



圖源：[worldskills](https://www.worldskills.org/)

商務軟體設計

IT Software Solutions for Business

目錄

壹、	前言.....	1
貳、	SQL Server (SSMS).....	1
一、	學習網站.....	1
二、	程式碼.....	1
1.	增刪查改 (CRUD).....	1
2.	創建資料表.....	3
3.	創建檢視表.....	3
4.	匯入大量資料.....	3
三、	經驗.....	4
參、	C# (Visual Studio).....	5
一、	推薦網站.....	5
二、	教學影片.....	5
三、	程式碼.....	5
1.	串接 API GET Method 取得字串	5
2.	串接 API POST Method Post json 字串	6
3.	串接 API GET Method 取得圖片	7
4.	Json 序列化 (C#物件轉 Json 字串).....	7
5.	Json 反序列化 (Json 字串轉 C#物件).....	8
6.	自定義類別 (Class).....	8
7.	實體資料模型.....	9
8.	LINQ.....	9
9.	LINQ to SQL	11
10.	匿名觀念.....	13
11.	Lambda 表達式	14

12.	Func<>、Action、Action<>.....	15
13.	擴充方法.....	15
14.	顯示表格 DataTable DataGridView.....	16
15.	篩選表格 BindingSource、DataGridView.....	18
16.	多線程 Task.....	19
17.	委派 Invoke、Delegate.....	19
四、	經驗.....	20
肆、	Android Studio (Java).....	24
一、	推薦網站.....	24
二、	教學影片.....	24
三、	程式碼.....	24
1.	串接 API GET Method 取得字串.....	24
2.	串接 API POST Method Post json 字串.....	27
3.	串接 API GET Method 取得圖片.....	29
4.	Json 序列化 JSONObject to JsonString.....	29
5.	Json 反序列化 JsonString to JSONObject.....	30
6.	try-catch Exception.....	30
7.	多線程.....	31
8.	runOnUiThread.....	31
9.	Context.....	32
10.	Activity、Fragment、Context.....	32
四、	經驗.....	33
伍、	Web API (全國賽).....	35
一、	前言.....	35
二、	程式碼.....	35
1.	新增控制器.....	35

2. 實體資料模型.....	35
3. 使用 Json 格式作為回傳值	36
4. GET Method 回傳資料	36
5. GET Method 回傳圖片	37
6. POST Method 新增資料	38
7. Put Method 修改資料.....	40
8. Delete Method 刪除資料.....	43
9. Body.....	48
10. 路由.....	48
11. HTTP 方法.....	49
12. 發佈 Web API.....	49
13. 部署到 IIS.....	49
三、 經驗.....	50
陸、 總結.....	50

壹、 前言

成為選手後，最重要的是培養自己解決問題的能力，不論是察看這份文件、上網找答案、使用 Ai 工具...等等。不論是何種方法，都是為了學習拆分問題、尋找解答。

這份文件將會上傳到 [GitHub 儲存庫](#)，未來有更新文件也會在 GitHub 上進行更新。如果有問題嘗試了各種方法仍然無法解決，可到 GitHub 上進行討論 (Discussions)。若文件有錯誤，也請在 [GitHub 儲存庫](#) 提出，我會盡快改正。

比賽過程中無法使用任何第三方程式庫!比賽前會有測試環境時間，請好好利用
以下所有程式碼僅供參考，請勿直接複製貼上，親手撰寫過才會有印象。

以下內容如有侵權，請告知刪除。

貳、 SQL Server (SSMS)

一、 學習網站

1. [SQL 語法教學-1Keydata](#)
2. [W3Schools-SQL](#)
3. [Learn SQL | Codecademy](#)

二、 程式碼

1. 增刪查改 (CRUD)

增加 (Create) 增加一筆資料

```
INSERT INTO TableName (Column1, Column2, Column3, ...)
VALUES (Value1, Value2, Value3, ...);
```

查詢 (Read) 查詢符合條件的資料

```
SELECT Column1, Column2, ...  
  
FROM TableName  
  
WHERE Condition;
```

```
SELECT *  
  
FROM TableName  
  
WHERE Condition;
```

改正 (Update) 修改符合條件的資料

```
UPDATE TableName  
  
SET Column1 = Value1, Column2 = Value2, ...  
  
WHERE Condition;
```

刪除 (Delete) 刪除符合條件的資料

```
DELETE FROM TableName  
  
WHERE Condition;
```

Condition 指的是篩選條件，例如，查詢 Name 是 John 的資料

```
SELECT *  
  
FROM TableName  
  
WHERE Name = 'John';
```

Condition 中可以使用 LIKE、萬用字元、AND、OR，組合出不同篩選條件。

具體可參考 [SQL LIKE](#)、[SQL 萬用字元](#)、[SQL AND OR](#)。

2. 創建資料表

資料庫中儲存資料的基本架構，實際儲存資料。

SSMS 中常用工具創建資料表，不需撰寫 SQL 語法，不過在 Android Studio 中要創建資料表就必須使用 SQL 語法。

```
CREATE TABLE TableName (  
    Column1 DataType CONSTRAINTS,  
    Column2 DataType CONSTRAINTS,  
    Column3 DataType CONSTRAINTS,  
    ...  
);
```

CONSTRAINTS：為表格的某一個欄位增加限制，可有可無，常見的限制有 NOT NULL、UNIQUE、CHECK、主鍵 (Primary Key)、外來鍵 (Foreign Key)。具體可參考 [SQL CONSTRAINTS](#)。

3. 創建檢視表

檢視表只儲存語法，它本身並不實際儲存資料。顯示的資料僅為語法執行結果

```
CREATE VIEW ViewName AS  
  
SELECT Column1, Column2, ...  
  
FROM TableName  
  
WHERE Condition;
```

4. 匯入大量資料

可直接從 Excel 中複製貼上，不過面對大量資料，效率極差。

使用匯入匯出精靈，選擇 Microsoft Excel 作為資料來源匯入，不過疑似需要額外安裝軟體，比賽中部一定會安裝。或是將資料複製至記事本，資料來源選擇 Flat File Source。上述匯入方法皆有可能失敗，目前無較好的匯入資料方法。

三、 經驗

1. SQL 語法的**關鍵字(保留字)**不區分大小寫，但通常會全部大寫。
SELECT、FROM、WHERE、VIEW、AS...等等，都是保留字。SSMS 中保留字會以藍色顯示，具體可查詢[保留字列表](#)。
2. 必須要知道什麼是[主鍵](#)、[外來鍵](#)、[GROUP BY](#)、[JOIN](#)、[別名](#)。要建立檢視表前先熟知這些知識，才不會在撰寫 SQL 語法時感到混亂。
3. 在使用設計工具修改資料表結構時，可能會出現



不允許儲存變更。您所做的變更要求下列資料表必須先卸除然後再重新建立。您可能對無法重新建立的資料表做了變更，或是啟用了 [防止儲存需要資料表重建的變更] 選項。



Saving changes is not permitted. The changes you have made require the following tables to be dropped and re-created. You have either made changes to a table that can't be re-created or enabled the option Prevent saving changes that require the table to be re-created.

