這是一個實現會員系統的教學，包含至少五個 `Form.cs` 檔案，並融合了類別與物件、繼承與介面、過載與多型、例外處理、委派與執行緒等概念。

-----

### \*\*專案設定 (Project Setup)\*\*

1. \*\*建立新的專案:\*\*

\* 在 Visual Studio 中，選擇 `建立新的專案` (Create a new project)。

\* 搜尋並選擇 `Windows Forms App (.NET Framework)`。

\* 命名您的專案 (例如：`MemberSystemApp`)。

2. \*\*安裝 SQLite 套件:\*\*

\* 在 Visual Studio 中，點擊 `工具` (Tools) -\> `NuGet 封裝管理員` (NuGet Package Manager) -\> `管理方案的 NuGet 封裝` (Manage NuGet Packages for Solution)。

\* 在 `瀏覽` (Browse) 頁籤中，搜尋 `System.Data.SQLite`。

\* 選擇 `System.Data.SQLite` 並點擊 `安裝` (Install)。

-----

### \*\*資料庫設計 (Database Design)\*\*

我們將建立一個簡單的 `Members` 表，包含以下欄位：

\* `Id` (INTEGER PRIMARY KEY AUTOINCREMENT)

\* `Username` (TEXT UNIQUE NOT NULL)

\* `PasswordHash` (TEXT NOT NULL) - 儲存密碼的哈希值

\* `Email` (TEXT)

\* `Role` (TEXT NOT NULL) - 例如：`User` 或 `Admin`

-----

### \*\*核心類別設計 (Core Class Design)\*\*

我們將定義幾個核心類別來管理會員資料和業務邏輯。

#### 1\. `Models` 資料夾

新增一個名為 `Models` 的資料夾，並在其中建立 `Member.cs` 檔案。

\*\*`Member.cs` (類別與物件)\*\*

```csharp

using System;

namespace MemberSystemApp.Models

{

// Member 類別：代表系統中的一個會員實體

public class Member

{

public int Id { get; set; }

public string Username { get; set; }

public string PasswordHash { get; set; } // 儲存加密後的密碼

public string Email { get; set; }

public MemberRole Role { get; set; } // 使用 MemberRole 列舉

// 建構函數過載 (Overloading)：多型的一種形式

// 預設建構函數

public Member() { }

// 常用建構函數

public Member(string username, string passwordHash, string email, MemberRole role)

{

Username = username;

PasswordHash = passwordHash;

Email = email;

Role = role;

}

// 包含 ID 的建構函數 (用於從資料庫讀取)

public Member(int id, string username, string passwordHash, string email, MemberRole role)

{

Id = id;

Username = username;

PasswordHash = passwordHash;

Email = email;

Role = role;

}

// 顯示會員資訊的方法 (多型：如果子類別覆寫，行為會不同)

public virtual string DisplayInfo()

{

return $"ID: {Id}, 用戶名: {Username}, 角色: {Role}, 信箱: {Email}";

}

}

// 會員角色列舉

public enum MemberRole

{

User,

Admin

}

// AdminMember 類別：繼承自 Member (繼承)

public class AdminMember : Member

{

public AdminMember() : base() { }

public AdminMember(string username, string passwordHash, string email)

: base(username, passwordHash, email, MemberRole.Admin) { }

public AdminMember(int id, string username, string passwordHash, string email)

: base(id, username, passwordHash, email, MemberRole.Admin) { }

// 多型 (Polymorphism)：覆寫父類別的方法，實現不同的行為

public override string DisplayInfo()

{

return $"ID: {Id}, 用戶名: {Username} (管理員), 信箱: {Email}";

}

}

}

```

#### 2\. `Interfaces` 資料夾

新增一個名為 `Interfaces` 的資料夾，並在其中建立 `IMemberRepository.cs` 檔案。

\*\*`IMemberRepository.cs` (介面)\*\*

```csharp

using System.Collections.Generic;

using MemberSystemApp.Models;

namespace MemberSystemApp.Interfaces

{

// IMemberRepository 介面：定義了對會員資料操作的標準合約

public interface IMemberRepository

{

Member GetMemberByUsername(string username);

List<Member> GetAllMembers();

void AddMember(Member member);

void UpdateMember(Member member);

void DeleteMember(int memberId);

bool IsUsernameTaken(string username);

void InitializeDatabase(); // 初始化資料庫的方法

}

}

