「記住我」功能、以及即時輸入驗證（Real-time Input Validation）等功能，以提升整體使用者體驗，使系統更具人性化與便捷性。

### 附錄. 參考文獻