可依照 工具→選項→設計師→資料表和資料庫設計工具→防止儲存需要資料表重建的變更→取消勾選

Tools→Options→Designers→Table and Database Designers→Prevent saving changes that require table re-creation→取消勾選

不過實務上不建議這樣做，只是為了比賽中方便快捷更改資料表結構。

實務上需要使用 [ALTER TABLE](#) 變更資料表結構。

4. 在 SSMS 中，通常會使用工具創建資料表、建立資料庫關聯圖(創建外來鍵關係)，還可以創建檢視表，不過不常用，大多使用 SQL 語法[創建檢視表](#)。

先前提及[創建資料表](#)語法與增加限制語法常用在 Android Studio，目前並未提供 SQL 工具，故仍須了解其語法。

5. 測試時間可先將選項取消勾選，接著做其他測試。(三個科目共用測試時間)。

參、 C# (Visual Studio)

一、 推薦網站

1. [.Net 文件](#)
2. [C# 文件](#)
3. [C# 語言參考](#)
4. [C# 教程](#)

二、 教學影片

請參考原始文件

三、 程式碼

由於種類繁多，只將重要的程式碼片段收錄其中。若有問題無法解決，可至 [GitHub 儲存庫](#) 提出討論。

使用 .NET Framework 架構，專案類型為 Windows Forms App(.NET Framework)

1. 串接 API GET Method 取得字串

```
Ex:Uri = http://localhost:5000/api/UserVoice/GetRequestToken
HttpClient client = new HttpClient();
client.BaseAddress = new Uri("YourUri");
//client.BaseAddress = new Uri("http://localhost:5000");
HttpResponseMessage responseMessage =
client.GetAsync("YourUrl").Result;
//HttpResponseMessage responseMessage =
client.GetAsync("/api/UserVoice/GetRequestToken").Result;
if (responseMessage.IsSuccessStatusCode == true)
{
```

```
string result = responseMessage.Content.  
ReadAsStringAsync().Result;  
}
```

2. 串接 API POST Method Post json 字串

```
Ex:Uri = http://localhost:5000/api/UserVoice/ApplyUserVoice  
HttpClient client = new HttpClient();  
client.BaseAddress = new Uri("YourUri");  
// client.BaseAddress = new Uri("http://localhost:5000");  
StringContent content = new StringContent(post_string,  
Encoding.UTF8, "application/json");  
HttpResponseMessage responseMessage =  
client.PostAsync("YourUri", content).Result;  
//HttpResponseMessage responseMessage =  
client.PostAsync("/api/UserVoice/ApplyUserVoice",  
content).Result;  
if(responseMessage.IsSuccessStatusCode)  
{  
    string result =  
responseMessage.Content.ReadAsStringAsync().Result;//如果有回傳  
值  
}
```

3. 串接 API GET Method 取得圖片

與[取得字串](#)相似

```
//接續先前內容

if (httpResponseMessage.IsSuccessStatusCode == true)
{
    // 讀取回應內容作為 byte array
    byte[] imageBytes =
httpResponseMessage.Content.ReadAsByteArrayAsync().Result;

    // 使用 MemoryStream 將 byte array 轉換為 Bitmap
    using (MemoryStream ms = new MemoryStream(imageBytes))
    {
        Bitmap bitmap = new Bitmap(ms);
    }
}
```

4. Json 序列化 (C#物件轉 Json 字串)

必須先將 System.Text.Json 與 System.Memory 加入參考，加入的參考並不是第三方程式庫，故比賽中也可使用。

加入完參考後，必須在程式碼最前面 using 區塊，增加 using System.Text.Json;

```
//將 C# 物件序列化為 JSON 字串

string serializedJson = JsonSerializer.Serialize(YourObject);
```

或是

```
string serializedJson = JsonSerializer.Serialize(YourObject,
new JsonSerializerOptions { WriteIndented = true }); //將字串
增加縮排與換行，增加可讀性
```

更多資訊請參考 [JSON 序列化與反序列化](#)

5. Json 反序列化 (Json 字串轉 C#物件)

```
//反序列化 JSON 字串為 C# 物件  
YourClass obj =  
JsonSerializer.Deserialize<YourClass>(jsonString);
```

同 [Json 序列化](#)，必須先加入參考。更多資訊請參考 [JSON 序列化與反序列化](#)

6. 自定義類別 (Class)

在 Json 序列與反序列化中，會需要自定義類別，對應 Json 的格式。例如現在有個 Json 字串

```
{  
    "Name" : "wnilnay",  
    "Gender" : "Man",  
    "Age" : 18  
}
```

在 C#中就要自定義一個類別

```
public class Person  
{  
    public string Name { get; set; }  
    public string Gender { get; set; }  
    public int Age { get; set; }  
}
```

屬性名稱必須一樣，才能與之相互對應。若需要 C#屬性名與 Json 屬性名不同時，例如 "Body Mass Index": 17.5，因為屬性中包含空格，無法與 C#作對應，則自定義類別中可寫成：

```
[JsonPropertyName("Body Mass Index")]  
public float Body_Mass_Index { get; set; }
```

7. 實體資料模型

依照資料庫生成相應實體資料模型，可使用資料模型快速為資料表新增與移除資料。

根據資料庫關聯圖，可生成包括外部表格之資料表。類似使用[表格連接](#)，連接兩個資料表。

8. LINQ

請參考[LINQ 概觀](#)，裡面有更詳細的介紹。LINQ 內容非常多，無法全部詳細介紹，若有疑問，請參考官方文件。

- (1) 如果資料屬於實作 `IEnumerable<T>` 的型別或介面，則可以使用 LINQ to Objects 來查詢資料。包括但不限於 Array、List、DbSet...等。LINQ 相比於傳統資料查詢有著更高的可讀性與便利性，並且支援 IntelliSense。
- (2) 具體操作程式碼可參考[LINQ 概觀](#)。
- (3) LINQ 查詢可分為「查詢運算式」及「方法查詢」兩種，兩種查詢彼此等價且可互相轉換。具體可參考[LINQ 查詢](#)。
- (4) 下列範例示範簡單「查詢運算式」以及撰寫為「方法查詢」的語意對等查詢。範例來源：[LINQ 查詢](#)。
- (5) 方法查詢，方法內傳入 [Lambda](#) 運算式。

```
int[] numbers = [ 5, 10, 8, 3, 6, 12 ];

//Query syntax:
//可使用 var numQuery1 =
IEnumerable<int> numQuery1 =
    from num in numbers
    where num % 2 == 0
    orderby num
    select num;

//Method syntax:
//可使用 var numQuery2 =
IEnumerable<int> numQuery2 = numbers.Where(num => num % 2 ==
0).OrderBy(n => n);

foreach (int i in numQuery1)
{
    Console.Write(i + " ");
}
Console.WriteLine(System.Environment.NewLine);
foreach (int i in numQuery2)
{
    Console.Write(i + " ");
}
```

