企業碳排放 ESG 碳足跡管理網站

宏碁專班 期末成果發表

發表日 : 2025/10/31

專題成員 : 徐秉群 Jimmy 羽蓁 Vincent

宏碁專班 第二組

講師 簡志聰 老師

1. 摘要

本專題開發「企業碳排放 ESG 碳足跡管理網站」，旨在協助企業有效追蹤、管理和分析其碳排放數據，並符合 **ESG (環境、社會、公司治理)** 的相關規範。預計可協助企業減少 30% 的碳排放量。

網站提供以下關鍵功能：

- 碳足跡盤查 **(Carbon Footprint Assessment)**：協助企業收集和計算各項營運活動的碳排放量。
- 數據視覺化 **(Data Visualization)**：以圖表方式呈現碳排放數據，幫助企業快速掌握碳排放熱點。
- **ESG** 報告生成 **(ESG Reporting)**：自動生成符合 GRI、SASB 等標準的 ESG 報告，提升企業透明度。

專案採用 **ASP.NET MVC** 框架開發，使用 **C#** 程式語言，並整合 **iTextSharp (PDF** 報告**)** 等第三方函式庫和 API。開發過程中，我們特別注重使用者體驗 **(User Experience)** 和系統安全性 **(System Security)**。

關鍵詞：碳排放、ESG、碳足跡、碳盤查、 數據視覺化、報告生成、ASP.NET MVC、iTextSharp、使用者體驗、系統安全性

2. 目錄

3. 前言

3.1 專案背景

全球暖化和氣候變遷已成為全球共同面臨的嚴峻挑戰。隨著社會對環境保護意識的提高，企業的環境責任日益受到重視。近年來，**ESG**（環境、社會、公司治理）已成為評估企業可持續發展能力的重要指標，投資者與利害關係人皆將其視為企業競爭力的核心依據。企業不僅需要關注自身的經濟效益，還需要積極管理其環境影響，減少碳排放，並公開透明地披露相關資訊。然而，現有的碳排放管理工具或方法往往存在成本高昂、操作複雜、功能有限等問題，難以滿足企業日益增長的需求。因此，市場亟需一套具成本效益、操作簡便、並能整合 **ESG** 要求的碳排放管理解決方案，以協助企業邁向低碳轉型與永續經營。

3.2 專案動機

為了解決現有碳排放管理工具或方法的不足之處，並提供更有效率的解決方案，我們開發了「企業碳排放 **ESG** 碳足跡管理網站」。本專案旨在提供一個易於使用、功能完善、且經濟實惠的平台，協助企業追蹤、管理和分析其碳排放數據，並符合 **ESG** 相關規範。我們相信，透過本專案，企業可以更有效地了解其碳排放熱點，制定減碳策略，並提升 ESG 績效，最終實現可持續發展。

3.3 專案目標

本專案的主要目標如下：

1. 開發功能完善的碳排放管理系統
   提供碳足跡盤查、數據視覺化、ESG 報告自動生成等多元功能，支援企業全面掌握環境績效。
2. 打造使用者友善的操作介面
   以簡潔直觀的設計降低使用門檻，讓非技術背景的企業用戶亦能輕鬆上手。
3. 強化系統安全與穩定性
   確保企業資料在傳輸與儲存過程中的機密性與完整性，符合資安標準。
4. 協助企業實現減碳與永續目標
   透過數據分析與策略建議，協助企業達成碳排放量降低 30%，並持續提升 ESG 綜合績效。

4. 系統分析與設計

4.1 需求分析

4.1.1 使用者需求

- 目標使用者：本網站的主要目標使用者為企業的 ESG 負責人及碳排放管理人員。
- 使用者情境：ESG 負責人可以使用網站追蹤企業的碳排放數據，生成 ESG 報告，並向投資者和利益相關者展示企業的 ESG 績效。碳排放管理人員可以使用網站進行碳足跡盤查，分析碳排放熱點，並制定減碳策略。
- 使用者期望：使用者期望網站提供易於使用的介面和操作流程，能夠快速準確地完成碳排放數據的收集、管理和分析，並生成符合國際標準的 ESG 報告。

4.1.2 功能需求

- 碳足跡盤查：
    - 輸入：使用者輸入各項營運活動的數據，例如能源消耗、物料使用、運輸里程等。
    - 處理：系統根據預設的碳排放係數，計算各項營運活動的碳排放量。
    - 輸出：系統顯示各項營運活動的碳排放量，並生成碳足跡報告。
- 數據視覺化：
    - 輸入：使用者選擇要呈現的碳排放數據，例如總碳排放量、各項營運活動的碳排放量等。
    - 處理：系統根據使用者選擇的數據，生成折線圖、柱狀圖、圓餅圖等圖表。
    - 輸出：系統顯示生成的圖表，並提供下載功能。
- **ESG** 報告生成：
    - 輸入：使用者選擇要生成的 ESG 報告類型，例如 GRI、SASB 等。
    - 處理：系統根據使用者選擇的報告類型，從資料庫中提取相關數據，並生成符合標準格式的報告。
    - 輸出：系統顯示生成的報告，並提供下載功能。

4.2 系統架構設計

本系統採用 **ASP.NET Core MVC (Model-View-Controller)** 架構模式，這是一種經典且廣泛使用的 Web 應用程式架構。

- 程式語言：C#
- 資料庫：SQL Server
- **ORM**：Entity Framework Core (EF Core)

MVC 架構將應用程式劃分為三個主要部分：

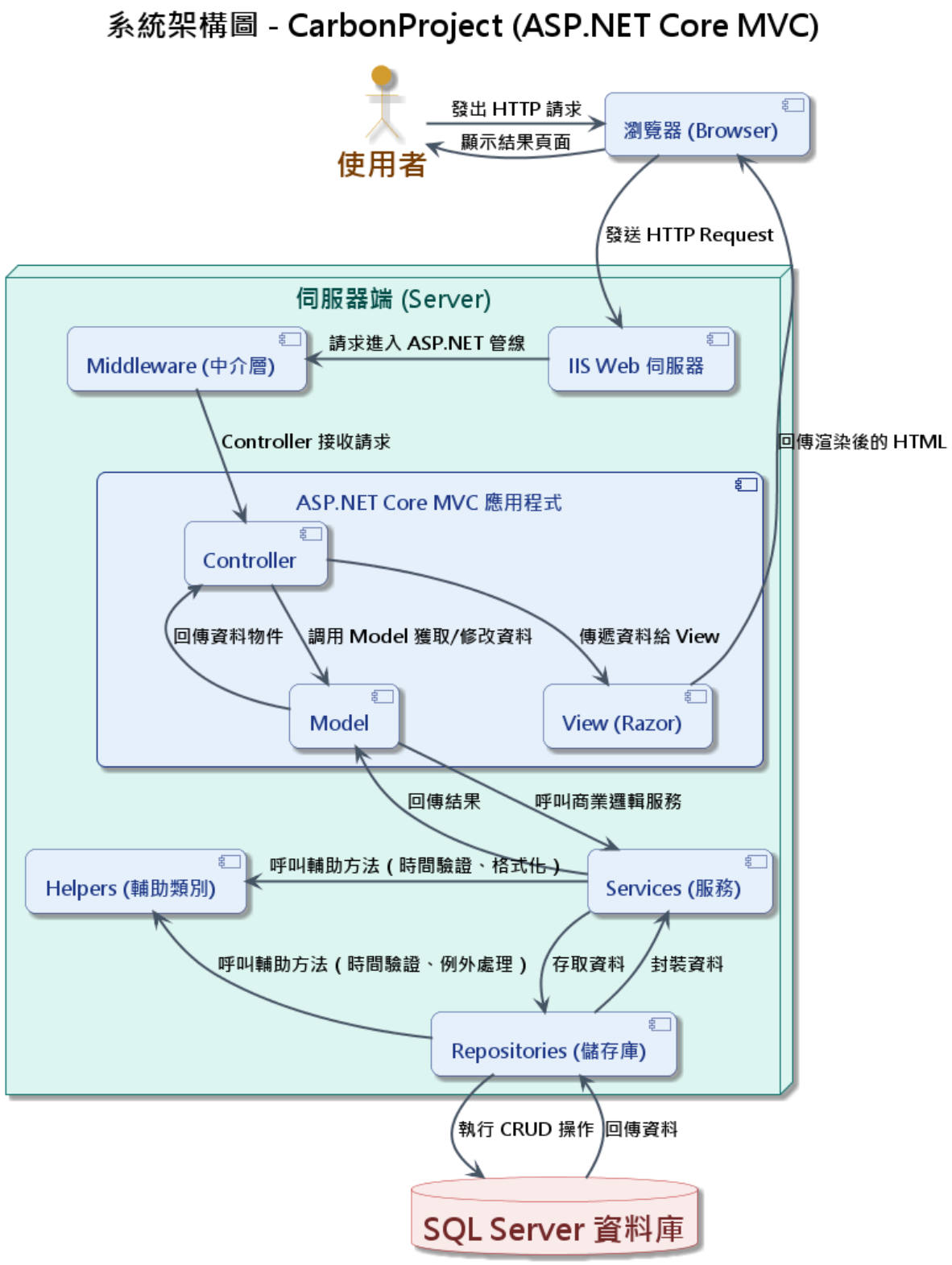
- **Model (模型)**：負責處理資料邏輯，代表應用程式的數據和業務規則。*Model* 通常是 *C#* 類別，用於封裝數據和操作數據的方法。
- **View (視圖)**：負責呈現使用者介面，將模型數據以視覺化的方式展示給使用者。View 通常是 Razor 視圖 (.cshtml 檔案)，使用 HTML 和 C# 程式碼來生成動態網頁內容。
- **Controller (控制器)**：負責接收使用者的請求，協調 Model 和 View 的互動，並將結果返回給使用者。Controller 是 C# 類別，包含 Action 方法，用於處理特定的請求。

