1. 分層(層級)使用不同專案
2. 生命週期 中間執行 application error/session

(會用到的內建的模組類別 資料)

1. 跟第三方對接 資料的model 是甚麼?(api)
2. attribute 是甚麼(httppost…Required…)

EX：[Required(ErrorMessage ="帳號不得為空")]

[RegularExpression ( "^(?=.\*[a-zA-Z])(?=.\*\\d)[a-zA-Z\\d]{8,20}$", ErrorMessage = "請輸入8~20位英文數字" )]

1. web config debug mode(false的話為預編譯, true的話會啟用熱重載)
2. actionResult 回傳的參數類型有些哪（延伸=>用處
3. viewbag/viewdata動態型別
4. catch到的error不要到error頁面（根據情況重定向到指定頁面）
5. Home = index (view) view的名子
6. shared資料夾研究, 設定共用view　有沒有其他客製的父用組件
7. 開單元測試做驗證
8. ?.稍微理解domain領域驅動開發
9. 分層架構

* Repository Pattern

首先為了把資料庫操作方法從action抽離出來，會使用一種設計模式［Repository Pattern(倉儲模式)］

要先建立 Repository 類別的介面，建立介面是為了要避免直接依賴 Repository 類別，往後在 Controller 中只需要用介面來進行資料操作，而不需直接使用 Repository 類別；

建立一個通用的IRepository以及實作GenericRepository就可以避免重複的CRUD操作

* Service層

服務層，主要是把系統的商業邏輯給封裝起來，Service 層使用 Repository 層所提供的服務來存取資料， Controller 則是透過 Service 層來做資料的處理，而不直接使用 Repository

* Web（Controller）層

處理controller action, 不涉及後端資料存取以及商務邏輯

1. MVC常用全局處理方法

* Application\_Error

捕獲全局未處理的異常，讓應用程式不至於崩潰

用途：記錄日至、得知異常訊息、清理已經處理完的錯誤（否則aps.net會默認處理）

* Session\_Start

Session\_Start事件會在用戶的會話首次開始時觸發(首次訪問網站或者過期時觸發)

用途：初始化會話狀態

* Session\_End

Session\_End事件會在會話結束時被觸發(登出或過期)

不過 Session\_End事件只會在Inproc會話模式下觸發，StateServer或SQLServer不會觸發

1. 數據傳輸對象（DTO）

業務模型包含接收資料庫資料的模型，且可以接收業務邏輯。

而數據傳輸對象：

目的－>用于不同系統或層之間的數據傳遞，通常是輕量級、結構化的數據容器，沒有業務邏輯，只包含數據字段

1. Attribute

是一種用於標註class, function或屬性的元數據，他們提供了關於應用程序組件的額外訊息，或影響程序的行為，可以簡化程式的複雜度

通過聲明的方式添加行為或者規則

* Action Filter

Authorize：用於控制用戶的訪問權限

OutputCache：用於緩存Action的Result，提高效能

HandleError：處理程式中未處理的異常

* Model Validation Attributes

用於驗證數據模型的字段值，用於控制器方法的參數驗證

Required：字段不為空

StringLength：字串的最大和最小長度

Range：限制數值範圍

RegularExpression：用正則表達式驗證值

* Routing Attributes

這些屬性用於控制路由和URL映射

Route：為某個控制器或動作指定字定義路由

* ActionMethodSelectorAttribute

ActionMethodSelectorAttribute是一個方法選擇器

[HttpPost]繼承自ActionMethodSelectorAttribute，使用HttpPost則重寫了當中的Request.Method變數，設定為Post

* 自定義Attribute

1. ActionResult

ActionResult是一個抽象基底類別，代表從Action方法返回的任何結果（使用到多型繼承等等概念）

常見子類：

* **ViewResult**：返回視圖頁面。
* **RedirectToRouteResult**：重定向到另一個路由。
* **JsonResult**：返回 JSON 格式的數據。
* **PartialViewResult**：返回局部視圖。
* **ContentResult**：返回文本內容。
* **FileResult**：返回文件下載。