9. LINQ to SQL

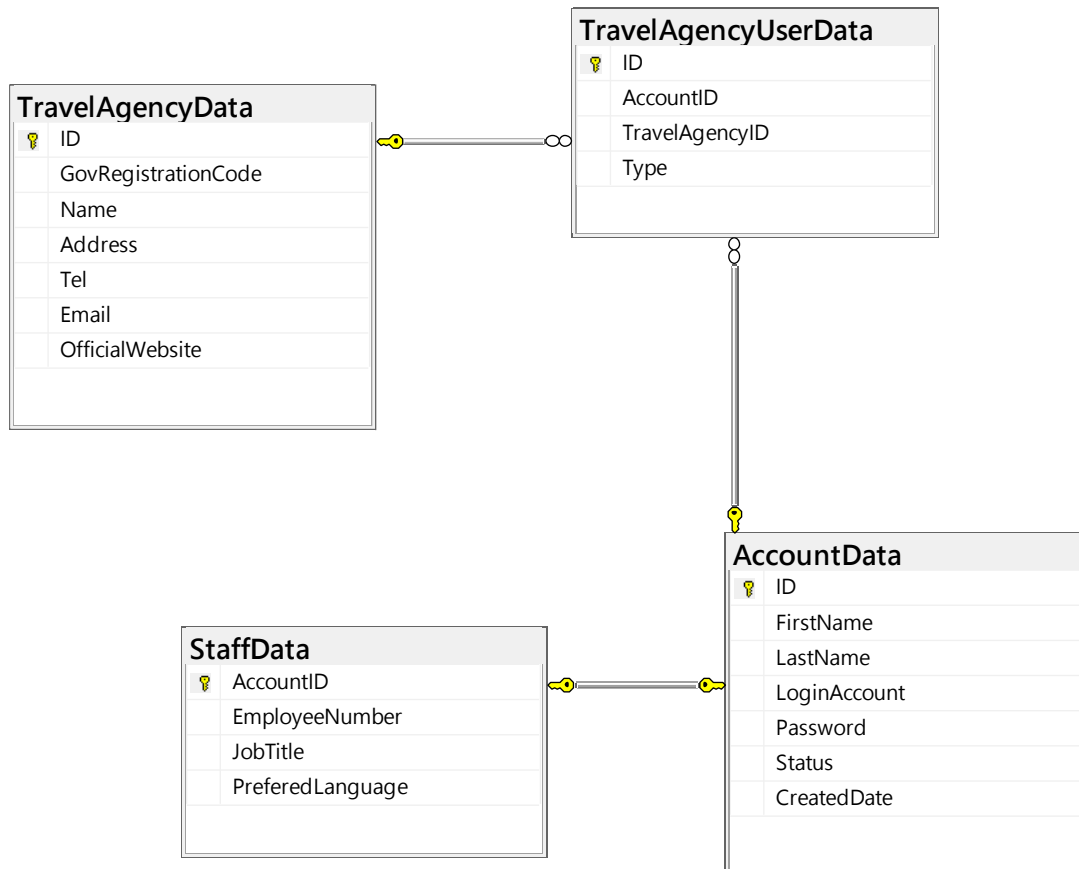
(1) 創建實體資料模型 YourEntities，YourEntities 內包含許多資料表，資料模型內的資料表屬於 DbSet<T>類別，故可用兩種 LINQ 語法，也支援其[擴充方法](#)。

(2) 程式碼

```
using (YourEntities db = new YourEntities())
{
    var table1 = (from a in db>YourTable
                  select a);
    foreach(var a in table1)
    {
        //Do something...
    }
}
```

LINQ 搭配資料庫關聯圖所生成的實體資料模型，可大幅增加篩選的便利性、增加篩選的速率。

例如現在有個實體資料模型 MyEntities，包括以下資料表，關聯圖也如下圖所示



若我現在想篩選某個帳號所屬的所有旅行社名稱，可使用

```

using (MyEntities db = new MyEntities ())
{
    var travelAgencyUserData = (from a in db.AccountData
                                where a.FirstName == "Maurice"
                                select a.TravelAgencyUserData).FirstOrDefault();
    foreach (var userData in travelAgencyUserData)
    {
        Console.WriteLine(userData.TravelAgencyData.Name);
    }
}
  
```

即可篩選出 FirstName 為 Maurice 的第一筆資料，其所屬旅行社名稱。

若要查看篩選結果的資料結構，可使用 LINQPad 查看。

10. 匿名觀念

C#中有幾種類型的資料結構或方法可以匿名使用，使用它們而無需明確地為它們命名或定義類型。以下是幾個主要的可以匿名的項目

(1) 匿名方法 (Anonymous Methods)，多用 Lambda 表達式

```
Func<int, int, int> add = delegate(int x, int y)
{
    return x + y;
};
```

(2) Lambda 表達式 (Lambda Expressions)，Lambda 表達式提供更精簡且更具表達性的方式來建立匿名函式。使用 [=> 運算子](#) 來建構 Lambda 運算式：

```
Func<int, int, int> add = (x, y) => x + y; //Lambda 運算式
Func<int, int, int> add = (x, y) =>
{
    return x + y;
}; //Lambda 陳述式
```

(3) 匿名物件 (Anonymous Types)，匿名物件允許你在不定義類型的情況下臨時創建物件。

```
var person = new { Name = "John", Age = 30 };
```

(4) 匿名陣列 (Anonymous Arrays)，匿名陣列允許你創建一個無名稱的陣列，陣列中的元素可以是不同類型。

```
var array = new object[] { "text", 42, true };
```

(5) 動態物件 (Dynamic Objects)，使用 `dynamic` 關鍵字，你可以創建一個類型在編譯時未定義的物件。(不常用)

(6) 隱含類型區域變數 (Implicitly Typed Local Variables)

使用 `var` 關鍵字，C# 允許你在編譯時由編譯器自動推斷變量的類型，而不是顯式聲明類型。

```
var number = 42; // 編譯器自動推斷為 int
var text = "Hello"; // 編譯器自動推斷為 string
```

(7) 匿名委派 (Anonymous Delegates)

你可以在使用匿名方法或 Lambda 表達式時創建匿名委派。這些委派可以在不定義具名委派的情況下直接使用。

```
Action greet = delegate { Console.WriteLine("Hello,
World!"); };
Action greet = () => { Console.WriteLine("Hello, World!"); };
greet();
//greet.Invoke();
```

11. Lambda 表達式

分為 Lambda 運算式、Lambda 陳述式

使用 「[=> 運算子](#)」，建立 Lambda 表達式，可使用 [Func](#)、[Action](#) 封裝

```
(input-parameters) => expression //Lambda 運算式
() => (); //Lambda 運算式
() => {}; //Lambda 陳述式
```

如果 Lambda 運算式只有一個輸入參數，括弧是選擇性的：

12. Func<>、Action、Action<>

皆是用於封裝方法，常用於 [Lambda 表達式](#) 封裝，當然也可封裝具名函式與具名方法。

Func<> 支援最多 16 個參數，一個回傳值。

Action 支援 0 個參數 0 個回傳值。Action<> 支援最多 16 個參數，沒有回傳值。

13. 擴充方法

擴充方法可讓您在現有類型中「加入」方法，而不需要建立新的衍生類型、重新編譯，或是修改原始類型。

通常要為類別增加方法，通常會使用繼承類別的方式擴充方法。但若不想繼承，或是類別本身無法被繼承，例如 string，就可以使用擴充方法。在靜態類別內建立靜態方法，參數內在要擴充類別前增加「this」。