```

#### 3\. `Repositories` 資料夾

新增一個名為 `Repositories` 的資料夾，並在其中建立 `SQLiteMemberRepository.cs` 檔案。

\*\*`SQLiteMemberRepository.cs` (類別、實作介面、例外處理)\*\*

```csharp

using System;

using System.Collections.Generic;

using System.Data.SQLite;

using System.IO;

using MemberSystemApp.Interfaces;

using MemberSystemApp.Models;

namespace MemberSystemApp.Repositories

{

// SQLiteMemberRepository 類別：實作 IMemberRepository 介面，處理 SQLite 資料庫操作

public class SQLiteMemberRepository : IMemberRepository

{

private readonly string \_connectionString;

private readonly string \_databasePath;

public SQLiteMemberRepository(string databaseFileName = "members.db")

{

\_databasePath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, databaseFileName);

\_connectionString = $"Data Source={\_databasePath};Version=3;";

InitializeDatabase();

}

// 初始化資料庫和資料表

public void InitializeDatabase()

{

// 如果資料庫檔案不存在，則建立它

if (!File.Exists(\_databasePath))

{

SQLiteConnection.CreateFile(\_databasePath);

}

using (var connection = new SQLiteConnection(\_connectionString))

{

try

{

connection.Open();

string createTableSql = @"

CREATE TABLE IF NOT EXISTS Members (

Id INTEGER PRIMARY KEY AUTOINCREMENT,

Username TEXT UNIQUE NOT NULL,

PasswordHash TEXT NOT NULL,

Email TEXT,

Role TEXT NOT NULL

);";

using (var command = new SQLiteCommand(createTableSql, connection))

{

command.ExecuteNonQuery();

}

}

// 例外處理 (Exception Handling)：捕捉資料庫操作中可能發生的錯誤

catch (SQLiteException ex)

{

Console.WriteLine($"資料庫初始化錯誤: {ex.Message}");

// 實際應用中可能需要記錄日誌或顯示錯誤訊息給使用者

}

}

}

public Member GetMemberByUsername(string username)

{

using (var connection = new SQLiteConnection(\_connectionString))

{

try

{

connection.Open();

string sql = "SELECT \* FROM Members WHERE Username = @Username";

using (var command = new SQLiteCommand(sql, connection))

{

command.Parameters.AddWithValue("@Username", username);

using (var reader = command.ExecuteReader())

{

if (reader.Read())

{

// 多型應用：根據 Role 判斷建立 Member 或 AdminMember 物件

MemberRole role = (MemberRole)Enum.Parse(typeof(MemberRole), reader["Role"].ToString());

if (role == MemberRole.Admin)

{

return new AdminMember(

Convert.ToInt32(reader["Id"]),

reader["Username"].ToString(),

reader["PasswordHash"].ToString(),

reader["Email"].ToString()

);

}

else

{

return new Member(

Convert.ToInt32(reader["Id"]),

reader["Username"].ToString(),

reader["PasswordHash"].ToString(),

reader["Email"].ToString(),

role

);

}

}

}

}

}

catch (SQLiteException ex)

{

Console.WriteLine($"獲取會員錯誤: {ex.Message}");

}

}

return null;

}

public List<Member> GetAllMembers()

{

List<Member> members = new List<Member>();

using (var connection = new SQLiteConnection(\_connectionString))

{

try

{

connection.Open();

string sql = "SELECT \* FROM Members";

using (var command = new SQLiteCommand(sql, connection))

{

using (var reader = command.ExecuteReader())

{

while (reader.Read())

{

MemberRole role = (MemberRole)Enum.Parse(typeof(MemberRole), reader["Role"].ToString());

if (role == MemberRole.Admin)

{

members.Add(new AdminMember(

Convert.ToInt32(reader["Id"]),

reader["Username"].ToString(),

reader["PasswordHash"].ToString(),

reader["Email"].ToString()

));

}

else

{

members.Add(new Member(

Convert.ToInt32(reader["Id"]),

reader["PasswordHash"].ToString(),

reader["Username"].ToString(),

reader["Email"].ToString(),

role

));

}

}

}

}

}

catch (SQLiteException ex)

{

Console.WriteLine($"獲取所有會員錯誤: {ex.Message}");

}

}

return members;

}

public void AddMember(Member member)

{

using (var connection = new SQLiteConnection(\_connectionString))

{

try

{

connection.Open();

string sql = "INSERT INTO Members (Username, PasswordHash, Email, Role) VALUES (@Username, @PasswordHash, @Email, @Role)";

using (var command = new SQLiteCommand(sql, connection))

{

command.Parameters.AddWithValue("@Username", member.Username);

command.Parameters.AddWithValue("@PasswordHash", member.PasswordHash);

command.Parameters.AddWithValue("@Email", member.Email);

command.Parameters.AddWithValue("@Role", member.Role.ToString());

command.ExecuteNonQuery();

}

}

catch (SQLiteException ex)

{

if (ex.ErrorCode == 19) // SQLite UNIQUE constraint failed error code

{

throw new Exception("該用戶名已被註冊。");

}

else

{

throw new Exception($"新增會員錯誤: {ex.Message}");

}

}

}

}

public void UpdateMember(Member member)

{

using (var connection = new SQLiteConnection(\_connectionString))

{

try

{

connection.Open();

string sql = "UPDATE Members SET Username = @Username, PasswordHash = @PasswordHash, Email = @Email, Role = @Role WHERE Id = @Id";

using (var command = new SQLiteCommand(sql, connection))

{

command.Parameters.AddWithValue("@Username", member.Username);

command.Parameters.AddWithValue("@PasswordHash", member.PasswordHash);

command.Parameters.AddWithValue("@Email", member.Email);

command.Parameters.AddWithValue("@Role", member.Role.ToString());

command.Parameters.AddWithValue("@Id", member.Id);

command.ExecuteNonQuery();

}

}

catch (SQLiteException ex)

{

throw new Exception($"更新會員錯誤: {ex.Message}");

}

}

}

public void DeleteMember(int memberId)

{

using (var connection = new SQLiteConnection(\_connectionString))

{

try

{

connection.Open();

string sql = "DELETE FROM Members WHERE Id = @Id";

using (var command = new SQLiteCommand(sql, connection))

{

command.Parameters.AddWithValue("@Id", memberId);

command.ExecuteNonQuery();

}

}

catch (SQLiteException ex)

{

throw new Exception($"刪除會員錯誤: {ex.Message}");

}

}

}

public bool IsUsernameTaken(string username)

{

using (var connection = new SQLiteConnection(\_connectionString))

{

try

{

connection.Open();

string sql = "SELECT COUNT(\*) FROM Members WHERE Username = @Username";

using (var command = new SQLiteCommand(sql, connection))

{

command.Parameters.AddWithValue("@Username", username);

int count = Convert.ToInt32(command.ExecuteScalar());

return count > 0;

}

}

catch (SQLiteException ex)

{

Console.WriteLine($"檢查用戶名是否重複錯誤: {ex.Message}");

return false; // 發生錯誤時視為不重複，或者根據業務邏輯決定

}

}

}

}

}