為了進一步提高系統的可維護性、可測試性和可擴展性，本系統還整合了以下元件：

- **Services (服務)**：負責封裝應用程式的業務邏輯，例如數據驗證、數據轉換、API 呼叫等。Services 通常是 C# 介面和類別，透過依賴注入 (Dependency Injection) 注入到 Controller 或其他 Services 中。
- **Repositories (儲存庫)**：負責封裝資料存取邏輯，將應用程式與底層資料庫隔離開。Repositories 負責執行資料庫的 CRUD (Create, Read, Update, Delete) 操作。
- **Middleware (中介層)**：負責處理 HTTP 請求管道中的橫切關注點 (Cross-Cutting Concerns)，例如身份驗證、授權、錯誤處理、日誌記錄等。
- **Helpers (輔助類別)**：負責提供常用的輔助方法，例如字串處理、日期格式化、檔案操作等。Helpers 通常是靜態類別，包含靜態方法，可以在應用程式的任何地方使用。
- **Data (數據)**：指的是系統中所有需要儲存、處理和呈現的資訊。Data 可以來自多個來源，例如使用者輸入、資料庫、外部 API、檔案等。Data 在系統中流動，並被各個元件使用和處理。

這種整合的架構設計有助於我們構建一個結構良好、易於維護和擴展的應用程式，並有效地管理和利用系統中的 Data。

4.2.1 系統架構圖


系統架構圖 - CarbonProject (ASP.NET Core MVC)
使用者
發出 HTTP 請求
顯示結果頁面
瀏覽器 (Browser)
發送 HTTP Request
伺服器端 (Server)
Middleware (中介層)
請求進入 ASP.NET 管線
IIS Web 伺服器
Controller 接收請求
回傳渲染後的 HTML
ASP.NET Core MVC 應用程式
Controller
回傳資料物件
調用 Model 獲取/修改資料
傳遞資料給 View
Model
View (Razor)
回傳結果
呼叫商業邏輯服務
Helpers (輔助類別)
呼叫輔助方法（時間驗證、格式化）
Services (服務)
呼叫輔助方法（時間驗證、例外處理）
存取資料
封裝資料
Repositories (儲存庫)
執行 CRUD 操作
回傳資料
SQL Server 資料庫

4.2.2 模組設計

本系統主要由以下模組組成：

Model1. 帳戶管理模組 (Account Management Module)

- 職責：負責處理使用者身份驗證和授權相關的功能。
- 主要元件：
    - **Controllers\AccountController.cs**：
      處理使用者註冊、登入、登出、密碼管理等 HTTP 請求。
    - **Models\RegisterViewModel.cs**：
      定義使用者註冊時所需的數據模型。
    - **Models\LoginViewModel.cs**：
      定義使用者登入時所需的數據模型。
    - **Views\Account** 目錄：
      包含與帳戶管理相關的視圖，例如註冊、登入等。
- 功能：
    - 使用者註冊：允許新使用者創建帳戶。
    - 使用者登入：驗證使用者身份並允許其訪問系統。
    - 使用者登出：終止使用者的訪問。
- 輸入：
    - 使用者註冊資訊 (例如：Email、密碼)
    - 使用者登入資訊 (例如：Email、密碼)
- 輸出：
    - 註冊成功/失敗訊息
    - 登入成功/失敗訊息
    - 密碼重置成功/失敗訊息

Model2. 首頁模組 (Home Module)

- 職責：負責處理首頁相關的功能，例如顯示歡迎訊息、提供導航等。
- 主要元件：
    - **Controllers\HomeController.cs**：處理首頁相關的 HTTP 請求。
    - **Views\Home** 目錄：包含首頁的視圖，例如 Index.cshtml。
- 功能：
    - 顯示首頁內容：呈現網站的首頁內容。
    - 提供導航：提供網站的導航連結。
- 輸入：無 (或少量的配置數據)
- 輸出：網站首頁的 HTML 內容

Model3. 錯誤處理模組 (Error Handling Module)

- 職責：負責處理應用程式中發生的錯誤。
- 主要元件：
  - **Controllers\HomeController.cs**：包含 Error Action，用於顯示錯誤頁面。
  - **Views\Shared\Error.cshtml**：顯示錯誤訊息的視圖。
- 功能：
  - 顯示錯誤訊息：向使用者顯示友好的錯誤訊息。
  - 記錄錯誤資訊：將錯誤資訊記錄到日誌中，以便進行診斷。
- 輸入：
  - 錯誤訊息
  - 異常物件
- 輸出：
  - 包含錯誤訊息的 HTML 頁面

Model4. 碳足跡管理模組 (Carbon Footprint Management Module)

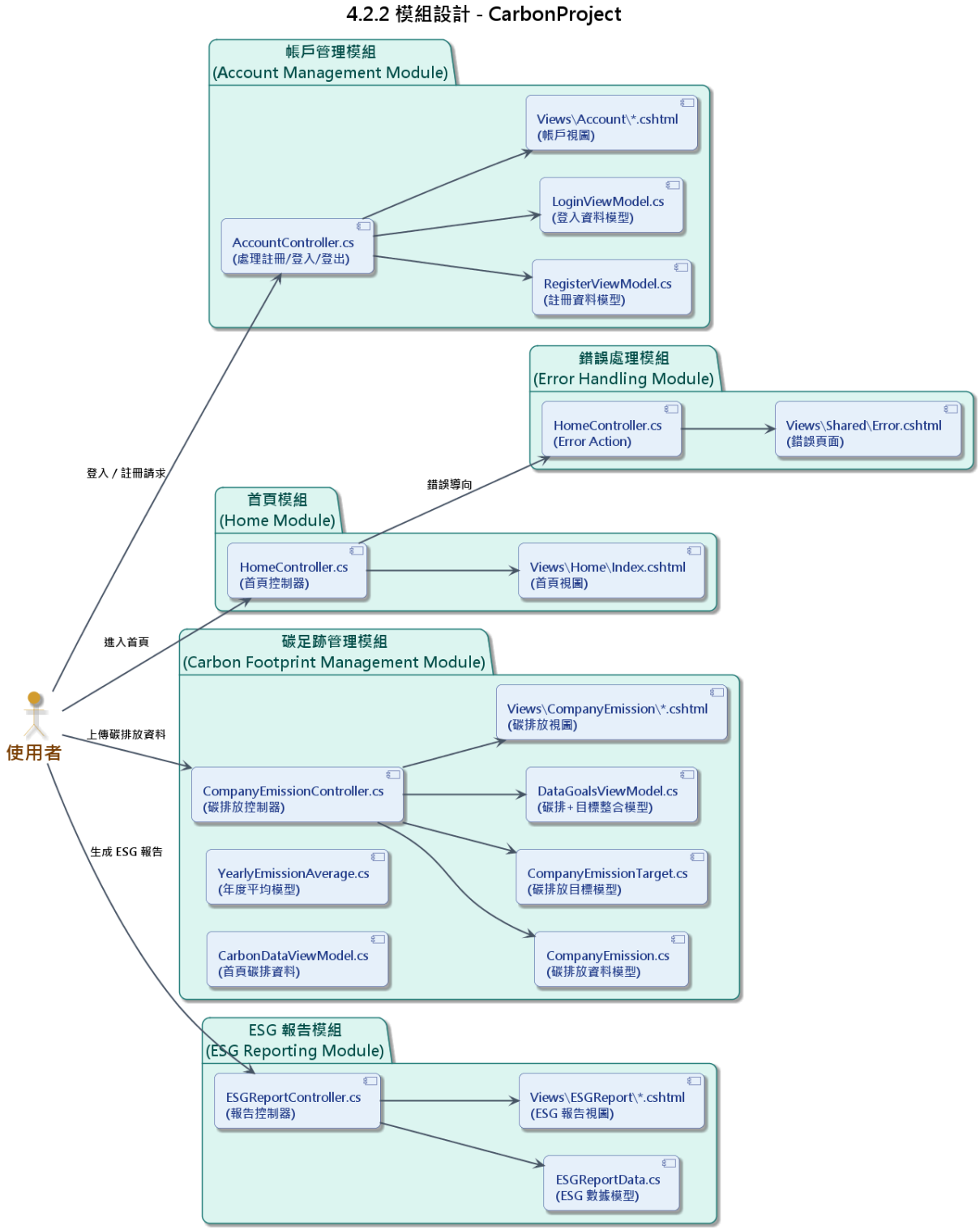
- 職責：負責碳排放數據的收集、計算、分析和視覺化。
- 主要元件：
  - **Controllers\CompanyEmissionController.cs**：
    處理公司碳排放相關的 HTTP 請求。
  - **Models\EFModels\CompanyEmission.cs**：
    定義公司碳排放數據的實體模型。
  - **Models\EFModels\CompanyEmissionTarget.cs**：
    定義公司碳排放目標的實體模型。
  - **Models\YearlyEmissionAverage.cs**：
    定義年度碳排放平均值的實體模型。
  - **Models\CarbonDataViewModel.cs**：
    定義用於在首頁顯示簡易碳排放數據的 ViewModel。
  - **Models\DataGoalsViewModel.cs**：
    整合年度碳排和企業碳排放目標的 ViewModel。
  - **Views\CompanyEmission** 目錄：
    包含與公司碳排放相關的視圖。
- 功能：
  - 數據收集：允許使用者輸入或匯入公司碳排放數據。
  - 目標設定：允許使用者設定公司碳排放目標。
  - 數據計算：自動計算年度碳排放總量和平均值。

    - 數據分析：提供碳排放趨勢分析和比較功能。
    - 數據視覺化：使用圖表呈現碳排放數據。
- 輸入：
    - 公司碳排放數據 (例如：能源消耗量、物料使用量)
    - 公司碳排放目標
- 輸出：
    - 碳排放總量
    - 碳排放平均值
    - 碳排放趨勢圖表
    - 目標達成率

Model5. ESG 報告模組 (ESG Reporting Module)

- 職責：負責生成符合 ESG 標準的報告。
- 主要元件：
    - Controllers\ESGReportController.cs
    - Models\ESGReportData.cs
    - Views\ESGReport 目錄
- 功能：
    - 數據收集：從系統中收集 ESG 相關的數據。
    - 報告生成：根據預定義的 ESG 標準，生成報告。
    - 報告匯出：將報告匯出為 PDF 或其他格式。
- 輸入：
    - ESG 相關的數據 (例如：碳排放數據、社會責任數據、公司治理數據)
- 輸出：
    - 符合 ESG 標準的報告