最常見的擴充方法是 LINQ 標準查詢運算子。

使用第 54 屆分區賽歷屆試題，密碼雜湊功能作為範例程式

```
public static class ProcessSHA256
{
    public static string ExtensionProcess(this string value)//
擴充方法
    {
        return _Process(value);
    }
    public static string StaticProcess(string value)//靜態方法
    {
        return _Process(value);
    }
}
```

在調用方法上，擴充方法有較高的可讀性。

```
str = str.ExtensionProcess();//擴充方法  
str = ProcessSHA256.StaticProcess(str);//靜態方法
```

14. 顯示表格 DataTable DataGridView

DataGridView，常用來顯示表格資料，我的做法是建立 DataTable，操作 DataTable，將 DataGridView 資料來源設為 DataTable。

我將以 54 屆分區賽歷屆試題，帳號篩選功能需顯示的表格作為示範。

(1) 在 Form 應用程式中建立 DataGridView 物件，為他增加所需的標題，修改

控制項的屬性。我的作法：

加入、編輯、刪除皆不啟用，資料行唯獨，類型依需求改變

AllowDrop、AllowUser...、AllowUser...，全部 False

AutoSizeColumnsMode = Fill，AutoSizeRowsMode = AllCells

ColumnHeadersDefaultCellStyle、DefaultCellStyle、RowHeadersDefaultCellStyle

全部置中(MiddleCenter)，RowHeadersVisible = False

(2) 關鍵程式碼

```
DataTable dataTable = new DataTable();  
dataTable.Columns.Clear();dataTable.Rows.Clear();  
//增加 DataTable 標題欄，為 DataTable 標題欄與 DataGridView 標題欄作  
連接  
for(int i = 0; i < dataGridView1.Columns.Count; i++)  
{  
    dataTable.Columns.Add(dataGridView1.Columns[i].HeaderText);  
    dataGridView1.Columns[i].DataPropertyName =  
dataTable.Columns[i].ColumnName;  
}
```

```
//隨機增加 10 筆測資

for (int i = 0; i < 10; i++)
{
    //new object[]為匿名陣列，可方便增加資料行
    dataTable.Rows.Add(new object[]
    {
        name,
        accountStatus,
        updateTime,
        orderCount,
        ticketCount,
        effiectedCount,
        "查看"
    });
}

//將 DataGridView 資料來源設定成 DataTable
dataGridView1.DataSource = dataTable;
```

(3) 最後效果

全名	帳號狀態	資料更新時間	有效訂單數	有效總票券數	未到期的票券數	詳細資料
Jean Rauprich	正常	2024-08-12 10:45:15	6	15	14	查看
Tamqrah Rauprich	正常	2024-08-12 13:45:19	4	20	17	查看
Jean Kempster	正常	2024-08-12 16:51:15	0	0	0	查看
Markos McGucken	正常	2024-08-12 15:51:24	1	1	1	查看
Jean Rauprich	正常	2024-08-12 18:52:20	8	8	1	查看
Dana MacGow	正常	2024-08-12 09:52:15	6	8	5	查看
Tamqrah McGucken	正常	2024-08-12 13:47:24	1	1	1	查看
Maurice Kempster	正常	2024-08-12 14:51:15	6	12	6	查看
Dana Laverack	正常	2024-08-12 09:50:23	7	5	5	查看
Edythe McGucken	正常	2024-08-12 11:48:24	2	7	7	查看

15. 篩選表格 BindingSource、DataGridView

此節延續[顯示表格](#)，為表格進行篩選。

```
BindingSource bindingSource = new BindingSource();
dataGridView1.DataSource = bindingSource.DataSource =
dataTable;

//篩選與 SQL 篩選極為相似
//使用 like 進行模糊篩選，通常搭配 「%」或「_」萬用字元
bindingSource.Filter = "全名 like '%" + name + "%'";
//也可使用「=」
bindingSource.Filter = "有效總票券數 = '" + textBox2.Text + "'";
bindingSource.Filter = "有效總票券數 like '%" + textBox2.Text +
"%'";
//使用 AND 或是 OR 連接篩選條件
bindingSource.Filter = "全名 like '%" + name + "%' AND 有效總票
券數 = '" + textBox2.Text + "'";
```

16. 多線程 Task

載入表格資料大多需要進行許多操作與計算，大多為耗時任務，故運行在主線程會導致應用程式當機，因此需要多線程讓耗時任務運行在其他線程中。

要達成多線程有非常多種方法，這裡介紹我比較常用的方法「Task」。

```
Task.Run(() =>
{
    //Do something on other thread
});
//若需要對任務做些操作，可使用
Task task = Task.Run(() =>
{
    //Do something on other thread
});
```

Task.Run()方法內傳入一個 [Action](#) 或是 [Func](#)，也可用 [Lambda 表達式](#)，具體可參考 [Task.Run 方法](#)。

多線程也常與 Async、Await 一起使用，不過我不太熟悉，若有需要，可參考 [使用 async 和 await 進行非同步程式設計](#)

17. 委派 Invoke、Delegate

我對 Delegate 不是很熟悉，本節只針對多線程中使用 Invoke 回到 UI 線程。

在多線程中變更 UI 線程，會觸發 System.InvalidOperationException: 跨執行緒作業無效。這時會需要使用 Invoke 回到 UI 線程操作。

```
控制項.Invoke(new Action(() =>
{
    //Do something on UI thread
}));
```

Invoke()需要一個 Delegate 類別作為參數傳入，我這裡使用 [Action\(\)](#) 搭配 [Lambda](#) 創建一個匿名函式，當作參數傳入。除了使用 [Action](#)，也可使用 [Func](#) 或是 delegate 等函式。

具體程式碼可能略有不同，請依照情況進行修改。

```
Func<> func = 具名函式;

Func<> func = () => {};

Func<> func = () => ();

Func<> func = delegate {};

Action func = 具名函式;

Action func = () => {};

Action func = () => ();

Action func = delegate {};

控制項.Invoke(func);
```

四、經驗

以上程式碼，關於委派、Delegate、匿名委派、Func、Action 相關部分不是很熟悉，若有不清楚或錯誤的部分，請見諒。

若未來有更深的理解，將會在 [GitHub 儲存庫](#) 上更新，歡迎指教與討論。

上述內容有些部分較偏重理論，建議以實作為主，先試著自己動手做做看，若有餘力，再將理論部分補足即可。先以歷屆試題開始做起，有不理解的部分，再找資料解決。

關於 Language Integrated Query，縮寫：LINQ 音"link"，LINQ to SQL 可使用 [LINQPad](#) 查看結構與轉換後的 SQL 語句，這在學習過程中非常有用。

上述內容對於類別與物件提及較少，我認為這是最基本要會的，且在教學影片中，對於 C# 物件導向有著清楚的介紹，故在此不再贅述。

補充關於 Task.Run()、Invoke、表格操作

在 DataTable 設定成 DataGridView 的 DataSource 後，為 DataTable 增加資料，通常也會自動在 DataGridView 上進行更新。故可在一開始就做初始設定。

```
//可直接在多線程增加資料，幾乎都可以在 DataGridView 上正常更新
Task.Run(() =>
{
    dataTable.Rows.Add(new object[]
    {
        name,
        accountStatus,
        updateTime,
        orderCount,
        ticketCount,
        effiectedCount,
        "查看"
    });
});
```