延伸到SOLID中的 開放－封閉原則　意思擴展開放，對修改封閉

S = **Single-responsibility principle** **(SRP)** = 單一職責原則  
O = **Open–closed principle** **(OCP)** = 開放封閉原則  
L =**Liskov substitution principle** **(LSP)** = 里氏替換原則  
I = **Interface segregation principle** **(ISP)** = 介面隔離原則

D = **Dependency inversion principle** **(DIP)** = 依賴反向原則

ViewBag/ViewData

**ViewData** 和 **ViewBag** 都是用來在控制器和視圖之間傳遞數據的容器。

* ViewData
* 儲存結構：ViewData是字典（Dictionary<string, object>）其中鍵是字符串，而object是任意對象
* 數據訪問方式：使用鍵來訪問對應數據
* 原理：ViewData 存儲在 Controller 類型的 ViewDataDictionary 屬性中，並在同一個請求中傳遞到視圖。ViewData 會在頁面渲染後被銷毀。
* ViewBag

ViewBag是一個動態對象，是C#中的dynamic類型，所以可以將數據做為屬性附加上去。

* 儲存結構：ViewBag是基於ViewData的一個動態屬性封裝
* 數據訪問方式：像訪問普通物件屬性一樣訪問 ViewBag 中的數據
* 原理：允許在運行時動態地創建屬性
* 缺點：動態類型的屬性在編譯時不會被檢查，要小心

Layout視圖（共用視圖）

可以在子view使用程式碼來指定希望的父頁面

@{ Layout = "~/Views/Shared/\_LoginLayout.cshtml"; }

Domain領域驅動

領域驅動設計（Domain-Driven Design，簡稱DDD）是一種軟體設計方法論

DDD 強調以業務領域為中心來設計系統，並通過建立領域模型來構建能夠清楚表達業務需求的軟體系統。

* **核心概念**

1. **領域（Domain）：**

指的是業務問題所在的範疇，通常由業務專家來定義。領域是設計的焦點，開發者需要與業務專家密切合作，理解他們的需求和語言。

1. **子領域（Subdomain）**：

領域可以被分解為若干子領域，每個子領域解決一部分業務問題。

1. **限界上下文（Bounded Context）**：

一個限界上下文是在系統中對某個領域的特定描述或模型進行定義的範圍。每個限界上下文內部可以有一個獨立的領域模型，這有助於避免不同領域模型之間的混淆和衝突。

1. **領域模型（Domain Model）**：

領域模型是業務領域的抽象表示，通常由實體（Entity）、值對象（Value Object）和聚合（Aggregate）等組成。是與專家交流的工具

1. **實體（Entity）**：

* 實體是具有唯一標識符的對象，這意味著即使它的屬性改變，實體依然可以被區分開來。實體通常代表業務中的某個對象，如用戶、訂單等

1. **值對象（Value Object）**：

值對象沒有唯一標識符，它們是不可變的，並且通常代表某些屬性組合。例如，地址或金額通常可以作為值對象來處理。

1. **聚合（Aggregate）**：

聚合是由一組相關聯的實體和值對象構成的集合，它們一起作為一個單位來進行操作。聚合有一個根實體（Aggregate Root），它是外部與聚合交互的唯一入口。

1. **領域服務（Domain Service）**：

當某些業務邏輯不能完全納入某個實體或值對象時，這些邏輯應該被放入領域服務中。領域服務通常處理跨多個實體的業務邏輯。

1. **應用服務（Application Service）**：

應用服務是用來協調領域邏輯和應用邏輯的層，通常處理外部請求，調用相應的領域服務或聚合來完成具體的業務操作。

1. **反向工程（Event Sourcing）和CQRS（Command Query Responsibility Segregation）**：

這兩個技術和架構模式有助於管理複雜的業務邏輯，通過事件驅動設計來更清晰地建模狀態變化。