```

#### 4\. `Utilities` 資料夾

新增一個名為 `Utilities` 的資料夾，並在其中建立 `PasswordHasher.cs` 檔案。

\*\*`PasswordHasher.cs` (密碼哈希處理)\*\*

```csharp

using System.Security.Cryptography;

using System.Text;

namespace MemberSystemApp.Utilities

{

// 用於密碼哈希處理的靜態類別

public static class PasswordHasher

{

public static string HashPassword(string password)

{

using (SHA256 sha256Hash = SHA256.Create())

{

// 從字串計算哈希

byte[] bytes = sha256Hash.ComputeHash(Encoding.UTF8.GetBytes(password));

// 將位元組陣列轉換為十六進制字串

StringBuilder builder = new StringBuilder();

for (int i = 0; i < bytes.Length; i++)

{

builder.Append(bytes[i].ToString("x2"));

}

return builder.ToString();

}

}

public static bool VerifyPassword(string password, string storedHash)

{

string hashOfInput = HashPassword(password);

return hashOfInput == storedHash;

}

}

}

```

-----

### \*\*表單設計 (Form Design)\*\*

現在我們來設計五個主要的表單。

#### 1\. `LoginForm.cs` (登入表單)

這是應用程式的啟動表單。

\* \*\*設計介面:\*\*

\* `TextBox` 用於輸入用戶名 (`txtUsername`)

\* `TextBox` 用於輸入密碼 (`txtPassword`) - 將 `UseSystemPasswordChar` 屬性設為 `true`

\* `Button` 用於登入 (`btnLogin`)

\* `Button` 用於註冊 (`btnRegister`)