//若無法正常更新，可將增加資料的操作移到 UI 線程，應該可以解決問題

```
Task.Run(() =>
{
    dataGridView1.Invoke(new Action(() =>
    {
        dataTable.Rows.Add(new object[]
        {
            name,
            accountStatus,
            updateTime,
            orderCount,
            ticketCount,
            effiectedCount,
            "查看"
        });
    }));
});
```

Visual Studio 環境設定，參考至[伊果的沒人看筆記本](#)

1. 字體推薦使用 Consolas
2. 擴充套件

(1) [CodeMaintainability 2022](#)

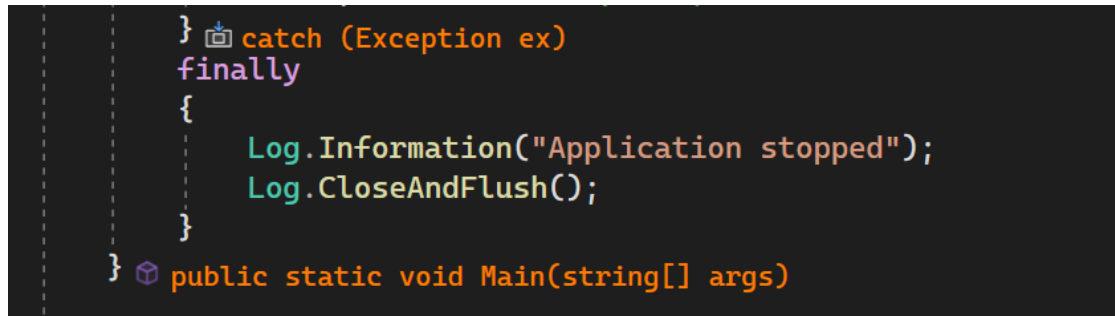
查看程式碼複雜度

(2) [Editor Guidelines](#)

在特定字元數位置畫輔助線，避免程式碼過長，若長度過長，請為程式碼換行。

(3) [CodeBlockEndTag](#)

括弧的結束部分會顯示出這個括弧所屬的區塊，如下圖所示

A screenshot of a code editor with a dark background. It shows a C# code snippet. A vertical dashed line is positioned at the start of the 'finally' block. The code is as follows:

```
    } catch (Exception ex)
    finally
    {
        Log.Information("Application stopped");
        Log.CloseAndFlush();
    }
} public static void Main(string[] args)
```

圖片來源：[CodeBlockEndTag](#)

3. Visual Studio 設定

開啟內嵌提示，可增加程式碼的可讀性。例如呼叫方法時顯示參數名稱，詳情可參考[伊果的沒人看筆記本](#)

在開發期間，常常會將變數輸出至 Console 查看，但 Visual Studio 輸出視窗還有其他系統訊息，常常找不到自己輸出的數值。這時可以點擊專案的 Properties，修改輸出類型為「主控台應用程式」，這時執行程式將會多出一個 Console 視窗，方便調適。

在開發完成後，或是比賽最後需要繳交檔案時，請將輸出類型修改回原本的類型。

測試時間可測試 API 串接、加入參考是否可正常運行

肆、 Android Studio (Java)

一、 推薦網站

1. [開發人員指南 | Android Developers](#)
2. [Android API reference | Android Developers](#)
3. [Android SDK | Android Developers](#)
4. [Android 教學](#)

二、 教學影片

請參考原始文件

三、 程式碼

以下程式碼皆為「Java」語言，不過 Google 漸漸轉用「Kotlin」進行 Android 開發。但應該不會相差太多。

專案類型為：Empty Views Activity

1. 串接 API GET Method 取得字串

要串接 API 之前，APP 必須取得網際網路權限

```
<!--AndroidManifest.xml-->

<manifest>

    <uses-permission android:name="android.permission.INTERNET"/>

</manifest>
```

Android 預設不支持「http://」，只支援「https://」，所以必須增加

```
<!--AndroidManifest.xml-->
<application
    android:usesCleartextTraffic="true"
    ...>
</application>
```

使用 HttpURLConnection 串接 API

```
URL url = new URL(YourUrl);
HttpURLConnection httpURLConnection = (HttpURLConnection)
url.openConnection();
//設定方法"GET"
httpURLConnection.setRequestMethod("GET");
//要接收回傳值
httpURLConnection.setDoInput(true);
//設定 Timeout 為 5000ms
httpURLConnection.setConnectTimeout(5000);
//開始連接
httpURLConnection.connect();
InputStream inputStream = httpURLConnection.getInputStream();
//InputStreamReader inputStreamReader = new
InputStreamReader(inputStream, Charset.forName("UTF-8"));
InputStreamReader inputStreamReader = new
InputStreamReader(inputStream);
```

```
BufferedReader bufferedReader = new
BufferedReader(inputStreamReader);
//BufferedReader bufferedReader1 = new BufferedReader(new
InputStreamReader(httpURLConnection.getInputStream()));
StringBuilder stringBuilder = new StringBuilder();
//斷開連接
httpURLConnection.disconnect();
if(httpURLConnection.getResponseCode() ==
HttpURLConnection.HTTP_OK){
    String line;

    while ((line = bufferedReader.readLine()) != null){
        stringBuilder.append(line).append("\n");
    }
}
```

- (1) 取得 InputStream 後，可直接使用 InputStream 的 read()方法搭配迴圈取得字元，連接成字串。具體可參考 [InputStream](#)
- (2) 為了讀取方便，最終可以使用 BufferedReader 讀取一整行，但 new BufferedReader()只接收 InputStreamReader，所以要將 InputStream 轉換成 InputStreamReader。
- (3) InputStreamReader 與 InputStream 類似。
- (4) 使用 StringBuilder 連接字串。
- (5) 可使用 stringBuilder.toString();取得連接後的字串。

2. 串接 API POST Method Post json 字串

與 GET 方法一樣，必須在 AndroidManifest.xml 進行設定(同個專案只要設定一次)

使用 HttpURLConnection 串接 API

```
URL url = new URL(YourUri);
HttpURLConnection httpURLConnection = (HttpURLConnection)
url.openConnection();
httpURLConnection.setConnectTimeout(5000);
//設定可輸入
httpURLConnection.setDoInput(true);
//設定可輸出
httpURLConnection.setDoOutput(true);
//設定請求方法 POST
httpURLConnection.setRequestMethod("POST");
//設定請求屬性
httpURLConnection.setRequestProperty("Content-
Type","application/json; Charset=UTF-8");
//取得 OutputStream
OutputStream outputStream =
httpURLConnection.getOutputStream();
OutputStreamWriter outputStreamWriter = new
OutputStreamWriter(outputStream, Charset.forName("UTF-8"));
BufferedWriter bufferedWriter = new
BufferedWriter(outputStreamWriter);
```

```
//BufferedWriter bufferedWriter1 = new BufferedWriter(new
OutputStreamWriter(httpURLConnection.getOutputStream()));

//寫入字串至緩衝區，以便稍後輸出
bufferedWriter.write(jsonString);

//將數據寫入目的地
bufferedWriter.flush();

//關閉 BufferedWriter
bufferedWriter.close();

if(httpURLConnection.getResponseCode() ==
HttpURLConnection.HTTP_OK){

    //若有回傳值
    InputStream inputStream = httpURLConnection.getInputStream();

    InputStreamReader inputStreamReader = new
    InputStreamReader(inputStream, Charset.forName("UTF-8"));

    BufferedReader bufferedReader = new
    BufferedReader(inputStreamReader);

    //BufferedReader bufferedReader1 = new BufferedReader(new
    InputStreamReader(httpURLConnection.getInputStream()));

    StringBuilder stringBuilder = new StringBuilder();

    String line;

    while ((line = bufferedReader.readLine()) != null){

        stringBuilder.append(line).append("\n");

    }

}

httpURLConnection.disconnect();
```