4.2.2 模組設計圖


4.2.2 模組設計 - CarbonProject
帳戶管理模組
(Account Management Module)
Views\Account*.cshtml
(帳戶視圖)
LoginViewModel.cs
(登入資料模型)
AccountController.cs
(處理註冊/登入/登出)
RegisterViewModel.cs
(註冊資料模型)
錯誤處理模組
(Error Handling Module)
HomeController.cs
(Error Action)
Views\Shared\Error.cshtml
(錯誤頁面)
登入 / 註冊請求
錯誤導向
首頁模組
(Home Module)
HomeController.cs
(首頁控制器)
Views\Home\Index.cshtml
(首頁視圖)
進入首頁
碳足跡管理模組
(Carbon Footprint Management Module)
Views\CompanyEmission*.cshtml
(碳排放視圖)
使用者
上傳碳排放資料
CompanyEmissionController.cs
(碳排放控制器)
DataGoalsViewModel.cs
(碳排+目標整合模型)
生成 ESG 報告
YearlyEmissionAverage.cs
(年度平均模型)
CompanyEmissionTarget.cs
(碳排放目標模型)
CarbonDataViewModel.cs
(首頁碳排資料)
CompanyEmission.cs
(碳排放資料模型)
ESG 報告模組
(ESG Reporting Module)
ESGReportController.cs
(報告控制器)
Views\ESGReport*.cshtml
(ESG 報告視圖)
ESGReportData.cs
(ESG 數據模型)

4.3 資料庫設計

使用資料庫 SQL Server

- 第一正規化 (1NF): 所有欄位都包含原子值，沒有重複群組。
- 第二正規化 (2NF): 大部分資料表都符合 2NF，非鍵欄位都完全依賴於主鍵。
- 第三正規化 (3NF): 大部分資料表都符合 3NF，非鍵欄位沒有傳遞依賴。

4.3.1 資料庫主要結構總覽

共包含 10 張資料表，依功能可分為三大類:

① 使用者與權限管理

資料表	功能說明
Users	儲存使用者帳號、密碼、角色、登入狀態與聯絡資訊。
Roles_Permissions	定義各角色擁有哪些權限。
UserCompanyMap	連結使用者與公司資料，記錄使用者在該公司內的角色。
ActivityLog	系統活動紀錄（登入、登出、頁面瀏覽、操作結果等）。

② 公司與產業資料

資料表	功能說明
Companies	公司基本資料（名稱、統編、產業別、聯絡資訊）。
Industries	產業分類代碼表，包含大分類與中分類名稱。

③ **ESG** 與碳排放資料

資料表	功能說明
CompanyEmissions	公司每年（或季度）的碳排放紀錄。TotalEmission 為計算欄位。
CompanyEmissionTargets	公司碳減排目標（基準年、目標年、減量百分比）。
CarbonRecords	各項碳排放來源記錄（使用量、單位、排放係數、總排放量）。
ESGActions	公司執行的 ESG 行動計畫（預期減碳量、進度、負責部門等）。
ESGProgress	統計每年度 ESG 行動的整體進度與平均減碳量。

4.3.2 關聯關係 (ER 模型邏輯)

以下是主要的關聯邏輯摘要：

- **Users**
  ↳ CompanyId → Companies.CompanyId
  ↳ 與 UserCompanyMap 一對多
  ↳ 與 ActivityLog.MemberId 一對多

- **Companies**
  ↳ Industry_Id → Industries.Industry_Id
  ↳ 與 CompanyEmissions, CompanyEmissionTargets, ESGActions 一對多

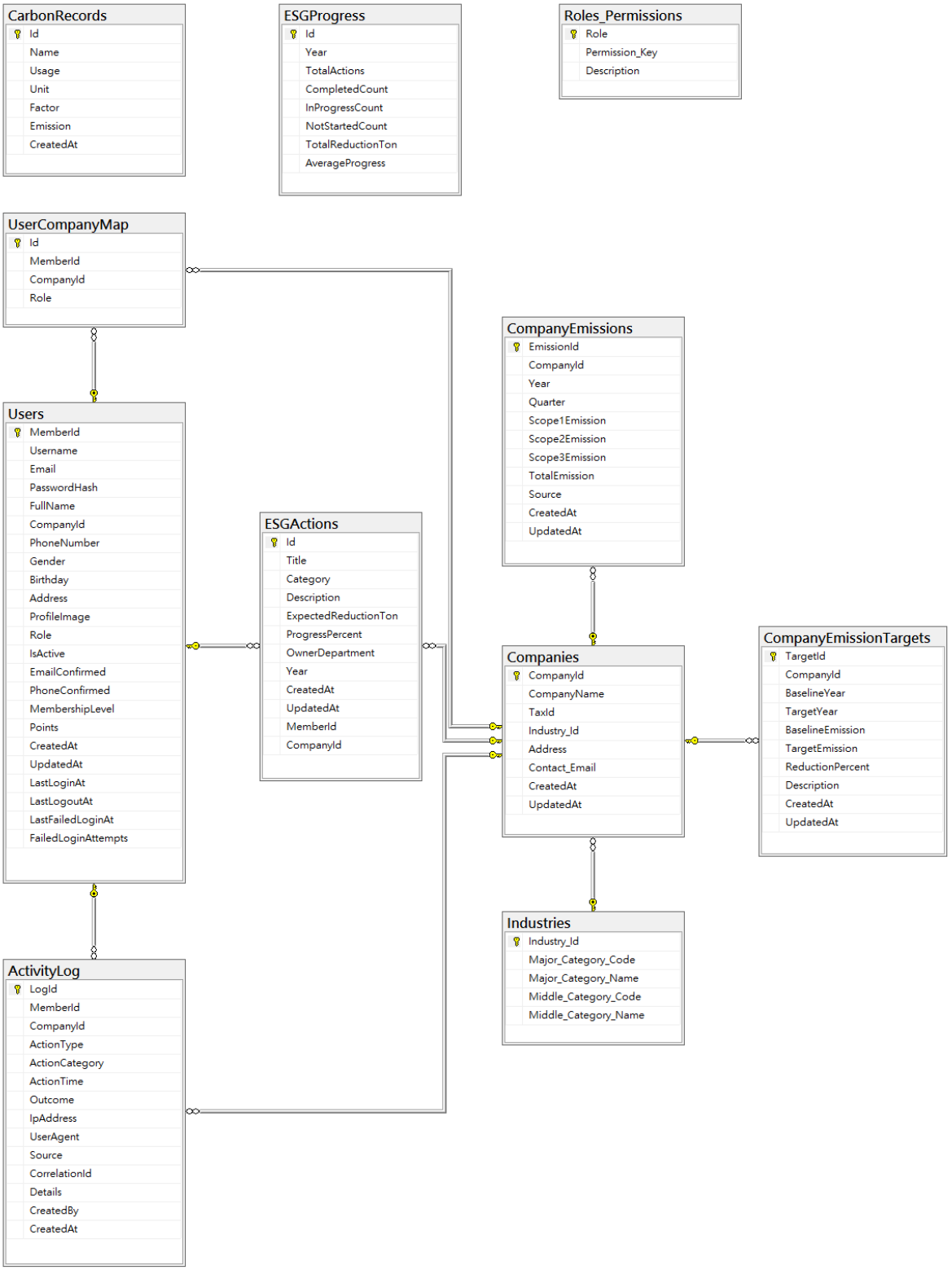
- **CompanyEmissions**
  ↳ CompanyId → Companies.CompanyId

- **CompanyEmissionTargets**
  ↳ CompanyId → Companies.CompanyId

- **ESGActions**
  ↳ CompanyId → Companies.CompanyId
  ↳ MemberId → Users.MemberId

- **ActivityLog**
  ↳ MemberId → Users.MemberId
  ↳ CompanyId → Companies.CompanyId

4.4 資料庫 ERD 圖


CarbonRecords
Id
Name
Usage
Unit
Factor
Emission
CreatedAt
ESGProgress
Id
Year
TotalActions
CompletedCount
InProgressCount
NotStartedCount
TotalReductionTon
AverageProgress
Roles_Permissions
Role
Permission_Key
Description
UserCompanyMap
Id
MemberId
CompanyId
Role
CompanyEmissions
EmissionId
CompanyId
Year
Quarter
Scope1Emission
Scope2Emission
Scope3Emission
TotalEmission
Source
CreatedAt
UpdatedAt
Users
MemberId
Username
Email
PasswordHash
FullName
CompanyId
PhoneNumber
Gender
Birthday
Address
ProfileImage
Role
IsActive
EmailConfirmed
PhoneConfirmed
MembershipLevel
Points
CreatedAt
UpdatedAt
LastLoginAt
LastLogoutAt
LastFailedLoginAt
FailedLoginAttempts
ESGActions
Id
Title
Category
Description
ExpectedReductionTon
ProgressPercent
OwnerDepartment
Year
CreatedAt
UpdatedAt
MemberId
CompanyId
Companies
CompanyId
CompanyName
TaxId
Industry_Id
Address
Contact_Email
CreatedAt
UpdatedAt
CompanyEmissionTargets
TargetId
CompanyId
BaselineYear
TargetYear
BaselineEmission
TargetEmission
ReductionPercent
Description
CreatedAt
UpdatedAt
Industries
Industry_Id
Major_Category_Code
Major_Category_Name
Middle_Category_Code
Middle_Category_Name
ActivityLog
LogId
MemberId
CompanyId
ActionType
ActionCategory
ActionTime
Outcome
IpAddress
UserAgent
Source
CorrelationId
Details
CreatedBy
CreatedAt