\*\*`LoginForm.cs` 程式碼 (委派、例外處理、執行緒)\*\*

```csharp

using System;

using System.Threading;

using System.Windows.Forms;

using MemberSystemApp.Models;

using MemberSystemApp.Repositories;

using MemberSystemApp.Utilities;

namespace MemberSystemApp

{

public partial class LoginForm : Form

{

private readonly SQLiteMemberRepository \_memberRepository;

// 定義一個委派，用於在登入成功後通知主程式 (例如：打開其他表單)

public delegate void LoginSuccessEventHandler(Member loggedInMember);

public event LoginSuccessEventHandler OnLoginSuccess;

public LoginForm()

{

InitializeComponent();

\_memberRepository = new SQLiteMemberRepository();

// 確保資料庫在應用程式啟動時就被初始化

\_memberRepository.InitializeDatabase();

// 讓密碼輸入框顯示星號

txtPassword.UseSystemPasswordChar = true;

}

private void btnLogin\_Click(object sender, EventArgs e)

{

string username = txtUsername.Text.Trim();

string password = txtPassword.Text;

if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))

{

MessageBox.Show("請輸入用戶名和密碼。", "登入錯誤", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

// 使用執行緒 (Thread) 來模擬網路請求或耗時的登入驗證

// 避免 UI 凍結 (委派用於跨執行緒更新 UI)

btnLogin.Enabled = false; // 禁用按鈕防止重複點擊

ShowLoadingIndicator(true); // 顯示載入指示器

ThreadPool.QueueUserWorkItem((state) =>

{

Member member = null;

string errorMessage = string.Empty;

try

{

// 模擬耗時操作

Thread.Sleep(1500);

member = \_memberRepository.GetMemberByUsername(username);

if (member != null && PasswordHasher.VerifyPassword(password, member.PasswordHash))

{

// 登入成功

// 使用 Invoke 確保在 UI 執行緒上執行 UI 更新操作

this.Invoke((MethodInvoker)delegate

{

MessageBox.Show($"歡迎回來, {member.Username}!", "登入成功", MessageBoxButtons.OK, MessageBoxIcon.Information);

OnLoginSuccess?.Invoke(member); // 觸發登入成功事件

this.Hide(); // 隱藏登入表單

});

}

else

{

errorMessage = "用戶名或密碼不正確。";

}

}

catch (Exception ex) // 處理資料庫或其他潛在錯誤

{

errorMessage = $"登入時發生錯誤: {ex.Message}";

}

finally

{

// 無論成功或失敗，都恢復按鈕狀態並隱藏載入指示器

this.Invoke((MethodInvoker)delegate

{

ShowLoadingIndicator(false);

btnLogin.Enabled = true;

if (!string.IsNullOrEmpty(errorMessage))

{

MessageBox.Show(errorMessage, "登入失敗", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

});

}

});

}

private void btnRegister\_Click(object sender, EventArgs e)

{

RegistrationForm registerForm = new RegistrationForm(\_memberRepository);

// 註冊成功後，自動填入用戶名到登入表單

registerForm.OnRegistrationSuccess += (registeredUsername) =>

{

txtUsername.Text = registeredUsername;

txtPassword.Clear();

this.Show(); // 顯示登入表單

};

this.Hide(); // 隱藏登入表單

registerForm.Show();

}

// 簡化版的載入指示器（實際應用中可使用 ProgressBar 或 Label 顯示 "載入中..."）

private void ShowLoadingIndicator(bool show)

{

// 例如：可以添加一個 Label 或 ProgressBar 來顯示載入狀態

// For simplicity, we'll just disable/enable controls

txtUsername.Enabled = !show;

txtPassword.Enabled = !show;

btnRegister.Enabled = !show;

// 可以在此處添加一個 Label 顯示 "登入中..."

// lblLoading.Visible = show;

}

private void LoginForm\_FormClosing(object sender, FormClosingEventArgs e)

{

// 當登入表單關閉時，確保應用程式退出

Application.Exit();

}

}

}

```

#### 2\. `RegistrationForm.cs` (註冊表單)

\* \*\*設計介面:\*\*

\* `TextBox` 用於輸入用戶名 (`txtUsername`)

\* `TextBox` 用於輸入密碼 (`txtPassword`)

\* `TextBox` 用於確認密碼 (`txtConfirmPassword`)

\* `TextBox` 用於輸入信箱 (`txtEmail`)

\* `Button` 用於註冊 (`btnRegister`)

\* `Button` 用於返回 (`btnBack`)