- (1) 取得 OutputStream 後，也可以直接使用 outputStream.write()，不過只能寫入字元。
- (2) 為了方便寫入資料，使用 BufferedWriter 寫入資料，但 BufferedWriter()只接收 OutputStreamWriter 作為參數，則需再將 OutputStream 轉為 OutputStreamWriter。
- (3) OutputStream 與 OutputStreamWriter 相似。
- (4) 若 POST 方法沒有回傳值，則不需撰寫接收回傳值的部分。
- (5) 使用 StringBuilder 連接字串。
- (6) 可使用 stringBuilder.toString();取得連接後的字串。

3. 串接 API GET Method 取得圖片

與[取得字串](#)相似

```
if(httpURLConnection.getResponseCode() ==  
HttpURLConnection.HTTP_OK){  
    InputStream inputStream =  
httpURLConnection.getInputStream();  
    Bitmap bitmap = BitmapFactory.decodeStream(inputStream);  
}
```

4. Json 序列化 JSONObject to JsonString

Java 中並沒有原生的方法將 Java 物件轉成 Json 字串。

不過 Java 中內建 JSONObject，可達到類似的效果。

```
JSONObject jsonObject = new JSONObject();  
jsonObject.put(key/name,value);
```

可多次 put，創建許多鍵值對。

若要取得 Json 字串，可使用

```
jsonObject.toString();
```

取得序列化後的 Json 字串。

5. Json 反序列化 JsonString to JSONObject

Java 中並沒有原生的方法將 Json 字串轉成 Java 物件。

不過 Java 中內建 JSONObject，可達到類似的效果。

```
JSONObject jsonObject = new JSONObject(jsonString);  
jsonObject.get(key/name);  
jsonObject.getInt(key/name);  
jsonObject.getString(key/name);
```

jsonObject 擁有許多 get 方法，可依照需求取得所需的資料。

6. try-catch Exception

在串接 API 時，避免 API 出現問題，造成程式不穩定，Android Studio 要求處

理例外情況，否則編譯器會報錯。例如，URLConnection 要求處理

IOException，JSONObject 要求處理 JSONException

catch 中可一次處理多個例外，每個例外使用「|」隔開。

```
catch (IOException | JSONException exception){  
    //Do something  
}
```

7. 多線程

在執行網路任務時，例如 API 串接，避免等待響應造成 UI 線程阻塞，網路任務必須執行在其他線程中，這並不是可選項，而是必要的，否則 APP 會拋出例外並且閃退。

在 Android 中，有許多方法可以達成多線程，這裡介紹較為簡單的方法

「Thread」

```
new Thread(new Runnable() {  
    @Override  
    public void run() {  
        //Do something on other thread  
    }  
}).start();
```

8. runOnUiThread

顧名思義，將下列程式碼，執行在 UI 線程中。在 Android Studio 中，要更新 UI 畫面，也必須在 UI 線程中執行。

```
new Thread(new Runnable() {  
    @Override  
    public void run() {  
        //Do something on other thread  
        runOnUiThread(new Runnable() {  
            @Override  
            public void run() {  
                //Do something on UI thread  
            }  
        })  
    }  
});
```

```
});  
  
}  
}).start();
```

Java 中，Timer 內的任務也屬於多線程。

而 C#中的 Timer_Tick 事件，依然屬於主線程。

9. Context

Android Studio 中的 Context 通常指的是當前的 Activity，可使用「this」，指向當前的 Activity 物件。

因為 Fragment 是依附在 Activity 中的物件，因此在 Fragment，可使用 `getContext()`、`getActivity()` 取得父層的 Activity。

10. Activity、Fragment、Context

以下內容不會提到有關於頁面跳轉與資料傳遞的內容，若有需要，請參考教學影片。

以下部分內容為我個人的理解與猜測，並不能保證絕對正確。

某些顯示元件，例如 Toast、AlertDialog...等等，都會需要傳入 Context 參數，這代表他們與 Fragment 一樣，都需要基於某個 Activity 才能工作，所以使用的 Context 大多是 Activity。

在 Fragment 中，若要使用 Toast、AlertDialog 等元件，則 Context 參數必須傳入其父層 Activity，可由 `getContext()`或是 `getActivity()`取得。

而在 Fragment 的多線程，要回到 UI 線程時，也必須使用其父層 `Activity.runOnUiThread()`方法。

四、經驗

在 Android Studio 中，有許多的顯示元件，這些元件的用法必須熟知。還有一些常用元件，例如 Spinner(下拉式選單)、ListView(清單)、RecyclerView(客製化的 ListView)，如有需要，請上網查詢使用方法。

Android Studio 建議先將重要的教學影片看完，接著直接開始練習歷屆試題，Android 理論部分我覺得較少，大部分都是基於前人的成果進行修改。我認為這部分最重要的是了解有哪些工具可以使用，使用方法又是如何。

在 Android Studio 操作 SQLite 資料庫時，有時需撰寫 SQL 語法，可在撰寫字串時(雙引號中間)，按下鍵盤「Alt + Enter」，選擇「Inject Language or reference」，搜尋 SQL，選擇「SQLITE-SQL」，即可擁有 SQL Intellisense 與標示關鍵字的功能。

(此選項為新版擁有的選項，比賽現場與選手室電腦不一定會有，在測試環境時可先測試)

若無「SQLITE-SQL」選項，也可選擇「RoomSql」，雖無 Intellisense 功能，但仍然會標示關鍵字。

若需要撰寫其他語言，例如 Json，也可以使用該方法，看有沒有支援。

Java 中，物件屬性的 getter、setter 往往使用「get 屬性名」、「set 屬性名」方法。在初次使用物件時，可使用先撰寫「物件.get」，來查看此物件有什麼屬性。

Android 建議使用實機進行開發，避免效能瓶頸，造成開發速度緩慢。

API 的部分，使用虛擬機，若是 localhost 網址，則需將 localhost 改成 10.0.2.2。

實機連接 localhost API

若要連接自己建立的 API，可將 API 發布後，部署到 IIS，實機與電腦連接到同一個網際網路。開啟防火牆後，即可使用 `http(https)://電腦 IP:連接埠/路徑`，至於使用 `http` 或是 `https`，取決於一開始發布在 IIS 使用的類型。