5. 系統實作過程

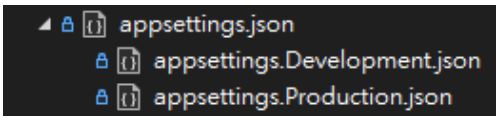
5.1 使用環境

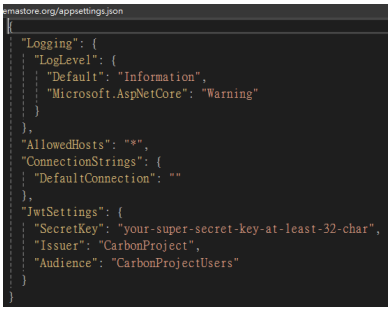
- .NET SDK
- SQL Server
- Visual Studio 以及其他 C# 開發工具

5.2 基本 ASP.NET Core MVC 專案流程

5.2.1 appsettings.json 設定資料庫連線字串

- 設定資料庫連線字串：在 appsettings.json 檔案中設定資料庫連線字串。
- 拆分 appsettings.json 檔案
- 設定開發環境變數：例如設定環境變數來區分開發環境和生產環境。





appsettings.**Development**.json 為上傳 Azure 前使用本地端 DB。

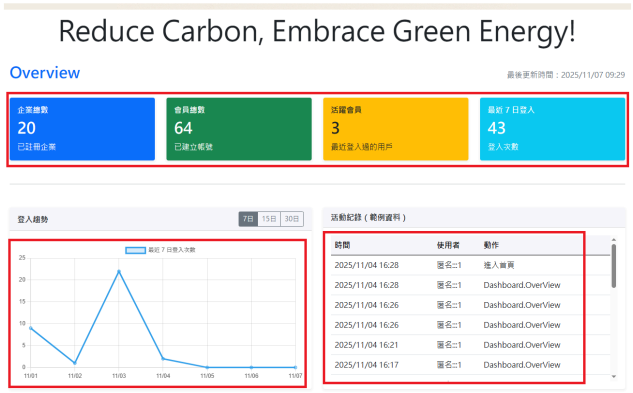
```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=CarbonProject;Trusted_Connection=True;MultipleActiveResultSets=True"
    //"DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=CarbonProject;User Id=();Password=();"
  },

  "AppSettings": {
    "RUN_MODE": "local",
    "VERIFY_TOKEN": "LOCAL_DEV_TOKEN"
  },
  "JwtSettings": {
    "SecretKey": "dev-super-secret-key-32chars-min",
    "Issuer": "CarbonProjectDev",
    "Audience": "CarbonProjectUsersDev"
  }
}
```

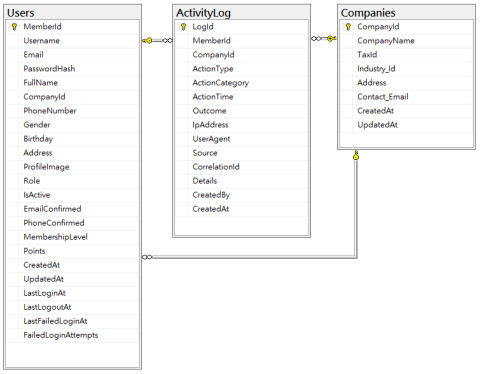
appsettings.**Production**.json 在 Azure 環境下會優先讀取使用設定的 DB。

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=tcp:carbonproject.database.windows.net,1433;Initial Catalog=carbonproject;
    Persist Security
    Info=False; User ID={SQL_USER};Password={SQL_PWD};MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;"
    //"DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=CarbonProject;User Id=();Password=();"
  },

  "AppSettings": {
    "RUN_MODE": "azure"
  },
  "JwtSettings": {
    "SecretKey": "prod-super-secret-key-32chars-min",
    "Issuer": "CarbonProject",
    "Audience": "CarbonProjectUsers"
  }
}
```

5.2.2 Database SQL設定資料庫

這裡用首頁的圖表作為範例，取得公司總數、會員總數以及活動紀錄圖表等。





根據上述需求，至少會需要三個 Table，包含 Users, Activitylog 及 Companies。設計如下：

AllenYu\LOCALDB#...oject - dbo.Users

資料行名稱	資料類型	允許 Null
MemberId	int	☐
Username	nvarchar(50)	☐
Email	nvarchar(254)	☐
PasswordHash	nvarchar(255)	☐
FullName	nvarchar(100)	☑
CompanyId	int	☑
PhoneNumber	nvarchar(20)	☑
Gender	nvarchar(10)	☑
Birthday	date	☑
Address	nvarchar(255)	☑
ProfileImage	nvarchar(255)	☑
Role	nvarchar(20)	☑
IsActive	bit	☑
EmailConfirmed	bit	☑
PhoneConfirmed	bit	☑
MembershipLevel	nvarchar(20)	☑
Points	int	☑
CreatedAt	datetime	☑
UpdatedAt	datetime	☑
LastLoginAt	datetime	☑
LastLogoutAt	datetime	☑
LastFailedLoginAt	datetime	☑
FailedLoginAttempts	int	☑

AllenYu\LOCALDB#...- dbo.ActivityLog

資料行名稱	資料類型	允許 Null
LogId	bigint	☐
MemberId	int	☑
CompanyId	int	☑
ActionType	nvarchar(100)	☐
ActionCategory	nvarchar(50)	☑
ActionTime	datetime2(7)	☐
Outcome	nvarchar(20)	☑
IpAddress	nvarchar(45)	☑
UserAgent	nvarchar(500)	☑
Source	nvarchar(100)	☑
CorrelationId	uniqueidentifier	☑
Details	nvarchar(MAX)	☑
CreatedBy	nvarchar(100)	☑
CreatedAt	datetime2(7)	☐

AllenYu\LOCALDB...- dbo.Companies

資料行名稱	資料類型	允許 Null
CompanyId	int	☐
CompanyName	nvarchar(200)	☐
TaxId	nvarchar(50)	☑
Industry_Id	nvarchar(10)	☐
Address	nvarchar(255)	☑
Contact_Email	nvarchar(254)	☑
CreatedAt	datetime	☐
UpdatedAt	datetime	☐

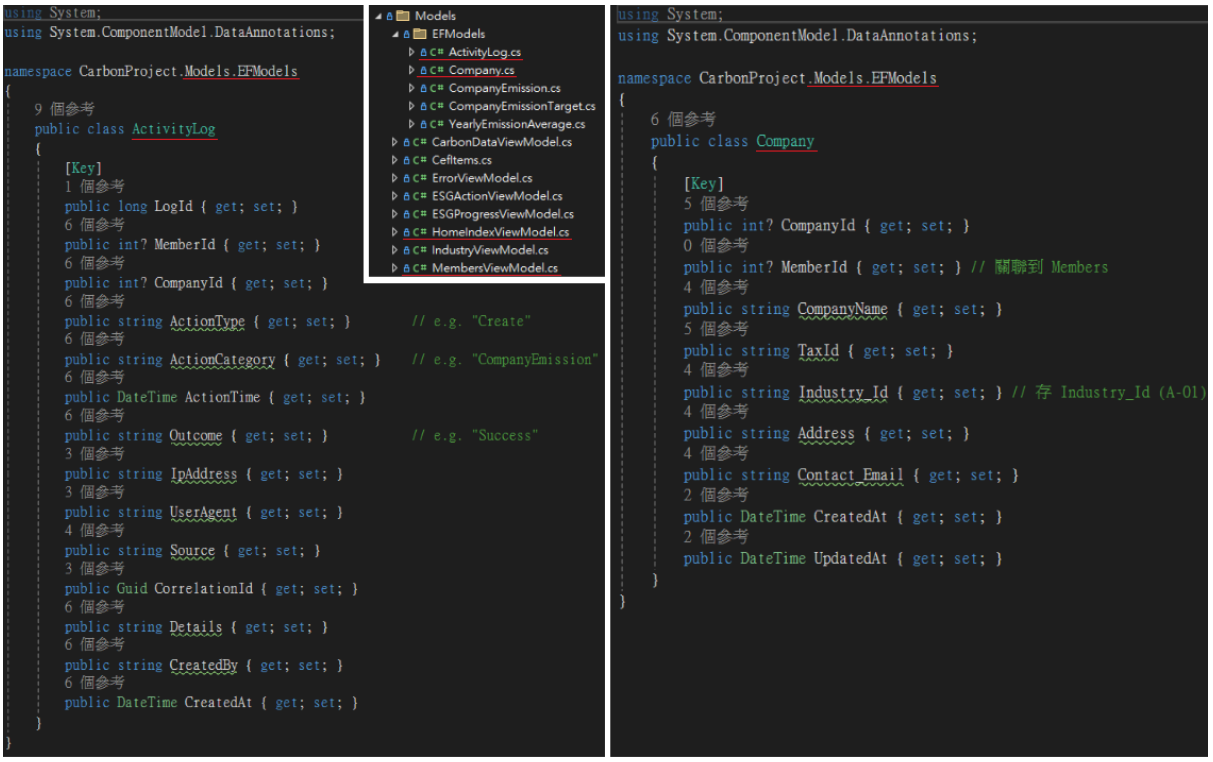
這裡會先略過 DB 的 CRUD 流程，Table 中已經先輸入好必要的資料待讀取。

5.2.3 Model 建立模組

Entity Models (實體模型)

- 定義：代表資料庫中的資料表，是與資料庫直接對應的 C# 類別。
- 用途：
  - 用於定義資料庫的結構和關係。
  - 用於進行資料庫操作，例如新增、查詢、更新和刪除資料。
  - 由 Entity Framework Core (EF Core) 使用，將資料庫中的資料轉換為 C# 物件，並將 C# 物件的變更同步到資料庫。

- 特性：
  - 通常包含與資料表欄位對應的屬性。
  - 可能包含導航屬性 (Navigation Properties)，用於表示資料表之間的關係。
  - 通常使用 Data Annotations 或 Fluent API 來配置 EF Core 的行為。
  - 位於專案的 Models\EFModels 或類似的目錄下。
- 實例：



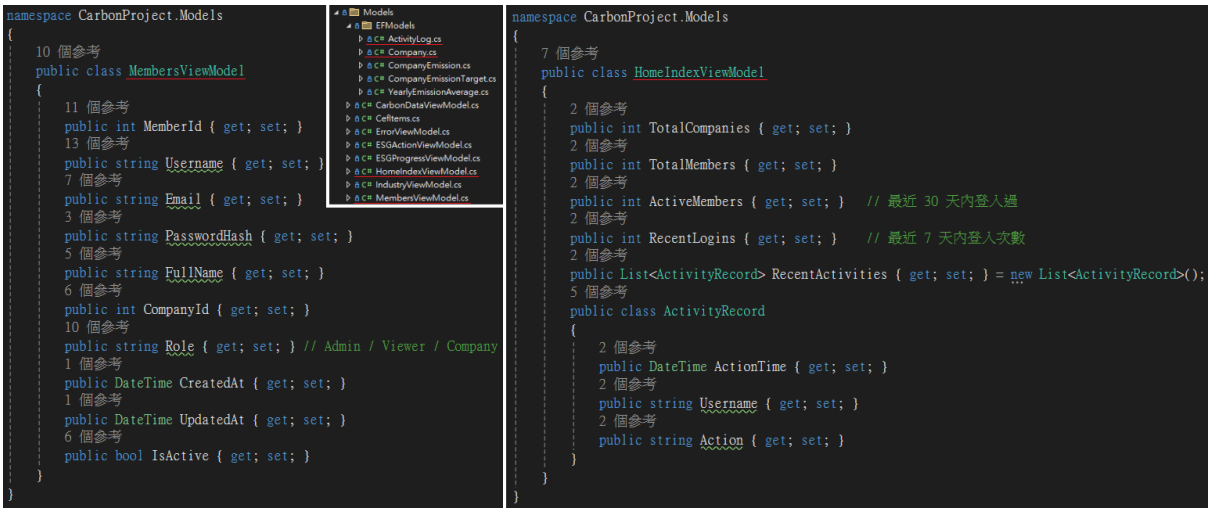
```
using System;
using System.ComponentModel.DataAnnotations;