\*\*`RegistrationForm.cs` 程式碼 (例外處理)\*\*

```csharp

using System;

using System.Windows.Forms;

using MemberSystemApp.Models;

using MemberSystemApp.Repositories;

using MemberSystemApp.Utilities;

namespace MemberSystemApp

{

public partial class RegistrationForm : Form

{

private readonly SQLiteMemberRepository \_memberRepository;

// 定義一個委派，用於在註冊成功後通知登入表單

public delegate void RegistrationSuccessEventHandler(string registeredUsername);

public event RegistrationSuccessEventHandler OnRegistrationSuccess;

public RegistrationForm(SQLiteMemberRepository memberRepository)

{

InitializeComponent();

\_memberRepository = memberRepository;

txtPassword.UseSystemPasswordChar = true;

txtConfirmPassword.UseSystemPasswordChar = true;

}

private void btnRegister\_Click(object sender, EventArgs e)

{

string username = txtUsername.Text.Trim();

string password = txtPassword.Text;

string confirmPassword = txtConfirmPassword.Text;

string email = txtEmail.Text.Trim();

// 輸入驗證

if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password) || string.IsNullOrEmpty(confirmPassword))

{

MessageBox.Show("用戶名、密碼和確認密碼為必填項。", "註冊錯誤", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

if (password != confirmPassword)

{

MessageBox.Show("密碼和確認密碼不匹配。", "註冊錯誤", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

// 簡單的密碼複雜度檢查

if (password.Length < 6)

{

MessageBox.Show("密碼長度至少為6個字元。", "註冊錯誤", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

try

{

if (\_memberRepository.IsUsernameTaken(username))

{

MessageBox.Show("該用戶名已被使用，請選擇其他用戶名。", "註冊失敗", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

string hashedPassword = PasswordHasher.HashPassword(password);

Member newMember = new Member(username, hashedPassword, email, MemberRole.User);

\_memberRepository.AddMember(newMember);

MessageBox.Show("註冊成功！您現在可以登入。", "註冊成功", MessageBoxButtons.OK, MessageBoxIcon.Information);

OnRegistrationSuccess?.Invoke(username); // 觸發註冊成功事件

this.Close(); // 關閉註冊表單

}

// 例外處理 (Exception Handling)：捕捉 AddMember 可能拋出的自訂錯誤

catch (Exception ex)

{

MessageBox.Show($"註冊失敗: {ex.Message}", "註冊錯誤", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void btnBack\_Click(object sender, EventArgs e)

{

this.Close(); // 關閉註冊表單，返回到登入表單

}

private void RegistrationForm\_FormClosed(object sender, FormClosedEventArgs e)

{

// 當註冊表單關閉時，如果它沒有被其他事件處理程序重新顯示，確保登入表單被重新顯示

// 這確保了從註冊返回後，登入表單會再次出現

if (Application.OpenForms["LoginForm"] != null)

{

Application.OpenForms["LoginForm"].Show();

}

}

}

}

```

#### 3\. `DashboardForm.cs` (會員主頁/儀表板)

這是會員登入成功後看到的主要介面，根據會員角色顯示不同的資訊。

\* \*\*設計介面:\*\*

\* `Label` 顯示歡迎訊息 (`lblWelcome`)

\* `Label` 顯示用戶資訊 (`lblUserInfo`)

\* `Button` 用於登出 (`btnLogout`)

\* `Button` 用於更新個人資料 (`btnUpdateProfile`)

\* `Button` 用於管理員功能 (如果用戶是管理員則顯示) (`btnAdminPanel`)

\*\*`DashboardForm.cs` 程式碼 (多型)\*\*

```csharp

using System;

using System.Windows.Forms;

using MemberSystemApp.Models;

using MemberSystemApp.Repositories;

namespace MemberSystemApp

{

public partial class DashboardForm : Form

{

private Member \_loggedInMember;

private readonly SQLiteMemberRepository \_memberRepository;

public DashboardForm(Member loggedInMember)

{

InitializeComponent();

\_loggedInMember = loggedInMember;

\_memberRepository = new SQLiteMemberRepository();

DisplayMemberInfo();

}

private void DisplayMemberInfo()

{

lblWelcome.Text = $"歡迎回來, {\_loggedInMember.Username}!";

// 多型應用：調用 DisplayInfo() 方法，根據 \_loggedInMember 的實際類型（Member 或 AdminMember）

// 會執行不同的實作

lblUserInfo.Text = \_loggedInMember.DisplayInfo();

// 根據角色顯示或隱藏管理員按鈕

btnAdminPanel.Visible = (\_loggedInMember.Role == MemberRole.Admin);

}

private void btnLogout\_Click(object sender, EventArgs e)

{

MessageBox.Show("您已成功登出。", "登出", MessageBoxButtons.OK, MessageBoxIcon.Information);

// 隱藏當前表單，並顯示登入表單

this.Close();

LoginForm loginForm = new LoginForm();

loginForm.OnLoginSuccess += (member) =>

{

DashboardForm newDashboard = new DashboardForm(member);

newDashboard.Show();

};

loginForm.Show();

}

private void btnUpdateProfile\_Click(object sender, EventArgs e)

{

// 打開更新個人資料表單

ProfileUpdateForm profileUpdateForm = new ProfileUpdateForm(\_loggedInMember, \_memberRepository);

profileUpdateForm.OnProfileUpdated += (updatedMember) =>

{

\_loggedInMember = updatedMember; // 更新當前登入會員的資料

DisplayMemberInfo(); // 重新顯示更新後的資訊

};

profileUpdateForm.ShowDialog(); // 使用 ShowDialog 讓用戶必須先處理此表單

}

private void btnAdminPanel\_Click(object sender, EventArgs e)

{

// 打開管理員面板表單 (只有管理員才能看到和點擊)

AdminPanelForm adminPanelForm = new AdminPanelForm(\_memberRepository);

adminPanelForm.ShowDialog();

}

private void DashboardForm\_FormClosing(object sender, FormClosingEventArgs e)

{

// 當主面板關閉時，如果沒有其他表單被顯示，則退出應用程式

// 這可以防止在登出後關閉 Dashboard 時應用程式仍然運行

if (Application.OpenForms.Count == 1 && Application.OpenForms[0] is DashboardForm)

{

Application.Exit();

}

}

}

}