要取得電腦 IP，可在 cmd 中撰寫指令「`ipconfig`」，查看 IPv4 地址。

使用上述方法時，必須增加防火牆輸入規則，允許特定的連接埠輸入規則。

輸入規則→新增規則→連接→TCP→建議使用特定本機連接埠→輸入 IIS 中設定 API 的連接埠→允許連接→輸入可辨識的名稱→完成。

歷屆試題中的 API，若仍然需要使用實機進行連接，可使用「[ngrok](#)」，讓外部的網址指向 api 的 localhost 地址，即可使用實機連結到 API，此方法建議只在開發時期使用。

若是自己建立的 API，也可以發布後，部署到 IIS，使用 [ngrok](#) 讓外網連接。

使用 [ngrok](#) 指向 localhost 地址，可以不需要連接到同一個網際網路，只要能夠連上網際網路，即可連接到 API。

比賽測試時間可先測試虛擬機是否能夠正常開啟，Android Studio 撰寫頁面設計的部分有沒有 Intellisense，應該會有，也應該要有。

伍、 Web API (全國賽)

此章節講解內容為全國賽考科。

以下內容使用專案類型為：ASP.NET Web 應用程式 (.NET Framework)

使用架構為：.NET Framework 4.8。或許未來會更換架構為.NET Core

若需更多詳細資料，可參考 [Web API](#)

一、 前言

API 常用的方法與資料庫相同。增刪查改 (CRUD)。

對應到 API 則為

查詢 (Read) → GET Method

增加 (Create) → Post Method

改正 (Update) → Put Method

刪除 (Delete) → Delete Method

API 操作常常與資料庫有關，但 API 操作並不只用在資料庫上。

二、 程式碼

1. 新增控制器

Controllers 資料夾上按右鍵→加入→控制器→Web API→具有讀取/寫入動作的 Web API 2 控制器。

2. 實體資料模型

創建實體資料模型，方便後續操作。創建實體資料模型時，驗證方式建議使用 SQL Serve 驗證。使用 Windows 驗證，部署到 IIS 會有權限不足的問題。
SQL Server 驗證中也需選擇「在連接字串中包含敏感性資料」

3. 使用 Json 格式作為回傳值

```
//Global.asax  
  
//新增以下程式碼，清除 Xml 格式。  
  
GlobalConfiguration.Configuration.Formatters.XmlFormatter.SupportedMediaTypes.Clear();
```

4. GET Method 回傳資料

```
//原始方法  
  
// GET api/values  
public IEnumerable<string> Get()  
{  
    return new string[] { "value1", "value2" };  
}  
  
// GET api/values/5  
public string Get(int id)  
{  
    return "value";  
}
```

可與實體資料模型做搭配，假設我要回傳所有帳號的姓名組合

```
// GET: api/DataBase  
public IEnumerable<string> Get()  
{  
    using(For_TestEntities db = new For_TestEntities())  
    {
```



```
        var name = (from a in db.AccountData
                      select a.FirstName + " " + a.LastName).ToList();
        return name;
    }
}
```

5. GET Method 回傳圖片

```
// GET api/values
public HttpResponseMessage Get()
{
    Bitmap bitmap =
ImageResource.it_software_solutions_for_business;
    HttpResponseMessage response = new HttpResponseMessage();

    using(MemoryStream memoryStream = new MemoryStream())
    {
        bitmap.Save(memoryStream, ImageFormat.Png);
        response.Content = new
ByteArrayContent(memoryStream.ToArray());
        response.Content.Headers.ContentType = new
MediaTypeHeaderValue("image/png");
        return response;
    }
}
```

若需要串接 API 取得圖片，可依照

[C#串接 API 取得圖片](#)


[Android Studio 串接 API 取得圖片](#)

6. POST Method 新增資料

```
//原始方法  
  
// POST api/values  
  
public void Post([FromBody] string value)  
{  
  
}
```

FromBody，強制 Web API 從要求本文讀取類型，Web API 會從文本讀取字串，並且存入 value 變數。

以下表為例，若需要使用 API 為 AccountData 資料表增加資料，則：

	Column Name	Data Type	Allow Nulls
	ID	uniqueidentifier	<input type="checkbox"/>
	FirstName	nvarchar(100)	<input type="checkbox"/>
	LastName	nvarchar(100)	<input type="checkbox"/>
	LoginAccount	nvarchar(20)	<input type="checkbox"/>
	Password	varchar(50)	<input type="checkbox"/>
	Status	int	<input type="checkbox"/>
	CreateDate	datetime	<input type="checkbox"/>

```
public class PostClass  
{  
  
    public string FirstName { get; set; }  
    public string LastName { get; set; }  
    public string LoginAccount { get; set; }  
    public string Password { get; set; }  
  
}
```

我設定使用者只需傳入以上四個屬性。

```

// POST: api/DataBase

//若無回傳值，使用 void

public AccountData Post([FromBody] PostClass postClass)
{
    using(For_TestEntities db = new For_TestEntities())
    {
        AccountData accountData = new AccountData();
        Guid id = Guid.NewGuid();
        accountData.ID = id;
        accountData.FirstName = postClass.FirstName;
        accountData.LastName = postClass.LastName;
        accountData.LoginAccount = postClass.LoginAccount;
        accountData.Password = postClass.Password;
        accountData.Status = 1;
        accountData.CreatedDate = DateTime.Now;
        db.AccountData.Add(accountData);
        db.SaveChanges();

        //回傳
        var account = (from a in db.AccountData
                        where a.ID == id
                        select a).FirstOrDefault();

        return account;
    }
}

```

7. Put Method 修改資料

```
//原始方法  
  
// PUT api/values/5  
  
public void Put(int id, [FromBody] string value)  
{  
  
}
```

接續 POST 案例，若我想做個修改密碼的 API 功能，則：

使用者只需輸入三個參數，帳號、密碼、欲更改的密碼。

API 回傳成功、失敗兩種狀態之一。若是成功修改，則附帶更改後的帳號資訊。

```
public class PutClass  
{  
    public string LoginAccount { get; set; }  
    public string Password { get; set; }  
    public string NewPassword { get; set; }  
}  
  
public class PutResponse : AccountData  
{  
    public string ResponseStatus { get; set; }  
}
```

```

// PUT: api/DataBase

public PutResponse Put([FromBody]PutClass putClass)
{
    using(For_TestEntities db = new For_TestEntities())
    {
        var accountData = (from a in db.AccountData
                           where a.LoginAccount == putClass.LoginAccount &&
                                 a.Password == putClass.Password
                           select a).FirstOrDefault();

        PutResponse response = new PutResponse();
        if (accountData != null)
        {
            accountData.Password = putClass.NewPassword;
            db.SaveChanges();
            response.ResponseStatus = "Success";

            var changedAccountData = (from a in db.AccountData
                                     where a.LoginAccount == putClass.LoginAccount &&
                                           a.Password == putClass.NewPassword
                                     select a).FirstOrDefault();

            response.ID = changedAccountData.ID;
            response.FirstName = changedAccountData.FirstName;
            response.LastName = changedAccountData.LastName;
            response.LoginAccount =
changedAccountData.LoginAccount;