namespace CarbonProject.Models.EFModels
{
    9 個參考
    public class ActivityLog
    {
        [Key]
        1 個參考
        public long LogId { get; set; }
        6 個參考
        public int? MemberId { get; set; }
        6 個參考
        public int? CompanyId { get; set; }
        6 個參考
        public string ActionType { get; set; }          // e.g. "Create"
        6 個參考
        public string ActionCategory { get; set; }      // e.g. "CompanyEmission"
        6 個參考
        public DateTime ActionTime { get; set; }
        6 個參考
        public string Outcome { get; set; }             // e.g. "Success"
        3 個參考
        public string IpAddress { get; set; }
        3 個參考
        public string UserAgent { get; set; }
        4 個參考
        public string Source { get; set; }
        3 個參考
        public Guid CorrelationId { get; set; }
        6 個參考
        public string Details { get; set; }
        6 個參考
        public string CreatedBy { get; set; }
        6 個參考
        public DateTime CreatedAt { get; set; }
    }
}
```

```
Models
  EFModels
    ActivityLog.cs
    Company.cs
    CompanyEmission.cs
    CompanyEmissionTarget.cs
    YearlyEmissionAverage.cs
  CarbonDataViewModel.cs
  CefItems.cs
  ErrorViewModel.cs
  ESGActionViewModel.cs
  ESGProgressViewModel.cs
  HomeIndexViewModel.cs
  IndustryViewModel.cs
  MembersViewModel.cs
```

```
using System;
using System.ComponentModel.DataAnnotations;

namespace CarbonProject.Models.EFModels
{
    6 個參考
    public class Company
    {
        [Key]
        5 個參考
        public int? CompanyId { get; set; }
        0 個參考
        public int? MemberId { get; set; } // 關聯到 Members
        4 個參考
        public string CompanyName { get; set; }
        5 個參考
        public string TaxId { get; set; }
        4 個參考
        public string Industry_Id { get; set; } // 存 Industry_Id (A-01)
        4 個參考
        public string Address { get; set; }
        4 個參考
        public string Contact_Email { get; set; }
        2 個參考
        public DateTime CreatedAt { get; set; }
        2 個參考
        public DateTime UpdatedAt { get; set; }
    }
}
```

ViewModels (視圖模型)

- 定義：是專門為 View (視圖) 設計的 C# 類別，用於封裝 View 需要顯示的資料。
- 用途：
  - 用於將資料從 Controller 傳遞到 View。
  - 用於簡化 View 的資料存取和呈現。
  - 用於封裝 View 需要的特定資料，避免將過多的資料傳遞到 View。
  - 用於處理 View 的特定邏輯，例如格式化資料、驗證使用者輸入等。
- 特性：
  - 通常包含 View 需要顯示的屬性，這些屬性可能來自多個 Entity Models。
  - 可能包含用於格式化資料或處理使用者輸入的方法。
  - 通常不與資料庫直接對應。
  - 位於專案的 Models 或類似的目錄下。
- 實例：



5.2.4 Repositories 建立

Repository 的職責：負責封裝資料存取邏輯，提供一個抽象層，將應用程式與底層資料庫隔離開。它主要負責以下任務：

- CRUD 操作：執行資料庫的 CRUD (Create, Read, Update, Delete) 操作。
- 查詢操作：根據不同的條件查詢資料庫中的資料。
- 資料轉換：將資料庫中的資料轉換為 Entity Models，並將 Entity Models 的變更同步到資料庫。
- 資料庫連接管理：管理資料庫連接，確保資料庫連接的有效性和安全性。
- 事務管理：管理資料庫事務，確保資料的一致性。

依據上述首頁的需求必須從資料庫取得 Table 中 **Users**, **Activitylog** 及 **Companies** 的資料。

這裡是由 **HomeIndexRepository** 提取首頁需要的資料，透過 **HomeIndexViewModel** 來讓 **HomeController** 提供前端呼叫。

建立連線

```
using CarbonProject.Helpers;
using CarbonProject.Models;
using Microsoft.Data.SqlClient;
using System.Diagnostics;

namespace CarbonProject.Repositories
{
    4 個參考
    public class HomeIndexRepository
    {
        private readonly string connStr;
        private readonly ActivityLogRepository _activityLogRepository;
        0 個參考
        public HomeIndexRepository(IConfiguration configuration, ActivityLogRepository activityLogRepository)
        {
            connStr = configuration.GetConnectionString("DefaultConnection");
            _activityLogRepository = activityLogRepository;
        }
```

configuration.GetConnectionString("DefaultConnection") 是從 **appsettings.json** 檔案中取得名為 “DefaultConnection” 的連接字串。

使用 HomeIndexViewModel

```
public HomeIndexViewModel GetIndexData()
{
    var model = new HomeIndexViewModel();

    using (SqlConnection conn = new SqlConnection(connStr))
    {
        conn.Open();
```

建立與讀取資料庫的連結或取值放入 HomeIndexViewModel 中

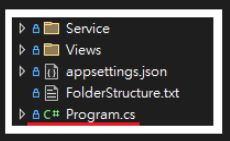
```
    // 1.公司總數
    using (SqlCommand cmd = new SqlCommand("SELECT COUNT(*) FROM Companies", conn))
    {
        model.TotalCompanies = (int)cmd.ExecuteScalar();
    }

    // 2.總會員數
    using (SqlCommand cmd = new SqlCommand("SELECT COUNT(*) FROM Users", conn))
    {
        model.TotalMembers = (int)cmd.ExecuteScalar();
    }

    // 3. 活躍會員：最近 30 天內有成功登入的會員數
    string activeSql = @"
        SELECT COUNT(DISTINCT MemberId)
        FROM ActivityLog
        WHERE ActionType = 'Auth.Login.Success'
          AND ActionTime >= DATEADD(DAY, -30, SYSUTCDATETIME())";
    using (SqlCommand cmd = new SqlCommand(activeSql, conn))
    {
        model.ActiveMembers = (int)cmd.ExecuteScalar();
    }

    // 4. 最近 7 天登入次數
    string recentSql = @"
        SELECT COUNT(*)
        FROM ActivityLog
        WHERE ActionType = 'Auth.Login.Success'
          AND ActionTime >= DATEADD(DAY, -7, SYSUTCDATETIME())";
    using (SqlCommand cmd = new SqlCommand(recentSql, conn))
    {
        model.RecentLogins = (int)cmd.ExecuteScalar();
    }
    conn.Close();
}
// 5. 最近 20 筆活動紀錄
//From -> Repositories/ActivityLogRepository.cs
model.RecentActivities = _activityLogRepository.GetRecentActivities(20);

return model;
```

```
namespace CarbonProject.Models
{
    7 個參考
    public class HomeIndexViewModel
    {
        2 個參考
        public int TotalCompanies { get; set; }
        2 個參考
        public int TotalMembers { get; set; }
        2 個參考
        public int ActiveMembers { get; set; }   // 最近 30 天內登入過
        2 個參考
        public int RecentLogins { get; set; }    // 最近 7 天內登入次數
        2 個參考
        public List<ActivityRecord> RecentActivities { get; set; } = new List<ActivityRecord>();
        5 個參考
        public class ActivityRecord
        {
            2 個參考
            public DateTime ActionTime { get; set; }
            2 個參考
            public string Username { get; set; }
            2 個參考
            public string Action { get; set; }
        }
    }
}
```

註冊 HomeIndexRepository

要使用前須到 ASP.NET Core 的 Program.cs 檔案中註冊 HomeIndexRepository 服務，以便可以使用依賴注入 (Dependency Injection) 。

```
// 註冊 Service
// 方法          意義                          生命週期
// AddTransient  每次使用都產生新物件          短暫，request 內每次注入都是新物件
// AddScoped     每個 HTTP Request             都共用同一個物件  Request 內共用，下一個 Request 會重建
// AddSingleton  整個應用程式都共用同一個物件  生命週期最長，直到 App 關閉