```

#### 4\. `ProfileUpdateForm.cs` (個人資料更新表單)

允許會員更新他們的 Email 和密碼。

\* \*\*設計介面:\*\*

\* `Label` 顯示用戶名 (`lblUsername`)

\* `TextBox` 用於輸入新的 Email (`txtEmail`)

\* `TextBox` 用於輸入新密碼 (`txtNewPassword`)

\* `TextBox` 用於確認新密碼 (`txtConfirmNewPassword`)

\* `Button` 用於儲存變更 (`btnSave`)

\* `Button` 用於取消 (`btnCancel`)

\*\*`ProfileUpdateForm.cs` 程式碼 (例外處理、委派)\*\*

```csharp

using System;

using System.Windows.Forms;

using MemberSystemApp.Models;

using MemberSystemApp.Repositories;

using MemberSystemApp.Utilities;

namespace MemberSystemApp

{

public partial class ProfileUpdateForm : Form

{

private Member \_currentMember;

private readonly SQLiteMemberRepository \_memberRepository;

// 定義一個委派，用於在資料更新後通知 DashboardForm

public delegate void ProfileUpdatedEventHandler(Member updatedMember);

public event ProfileUpdatedEventHandler OnProfileUpdated;

public ProfileUpdateForm(Member member, SQLiteMemberRepository memberRepository)

{

InitializeComponent();

\_currentMember = member;

\_memberRepository = memberRepository;

lblUsername.Text = $"用戶名: {\_currentMember.Username}";

txtEmail.Text = \_currentMember.Email;

txtNewPassword.UseSystemPasswordChar = true;

txtConfirmNewPassword.UseSystemPasswordChar = true;

}

private void btnSave\_Click(object sender, EventArgs e)

{

string newEmail = txtEmail.Text.Trim();

string newPassword = txtNewPassword.Text;

string confirmNewPassword = txtConfirmNewPassword.Text;

// 驗證密碼是否匹配

if (!string.IsNullOrEmpty(newPassword) && newPassword != confirmNewPassword)

{

MessageBox.Show("新密碼和確認密碼不匹配。", "更新錯誤", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

try

{

\_currentMember.Email = newEmail; // 更新 Email

if (!string.IsNullOrEmpty(newPassword))

{

// 檢查密碼長度

if (newPassword.Length < 6)

{

MessageBox.Show("新密碼長度至少為6個字元。", "更新錯誤", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

\_currentMember.PasswordHash = PasswordHasher.HashPassword(newPassword); // 更新哈希後的密碼

}

\_memberRepository.UpdateMember(\_currentMember);

MessageBox.Show("個人資料更新成功！", "更新成功", MessageBoxButtons.OK, MessageBoxIcon.Information);

OnProfileUpdated?.Invoke(\_currentMember); // 觸發事件，通知 DashboardForm

this.Close();

}

// 例外處理 (Exception Handling)：捕捉資料庫操作中可能發生的錯誤

catch (Exception ex)

{

MessageBox.Show($"更新失敗: {ex.Message}", "更新錯誤", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void btnCancel\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

```

#### 5\. `AdminPanelForm.cs` (管理員面板)

這個表單只有管理員才能訪問，用於管理系統中的所有會員。

\* \*\*設計介面:\*\*

\* `DataGridView` 顯示會員列表 (`dgvMembers`)

\* `Button` 用於刷新列表 (`btnRefresh`)

\* `Button` 用於刪除選定的會員 (`btnDeleteMember`)

\* `Button` 用於新增會員 (簡化起見，可以重用註冊表單或有單獨的簡易新增) (`btnAddMember`) - 我們這裡簡單實現為一個按鈕，實際可以打開一個新表單

\* `Button` 用於返回 (`btnBack`)