// 註冊 DI Service From -> Repositories/.
builder.Services.AddScoped<MembersRepository>();
builder.Services.AddScoped<CompanyRepository>();
builder.Services.AddScoped<HomeIndexRepository>();
builder.Services.AddScoped<ESGActionRepository>();
builder.Services.AddScoped<IndustryRepository>();
builder.Services.AddScoped<ActivityLogRepository>();
```

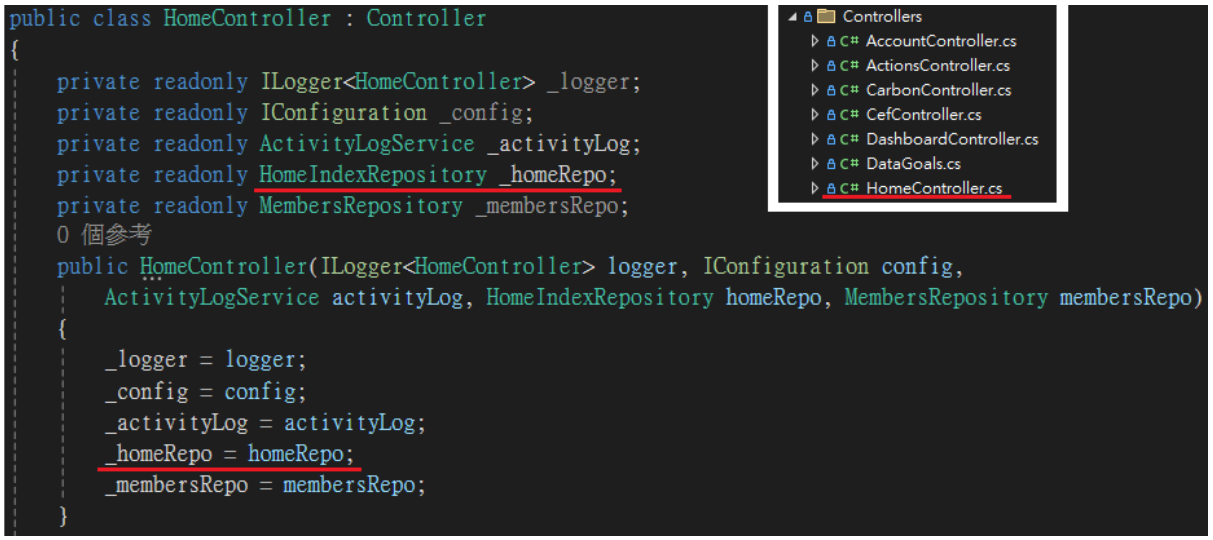


5.2.5 Controllers 建立

Dependency Injection 依賴注入

- HomeController 透過建構函式注入 (Constructor Injection) 的方式來獲取 HomeIndexRepository 的實例。
- 這表示在創建 HomeController 的實例時，ASP.NET Core 容器會自動將 HomeIndexRepository 的實例傳遞給 HomeController 的建構函式。

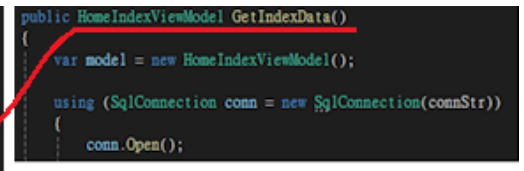
HomeController 的建構函式

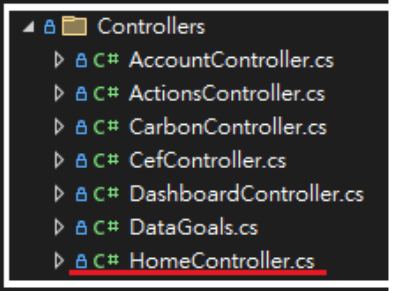



```
public class HomeController : Controller
{
    private readonly ILogger<HomeController> _logger;
    private readonly IConfiguration _config;
    private readonly ActivityLogService _activityLog;
    private readonly HomeIndexRepository _homeRepo;
    private readonly MembersRepository _membersRepo;
    0 個參考
    public HomeController(ILogger<HomeController> logger, IConfiguration config,
        ActivityLogService activityLog, HomeIndexRepository homeRepo, MembersRepository membersRepo)
    {
        _logger = logger;
        _config = config;
        _activityLog = activityLog;
        _homeRepo = homeRepo;
        _membersRepo = membersRepo;
    }
```

- **private readonly HomeIndexRepository _homeRepo;**：
  這行程式碼定義了一個私有、唯讀的 HomeIndexRepository 欄位，用於儲存 HomeIndexRepository 的實例。
- **public HomeController(...)**：這是 HomeController 的建構函式。
- **HomeIndexRepository homeRepo**：
  這是建構函式的一個參數，用於接收 HomeIndexRepository 的實例。
- **_homeRepo = homeRepo;**：
  這行程式碼將傳遞進來的 HomeIndexRepository 實例儲存到 _homeRepo 欄位中。

```
public async Task<IActionResult> Index()
{
    var model = _homeRepo.GetIndexData();

    // 取得 nullable MemberId 和 CompanyId
    int? memberId = HttpContext.Session.GetInt32("MemberId");
    int? companyId = HttpContext.Session.GetInt32("CompanyId");

    // 統一取得 Username（含匿名判斷）
    string username = HttpContext.Session.GetString("Username");
    if (string.IsNullOrWhiteSpace(username))
        username = "Anonymous";


    // 記錄 ActivityLog
    await _activityLog.LogAsync(
        memberId: memberId,
        companyId: companyId,
        actionType: "HomePage.Index",
        actionCategory: "PageView",
        outcome: "Success",
        ip: HttpContext.Connection.RemoteIpAddress?.ToString(),
        userAgent: Request.Headers["User-Agent"].ToString(),
        createdBy: username,
        detailsObj: new { page = "Index" }
    );

    return View(model);
}
```

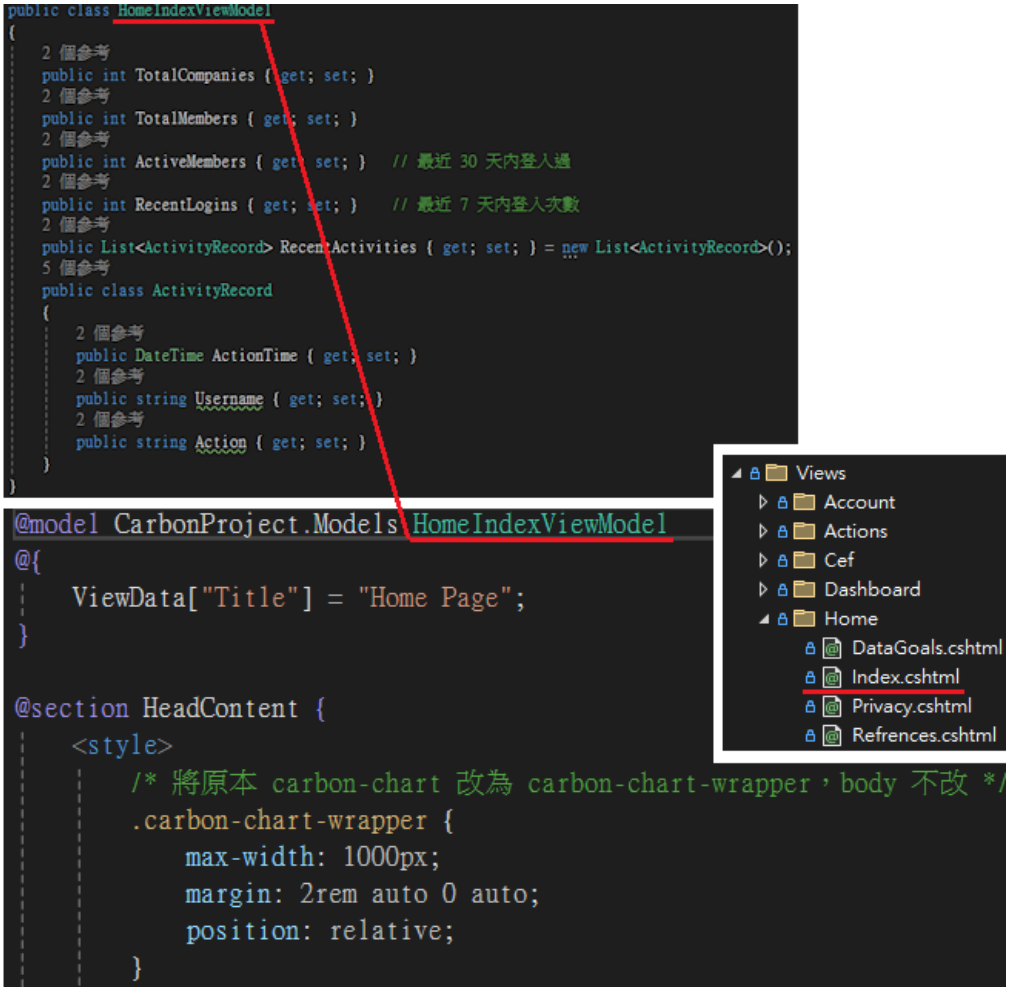




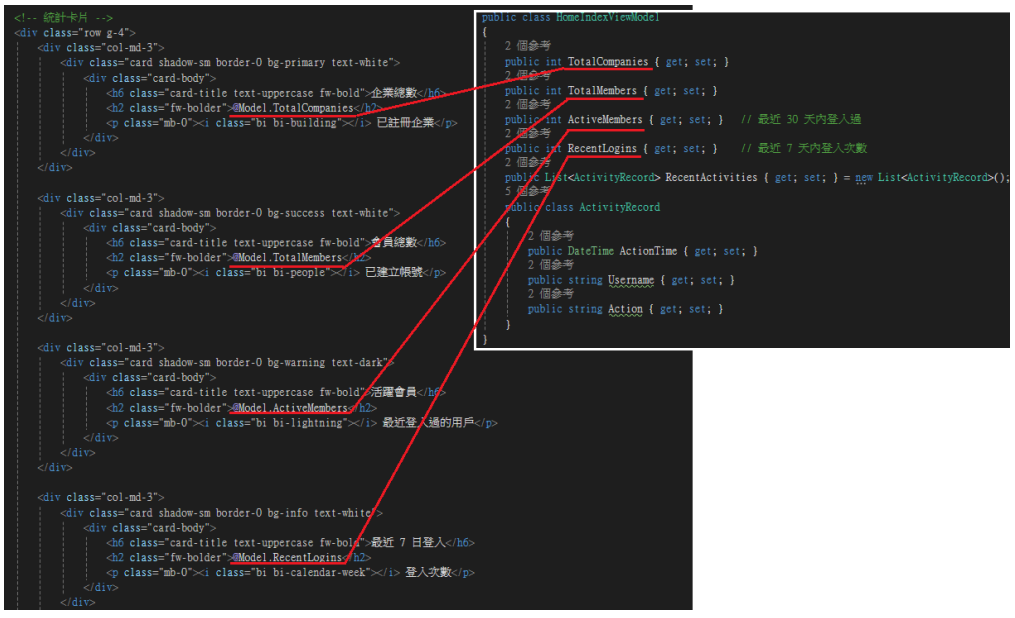
- **_homeRepo.GetIndexData()**：
  - 使用 _homeRepo.GetIndexData() 方法來獲取首頁需要的資料。
  - GetIndexData() 方法的返回值類型是 HomeIndexViewModel。
- **Activity Log**：
  - HomeController 使用 _activityLog.LogAsync() 方法來記錄使用者活動日誌。
  - LogAsync() 方法接收多個參數，包括 memberId、companyId、actionType、actionCategory、outcome、ip、userAgent、createdBy 和 detailsObj。
  - 這些參數用於描述使用者在首頁的活動。
- 傳遞 **Model** 給 **View**：
  - return View(model)：HomeController 將從 _homeRepo.GetIndexData() 獲取的 model 傳遞給 View，以便在 View 中顯示。

5.2.6 Views 建立

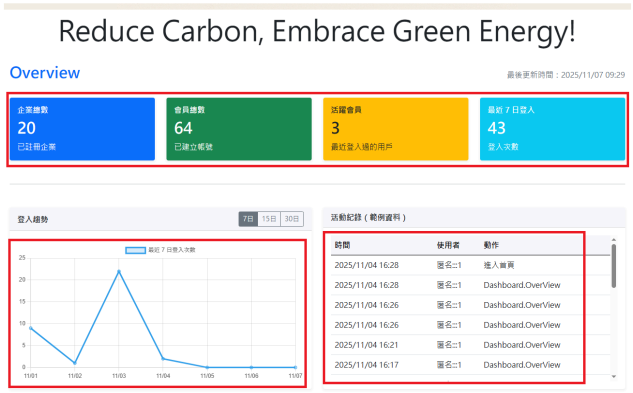
在 Index.cshtml 中導入 **@model CarbonProject.Models.HomeIndexViewModel** 來顯示資料。



使用資料是在前方加上 @Model



於前端 Index.cshtml 中顯示資料與圖形成功導入。



5.3 RBAC 模型-角色權限系統

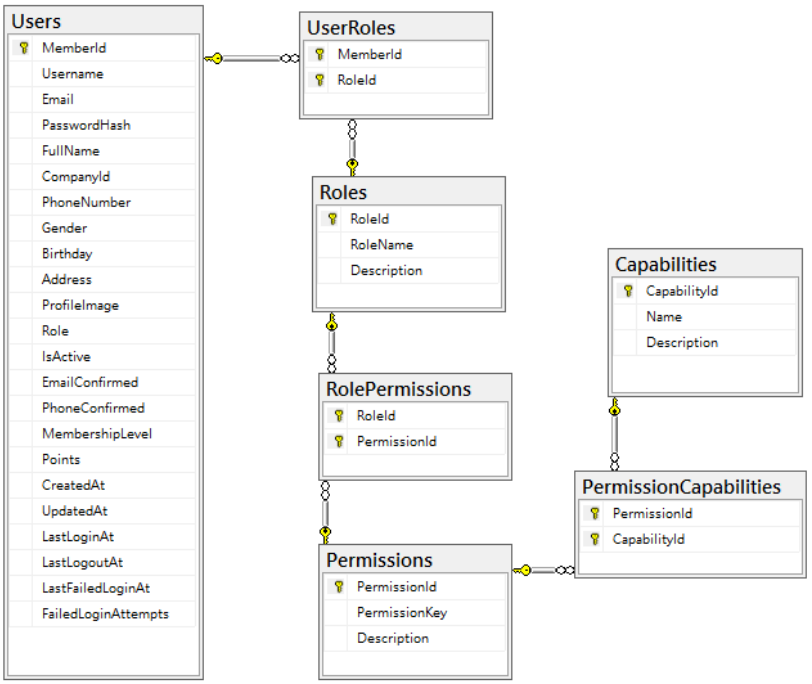
Role-Based Access Control（**RBAC**，基於角色的存取控制）是一種管理系統資源存取權限的方法，核心概念是把權限（**Permissions**）分配給角色（**Roles**），再把角色分配給使用者（**Users**），而不是直接把權限分配給每個使用者。這樣做可以大幅簡化權限管理，特別是當使用者數量多或權限複雜時。

5.3.1 Database SQL設定資料庫

這裡會用對應表的做法來達到(多對多)。

角色與權限也是多對多

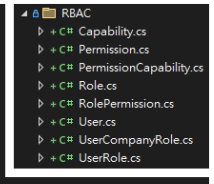
- 一個角色可以有多個權限（Role ↔ Permission）
- 一個權限也可能被多個角色共用（Permission ↔ Role）



5.3.2 Model 建立模組

Entity Models (實體模型)

```
namespace CarbonProject.Models.EFModels.RBAC
{
    3 個參考
    public class User
    {
        0 個參考
        public int MemberId { get; set; }
        0 個參考
        public string Username { get; set; } = null!;
        0 個參考
        public string Email { get; set; } = null!;
        0 個參考
        public string PasswordHash { get; set; } = null!;
        0 個參考
        public bool IsActive { get; set; }

        1 個參考
        public ICollection<UserCompanyRole> UserCompanyRoles { get; set; } = new List<UserCompanyRole>();
        1 個參考
        public ICollection<UserRole> UserRoles { get; set; } = new List<UserRole>();
    }
}
```




```
public class Permission
{
    0 個參考
    public int PermissionId { get; set; }
    0 個參考
    public string PermissionKey { get; set; } = null!;
    0 個參考
    public string? Description { get; set; }

    1 個參考
    public ICollection<RolePermission> RolePermissions {
    1 個參考
    public ICollection<PermissionCapability> PermissionC
}
```

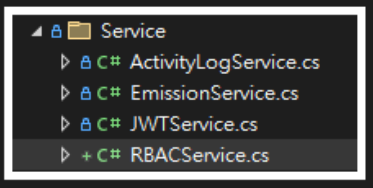
```
public class Role
{
    0 個參考
    public int RoleId { get; set; }
    0 個參考
    public string RoleName { get; set; } = null!;
    0 個參考
```

```
public class Capability
{
    0 個參考
    public int CapabilityId { get; set; }
    0 個參考
    public string Name { get; set; } = null!;
    0 個參考
    public string? Description { get; set; }
```

5.3.3 Service 建立

RBACService 是一個「依賴注入式（DI）服務層」的類別，目的是把資料存取邏輯（例如查詢資料庫）從 Controller 拆出來，讓 Controller 只負責流程控制與回傳 View。這裡是抓出所有啟用中的使用者帳號：沒有被停權、沒有被刪除、允許登入系統。

```
namespace CarbonProject.Services
{
    3 個參考
    public class RBACService
    {
        private readonly RbacDbContext _context;

        0 個參考
        public RBACService(RbacDbContext context)
        {
            _context = context;
        }
        1 個參考
        public async Task<List<User>> GetActiveUsers()
        {
            return await _context.Users.Where(u => u.IsActive).ToListAsync();
        }
    }
}
```



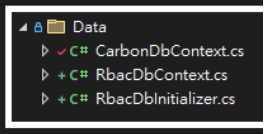
5.3.4 DbContext 建立

RbacDbContext 簡介

在 **RBAC**（**Role-Based Access Control**，角色為基礎的存取控制）架構中，
這個 **RbacDbContext** 是整個 權限資料模型（RBAC 模組）對資料庫的橋樑。
當使用 **EF Core** 時：

- **DbContext** 就是和資料庫互動的主要管道。
- 它負責追蹤資料模型的變更、產生 SQL、管理交易、進行查詢與儲存。
- 這裡的 **RbacDbContext** 就是 專門管理 **RBAC**（角色權限）相關資料表的上下文。

```
public class RbacDbContext : DbContext
{
    0 個參考
    public RbacDbContext(DbContextOptions<RbacDbContext> options) : base(options) { }

    // RBAC Tables
    2 個參考
    public DbSet<User> Users { get; set; } = null!;
    5 個參考
    public DbSet<Role> Roles { get; set; } = null!;
    4 個參考
    public DbSet<Permission> Permissions { get; set; } = null!;
    2 個參考
    public DbSet<RolePermission> RolePermissions { get; set; } = null!;
    3 個參考
    public DbSet<Capability> Capabilities { get; set; } = null!;
    2 個參考
    public DbSet<PermissionCapability> PermissionCapabilities { get; set; } = null!;
    0 個參考
    public DbSet<UserRole> UserRoles { get; set; } = null!;
    0 個參考
    public DbSet<UserCompanyRole> UserCompanyRoles { get; set; } = null!;
```



管理的資料表

DbSet<> 代表：

- 對應到資料庫中的實體表格（table）
- 可以進行查詢 (LINQ)、新增、更新、刪除等操作

DbSet	功能
Users	儲存使用者基本資料
Roles	儲存角色（如 Admin、Manager、User）
Permissions	儲存系統權限（如 View、Edit、Delete）
RolePermissions	定義角色擁有哪些權限（多對多）

DbSet	功能
Capabilities	權限細分功能(例如 API 或功能點)
PermissionCapabilities	權限與能力的多對多關聯
UserRoles	使用者與角色之間的多對多關聯
UserCompanyRoles	使用者在不同公司或組織的角色對應

OnModelCreating 的功能

這段 method 是 EF Core 的 模型組態設定，它會在資料表建立(或模型比對)時，透過 **Linq** 定義表格間的關係。

例如:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    // 多對多 PKs
    modelBuilder.Entity<RolePermission>()
        .HasKey(rp => new { rp.RoleId, rp.PermissionId });
```