\*\*`AdminPanelForm.cs` 程式碼 (例外處理、物件操作)\*\*

```csharp

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows.Forms;

using MemberSystemApp.Models;

using MemberSystemApp.Repositories;

namespace MemberSystemApp

{

public partial class AdminPanelForm : Form

{

private readonly SQLiteMemberRepository \_memberRepository;

public AdminPanelForm(SQLiteMemberRepository memberRepository)

{

InitializeComponent();

\_memberRepository = memberRepository;

LoadMembers();

}

private void LoadMembers()

{

try

{

List<Member> members = \_memberRepository.GetAllMembers();

// 為了安全，不直接在 UI 中顯示 PasswordHash

dgvMembers.DataSource = members.Select(m => new

{

m.Id,

m.Username,

m.Email,

m.Role

}).ToList();

}

// 例外處理 (Exception Handling)：捕捉資料庫操作中可能發生的錯誤

catch (Exception ex)

{

MessageBox.Show($"載入會員列表失敗: {ex.Message}", "錯誤", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void btnRefresh\_Click(object sender, EventArgs e)

{

LoadMembers();

}

private void btnDeleteMember\_Click(object sender, EventArgs e)

{

if (dgvMembers.SelectedRows.Count > 0)

{

DialogResult confirm = MessageBox.Show("確定要刪除選定的會員嗎？", "確認刪除", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (confirm == DialogResult.Yes)

{

try

{

// 獲取選定行的 Id

int memberIdToDelete = (int)dgvMembers.SelectedRows[0].Cells["Id"].Value;

\_memberRepository.DeleteMember(memberIdToDelete);

MessageBox.Show("會員已成功刪除。", "刪除成功", MessageBoxButtons.OK, MessageBoxIcon.Information);

LoadMembers(); // 重新載入列表

}

catch (Exception ex)

{

MessageBox.Show($"刪除會員失敗: {ex.Message}", "錯誤", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

else

{

MessageBox.Show("請選擇一個要刪除的會員。", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

private void btnAddMember\_Click(object sender, EventArgs e)

{

// 這裡可以打開一個新的表單來添加會員，例如重用 RegistrationForm

RegistrationForm addMemberForm = new RegistrationForm(\_memberRepository);

addMemberForm.Text = "新增會員"; // 修改表單標題

addMemberForm.OnRegistrationSuccess += (username) =>

{

MessageBox.Show($"新會員 {username} 已新增！", "新增成功", MessageBoxButtons.OK, MessageBoxIcon.Information);

LoadMembers(); // 新增成功後刷新列表

};

addMemberForm.ShowDialog();

}

private void btnBack\_Click(object sender, EventArgs e)

{

this.Close(); // 關閉管理員面板

}

}

}

```

-----

### \*\*`Program.cs` (應用程式入口)\*\*

最後，我們需要修改 `Program.cs` 來啟動我們的 `LoginForm`。

```csharp

using System;

using System.Windows.Forms;

using MemberSystemApp.Models;

namespace MemberSystemApp

{

internal static class Program

{

/// <summary>

/// 應用程式的主要進入點。

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

// 創建登入表單

LoginForm loginForm = new LoginForm();

// 訂閱登入成功事件

loginForm.OnLoginSuccess += (loggedInMember) =>

{

// 當登入成功時，隱藏登入表單並顯示儀表板

if (loginForm.InvokeRequired)

{

loginForm.Invoke((MethodInvoker)delegate { loginForm.Hide(); });

}

else

{

loginForm.Hide();

}

DashboardForm dashboardForm = new DashboardForm(loggedInMember);

dashboardForm.Show();

};

Application.Run(loginForm); // 啟動應用程式並顯示登入表單

}

}

}

```

-----

### \*\*如何運行和測試\*\*

1. \*\*在 Visual Studio 中打開專案。\*\*

2. \*\*確保已安裝 `System.Data.SQLite` NuGet 套件。\*\*

3. \*\*構建專案 (Build Solution)。\*\*

4. \*\*運行專案 (Start Debugging 或 F5)。\*\*

\*\*測試步驟：\*\*

1. \*\*第一次運行時，資料庫 `members.db` 將會自動在您的專案 `bin/Debug` (或 `bin/Release`) 目錄下建立。\*\*

2. \*\*點擊 `註冊` 按鈕。\*\*

3. \*\*註冊一個普通會員\*\*：

\* 用戶名：`testuser`

\* 密碼：`password123`

\* Email：`test@example.com`

\* 角色將預設為 `User`。

4. \*\*註冊一個管理員會員\*\* (您可以手動修改 `SQLiteMemberRepository` 的 `AddMember` 方法，暫時將 `new Member` 的 `MemberRole.User` 改為 `MemberRole.Admin` 讓您可以創建一個管理員。或者更簡單的做法是，註冊後手動用 SQLite Browser 修改資料庫中的 `Role` 欄位為 `Admin`)。為了這個範例，我們假設你已經在註冊後將某個用戶的 `Role` 設為 `Admin`。