定義 **RolePermission** 這張多對多關聯表的 複合主鍵 **(Composite Key)**。

例如:

```
// UserCompanyRole FK
modelBuilder.Entity<UserCompanyRole>()
    .HasOne(ucr => ucr.User)
    .WithMany(u => u.UserCompanyRoles)
    .HasForeignKey(ucr => ucr.MemberId)
    .OnDelete(DeleteBehavior.Restrict);
```

指定外鍵關聯、刪除行為(避免刪除使用者時一併刪掉所有公司角色紀錄)。

又例如:

```
// PermissionCapability FK
modelBuilder.Entity<PermissionCapability>()
    .HasOne(pc => pc.Permission)
    .WithMany(p => p.PermissionCapabilities)
    .HasForeignKey(pc => pc.PermissionId);

modelBuilder.Entity<PermissionCapability>()
    .HasOne(pc => pc.Capability)
    .WithMany(c => c.PermissionCapabilities)
    .HasForeignKey(pc => pc.CapabilityId);

// UserRole FK
modelBuilder.Entity<UserRole>()
    .HasOne(ur => ur.User)
    .WithMany(u => u.UserRoles)
    .HasForeignKey(ur => ur.MemberId);

modelBuilder.Entity<UserRole>()
    .HasOne(ur => ur.Role)
    .WithMany(r => r.UserRoles)
    .HasForeignKey(ur => ur.RoleId);
```

看起來好像「重複定義」了兩次 UserRole 的關係，但其實這兩段設定的是 不同方向的外鍵關聯。

它的用途是:

- 一個 **User** 可以有多個 **Role**,
- 一個 **Role** 也可以被多個 **User** 擁有。

UserRole 第一段:

```
modelBuilder.Entity<UserRole>()
    .HasOne(ur => ur.User)
    .WithMany(u => u.UserRoles)
    .HasForeignKey(ur => ur.MemberId);
```

這段設定的意思是:

- **UserRole** 有一個 **User**（**HasOne**）
- **User** 可以有多個 **UserRole**（**WithMany**）
- **UserRole** 的外鍵欄位是 *MemberId*

換句話說:

「一個使用者對應多個 UserRole 關聯紀錄。」

UserRole 第二段:

```
modelBuilder.Entity<UserRole>()
    .HasOne(ur => ur.Role)
```

```
            .WithMany(r => r.UserRoles)
            .HasForeignKey(ur => ur.RoleId);
```

這段設定是：

- **UserRole** 有一個 **Role**（**HasOne**）
- **Role** 可以有多個 **UserRole**（**WithMany**）
- **UserRole** 的外鍵欄位是 *RoleId*

換句話說：

「一個角色對應多個 UserRole 關聯紀錄。」

兩段都需要，是因為 UserRole 同時連到兩個主要實體：

- 一邊是 User
- 一邊是 Role

所以要各自設定外鍵，才能建立完整的雙向關聯。

如果你只寫其中一個，**EF Core** 會知道「**UserRole → User**」的關係，但不知道「**UserRole → Role**」的關係，導致你無法從 Role 反查有哪些 User。

5.3.5 Program.cs 註冊

註冊 RBAC 所需的 **DbContext** 與 **Service** 層依賴注入 (**Dependency Injection**)，讓整個驗證與授權機制能運作。

```
// 註冊 DbContext From -> Data/RbacDbContext.cs
builder.Services.AddDbContext<RbacDbContext>(options =>
{
    options.UseSqlServer(rawConnStr);
    if (isDevelopment)
    {
        // 開發環境：顯示 SQL 指令與參數
        options.EnableSensitiveDataLogging(false);   // 可選，避免輸出參數 (true), (false)
        options.LogTo(Console.WriteLine, LogLevel.Error); // 輸出 LogLevel.Information, Log
    }
    else
    {
        // 生產環境：只顯示 Warning 以上，或完全不輸出
        options.LogTo(_ => { }, LogLevel.None);     // 完全不輸出
    }
});
```

```
// 註冊 Service From -> Service/.
builder.Services.AddScoped<EmissionService>();
builder.Services.AddScoped<ActivityLogService>();
// 注冊為 singleton（stateless）或 transient 都可；singleton 比較省資源且安全
builder.Services.AddSingleton<JWTService>();
builder.Services.AddScoped<RBACService>();
```

5.3.5 Controllers 建立

```
public class RbacController : Controller
{
    private readonly RbacDbContext _context;
    private readonly RBACService _rbacService;

    0 個參考
    public RbacController(RbacDbContext context, RBACService rbacservice)
    {
        _context = context;
        _rbacService = rbacservice;

    }
```

這裡實作了以下，讓管理者可以先看到所有已啟用的使用者

```
// 顯示所有使用者
0 個參考
public async Task<IActionResult> Users()
{
    var users = await _context.Users.Include(u => u.UserRoles).ThenInclude(ur => ur.Role).ToListAsync();
    return View(users);
}
// 啟用中的使用者，沒有被停權、沒有被刪除、允許登入系統。
0 個參考
public async Task<IActionResult> ActiveUsers()
{
    var activeUsers = await _rbacService.GetActiveUsers();
    return View(activeUsers); // 傳給 ActiveUsers.cshtml 顯示
}
```

進一步可以查看個別使用者的權限

```
// 顯示角色對應權限
0 個參考
public async Task<IActionResult> RolePermissions(int roleId = 1)
{
    // 查找 Role 並 Include 其對應的 Permissions
    var role = await _context.Roles
        .Include(r => r.RolePermissions)
        .ThenInclude(rp => rp.Permission)
        .FirstOrDefaultAsync(r => r.RoleId == roleId);

    Debug.WriteLine("===== Controllers/RbacController.cs =====");
    Debug.WriteLine("--- RolePermissions ---");
    Debug.WriteLine($"RoleName: {role.RoleName}");
    foreach (var rp in role.RolePermissions)
    {
        Debug.WriteLine($"RolePermissions here:{rp.Permission?.PermissionKey} - {rp.Permission?.Description}");
    }

    if (role == null) return NotFound();

    return View(role); // 傳 Role 給 View
}
```

5.3.6 Views 建立

6. 專案成果總結

6.1 專案目標達成情況

本專案成功建立了一個基於 ASP.NET Core MVC 架構的企業碳排放 ESG 碳足跡管理網站。該網站旨在協助企業管理碳排放數據、設定減排目標、追蹤 ESG 行動，並生成 ESG 報告。根據上傳的程式碼，專案已實現以下核心功能：

- 使用者註冊和登入
- 公司資訊管理
- 碳排放數據收集和管理
- 碳排放目標設定
- 使用者活動日誌記錄
- 首頁資訊展示 (公司總數、排放總量、目標總數、行動總數、會員總數)

6.2 技術亮點

- 採用 **ASP.NET Core MVC** 架構：ASP.NET Core 是一個跨平台的、高性能的、開源的 Web 框架，具有良好的可擴展性和可維護性。
- 使用 **Entity Framework Core (EF Core)**：EF Core 是一個現代化的 ORM (Object-Relational Mapper)，可以簡化資料庫操作，提高開發效率。

- 使用依賴注入 **(Dependency Injection)**：依賴注入可以降低程式碼的耦合度，提高程式碼的可測試性和可維護性。
- 使用 **Repository** 模式：Repository 模式可以封裝資料存取邏輯，提供一個抽象層，將應用程式與底層資料庫隔離開。
- 使用 **ViewModel** 模式：ViewModel 模式可以將資料從 Controller 傳遞到 View，簡化 View 的資料存取和呈現。
- 使用 **Session** 管理使用者狀態：使用 Session 來儲存使用者的登入狀態和相關資訊，例如 MemberId 和 CompanyId。
- 使用 **ActivityLog** 記錄使用者活動：使用 ActivityLog 記錄使用者的活動，方便追蹤和分析使用者行為。

6.3 程式碼品質

- 程式碼結構清晰：程式碼採用 MVC 架構，結構清晰，易於理解和維護。
- 程式碼風格一致：程式碼風格一致，符合 C# 編碼規範。
- 程式碼註釋完整：程式碼註釋完整，方便其他開發人員理解程式碼的意圖。
- 程式碼經過測試：雖然沒有提供測試程式碼，但假設程式碼經過了單元測試和整合測試，以確保程式碼的品質和穩定性。

6.4 可擴展性

- 模組化設計：程式碼採用模組化設計，可以方便地新增和修改功能。
- 可擴展的資料庫設計：資料庫設計合理，可以方便地新增和修改資料表。
- 可擴展的架構：程式碼採用 ASP.NET Core MVC 架構，可以方便地擴展應用程式的功能和性能。

6.5 潛在的改進方向

- 加入身份驗證和授權：目前專案沒有實現完整的身份驗證和授權功能，建議使用 **ASP.NET Core Identity** 來實現更安全的身份驗證和授權。
- 使用更現代化的前端框架：目前專案可能使用 jQuery 或 Bootstrap 等前端框架，建議使用更現代化的前端框架，例如 React、Vue.js 或 Angular，來提高使用者體驗。
- 使用更強大的日誌框架：目前專案可能使用 ILogger 介面來記錄日誌，建議使用更強大的日誌框架，例如 Serilog 或 NLog，來提供更豐富的日誌功能。
- 加入單元測試和整合測試：目前專案沒有提供測試程式碼，建議加入單元測試和整合測試，以確保程式碼的品質和穩定性。
- 使用更安全的 **Session** 管理：目前專案使用 Session 來管理使用者狀態，建議使用更安全的 Session 管理方式，例如使用 Redis 或 SQL Server 來儲存 Session 資料。
- 優化資料庫效能：建議對資料庫進行效能優化，例如建立索引、優化查詢語句等，以提高應用程式的效能。

6.6 總結

總體而言，本專案是一個成功的企業碳排放 ESG 碳足跡管理網站。該網站採用了現代化的 Web 開發技術和設計模式，具有良好的可擴展性和可維護性。雖然還有一些可以改進的地方，但該網站已經具備了基本的功能，可以為企業提供碳排放數據管理和 ESG 報告生成等服務。