\* 用戶名：`admin`

\* 密碼：`adminpass`

\* Email：`admin@example.com`

\* \*\*手動將這個用戶的 `Role` 欄位在 `members.db` 中修改為 `Admin`。\*\* (您可以使用 [DB Browser for SQLite](https://sqlitebrowser.org/) 來編輯 `.db` 檔案)。

5. \*\*返回登入頁面。\*\*

6. \*\*嘗試使用 `testuser` 登入。\*\* 登入後應看到普通用戶的儀表板。

7. \*\*登出。\*\*

8. \*\*嘗試使用 `admin` 登入。\*\* 登入後應看到管理員的儀表板，並有 `管理員面板` 按鈕。

9. \*\*點擊 `管理員面板`。\*\* 您應該可以看到會員列表，並可以嘗試刪除會員。

-----

### \*\*程式碼說明與標註\*\*

我已經在程式碼中透過註解 `//` 和 `/\* ... \*/` 進行了詳細的標註，說明了每個物件導向和進階概念的應用位置。

\*\*核心概念應用點概述：\*\*

\* \*\*類別與物件 (Class and Object):\*\*

\* `Member`、`AdminMember`：定義會員的資料結構和行為。

\* `SQLiteMemberRepository`：定義資料庫操作的行為。

\* 各個 `Form` 類別：定義使用者介面的結構和互動邏輯。

\* 在 `Main` 函數和各個表單中，通過 `new` 關鍵字建立這些類別的實例（物件）。

\* \*\*繼承 (Inheritance):\*\*

\* `AdminMember` 繼承自 `Member`：`AdminMember` 自動擁有 `Member` 的所有屬性和方法，並可擴展或修改其行為。

\* \*\*介面 (Interface):\*\*

\* `IMemberRepository`：定義了會員資料操作的標準契約。

\* `SQLiteMemberRepository` 實作了 `IMemberRepository`：這表示 `SQLiteMemberRepository` 必須提供介面中定義的所有方法的實作，保證了程式碼的規範性和可擴展性。

\* \*\*過載 (Overloading):\*\*

\* `Member` 類別中定義了多個建構函數，它們具有相同的名稱但參數列表不同，這是典型的建構函數過載。

\* \*\*多型 (Polymorphism):\*\*

\* \*\*方法的覆寫 (Overriding):\*\* `AdminMember` 覆寫了 `Member` 的 `DisplayInfo()` 方法，使得當您有一個 `Member` 類型的引用指向一個 `AdminMember` 實例時，調用 `DisplayInfo()` 會執行 `AdminMember` 中的版本。在 `DashboardForm` 的 `DisplayMemberInfo` 方法中有所體現。

\* \*\*介面多型 (Interface Polymorphism):\*\* 在 `DashboardForm` 和其他地方，您可以傳遞一個 `IMemberRepository` 類型的引用，而它實際指向的是 `SQLiteMemberRepository` 的實例。這使得將來可以輕鬆替換為其他資料庫實現（例如 `MsSqlMemberRepository`）而無需修改調用程式碼。

\* \*\*例外處理 (Exception Handling):\*\*

\* 大量使用 `try-catch` 區塊來捕捉並處理可能發生的執行期錯誤，特別是在資料庫操作（`SQLiteMemberRepository`）和用戶輸入驗證（`RegistrationForm`、`ProfileUpdateForm`）中，提高了程式的健壯性。

\* \*\*委派 (Delegates):\*\*

\* `LoginSuccessEventHandler` 和 `RegistrationSuccessEventHandler`：定義了事件的簽名。

\* `OnLoginSuccess` 和 `OnRegistrationSuccess`：事件本身，允許其他類別（例如 `Program.cs` 和 `LoginForm`）訂閱並響應這些事件，實現表單之間的鬆散耦合通信。

\* `MethodInvoker`：在跨執行緒操作 UI 時，使用 `this.Invoke((MethodInvoker)delegate { ... });` 來確保 UI 更新操作在主 UI 執行緒上執行，防止發生跨執行緒操作異常。

\* \*\*執行緒 (Threads):\*\*

\* 在 `LoginForm` 的 `btnLogin\_Click` 事件中，使用 `ThreadPool.QueueUserWorkItem` 將耗時的登入驗證操作放入一個背景執行緒中執行。

\* 這樣做可以避免在執行耗時操作時主 UI 執行緒被阻塞，從而保持使用者介面的響應性，提供更好的用戶體驗。

這個專案結構和程式碼應該能很好地幫助您理解和應用這些重要的 C\# 和 OOP 概念。如果您在實現過程中遇到任何問題，或者需要對某個部分進行更深入的解釋，請隨時提出！