```

```
        response.Password = changedAccountData.Password;
        response.Status = changedAccountData.Status;
        response.CreatedDate = changedAccountData.CreatedDate;
        response.StaffData = changedAccountData.StaffData;
        response.TravelAgencyUserData =
changedAccountData.TravelAgencyUserData;
    }
    else
    {
        response.ResponseStatus = "Fail";
    }
    return response;
}
}
```

8. Delete Method 刪除資料

```
//原始方法  
  
// DELETE api/values/5  
public void Delete(int id)  
{  
  
}
```

HTTP 規範中，沒有明確禁止或允許 Delete 方法中是否可以包含 Body。

但有不少人認為 Delete 方法應該要允許包含 Body。

不過在某些伺服器中，包含 Body 可能會使伺服器無法正常運作。

在實務上，會偏向使用隱藏的方式，而不是直接刪除。

接續先前案例，使用 API 刪除一筆資料。

以下示範兩種方法，並且示範如何呼叫。

(1) 使用 Route(建議)

```
[HttpDelete]  
  
[Route("api/DatabaseDelete/{LoginAccount}/{Password}")]  
public object Delete(string LoginAccount, string Password)  
{  
    using (For_TestEntities db = new For_TestEntities())  
    {  
        var accountData = (from a in db.AccountData  
                            where a.LoginAccount == LoginAccount &&  
                                  a.Password == Password  
                            select a).FirstOrDefault();
```

```

        if (accountData == null)
        {
            var response = new { Status = "Fail" };
            return response;
        }
        else
        {
            db.AccountData.Remove(accountData);
            db.SaveChanges();
            var response = new { Status = "Success" };
            return response;
        }
    }
}

```

設定 Route，從 Uri 中取得 LoginAccount 與 Password，例如

<http://localhost:5147/api/database/wnilnay/NewPassword>

則，LoginAccount = wnilnay，Password = NewPassword

如何發出 Delete 請求

```

HttpClient httpClient = new HttpClient();
httpClient.BaseAddress = new Uri("http://localhost:5147");
httpClient.Timeout = new TimeSpan(0, 0, 5);
DeleteClass deleteClass = new DeleteClass();
deleteClass.LoginAccount = "wnilnay";
deleteClass.Password = "NewPassword";

```



```

HttpResponseMessage response = httpClient.DeleteAsync
($" /api/databaseDelete/{deleteClass.LoginAccount}/{deleteClass
.Password}").Result;

if (response.IsSuccessStatusCode)
{
    string result =
response.Content.ReadAsStringAsync().Result;
    Console.WriteLine(result);
}
else
{
    Console.WriteLine(response.StatusCode);
    Console.WriteLine(response.Content.ReadAsStringAsync()
.Result);
}

```

(2) 使用 Body

```

// DELETE: api/DataBase
public object Delete([FromBody] DeleteClass deleteClass)
{
    using (For_TestEntities db = new For_TestEntities())
    {

```

```
var accountData = (from a in db.AccountData
    where a.LoginAccount == deleteClass.LoginAccount &&
    a.Password == deleteClass.Password
    select a).FirstOrDefault();
if(accountData == null)
{
    var response = new { Status = "Fail" };
    return response;
}
else
{
    db.AccountData.Remove(accountData);
    db.SaveChanges();
    var response = new { Status = "Success" };
    return response;
}
}
```

在 HttpClient 的 DeleteAsync 方法中，無法傳入 Body。因此需要換個方法。

使用 HttpRequestMessage 可以達到我的需求。

```
DeleteClass deleteClass = new DeleteClass();

deleteClass.LoginAccount = "wnilnay";

deleteClass.Password = "NewPassword";


string json_delete = JsonSerializer.Serialize(deleteClass);


HttpRequestMessage request = new
HttpRequestMessage(HttpMethod.Delete,
"http://localhost:5147/api/database");

request.Content = new
StringContent(json_delete, Encoding.UTF8, "application/json");

HttpClient httpClient = new HttpClient();

HttpResponseMessage response =
httpClient.SendAsync(request).Result;


if (response.IsSuccessStatusCode)
{
    string result =
response.Content.ReadAsStringAsync().Result;

    Console.WriteLine(result);
}
```

```
else
{
    Console.WriteLine(response.StatusCode);

    Console.WriteLine(response.Content.ReadAsStringAsync()
        .Result);
}
```

9. Body

Post、Put 方法，需傳入 Body，通常傳入 Json 格式，傳入的 Json 字串會自動反序列化為一個物件。

且若 API 方法回傳一個物件，則也會自動序列化為 Json 字串。

10. 路由

預設路徑：http://localhost:連接埠/api/Controllers(/id)

主要變化在 Controllers 中，若新建一個 Controllers，叫做 BasicControllers，則

預設路徑為/api/basic(/id)

在程式碼中，可以使用[Route("路由")]更改路徑，例如

```
[Route("api/BasicGet/{id}")]
public void Func(int id)
{
}
```

使用以下網址：http://localhost:5000/api/basicGet/5

不管是使用何種請求，都會導向 Func 函式，且 id 為 5。

若限定特定請求方法，可以使用 [HTTP 方法](#)

11. HTTP 方法

包含

```
[HttpGet]、[HttpPost]、[HttpPut]、[HttpDelete]...
```

等等。

可限定特定請求方法，可與 [Route](#) 一起使用。

寫於函式前

```
[HttpGet]
[Route("api/BasicGet/{id}")]
public void Func(int id)
{
}
```

12. 發佈 Web API

在專案上按右鍵→發佈→資料夾→選擇資料夾位置→完成

完成後，必須點擊發佈，才會將專案發佈至資料夾。

13. 部署到 IIS

(1) 安裝 IIS

開啟或關閉 Windows 功能→勾選「Internal Information Services」

Internal Information Services→World Wide Web 服務→應用程式開發工具→

勾選「ASP.NET 4.8」

(2) 開啟 IIS

開啟 Internal Information Services (IIS) 管理員，左邊連線選擇

電腦名稱(電腦名稱\使用者)。若右側動作為啟動狀態，則可以至瀏覽器進

入網站「<http://localhost:80>」，若能成功進入網站，說明已成功開啟 IIS。

(3) 部署 API 至 IIS

左側選擇站台→右側新增網站→設定站台名稱、實體路徑、連接埠

實體路徑為先前[發佈](#)至資料夾的資料夾路徑。

(4) 測試是否部署成功

開啟瀏覽器，進入網址「http://localhost:連接埠」，若能進入網站，則說明部署成功。

三、 經驗

以上 Web API 與請求專案(Windows Forms App)程式碼，我將放置在另一個[GitHub 儲存庫](#)中，不過專案在另一台電腦中可能因為環境不同而無法正常執行，不過程式碼均無錯誤，主要是對上述 Web API 程式碼內容進行總結。

我對於這個科目練習的時間不多，沒有特別的經驗可以分享。

主要經驗內容與[C#經驗分享](#)相似。

陸、 總結

商務軟體設計需要學習的技能很多，此文件內容短時間內不需要全部理解。

以實作為優先，了解這些工具該如何使用，該如何變化，如何運用在一起。

C#的部分可以先試著做出歷屆試題全功能，若有需要補充理論，可再觀看教學影片或其他介紹。

Android Studio 建議先觀看教學影片重要的部分，例如畫面切換或是元件的使用，再去練習歷屆試題。若沒有足夠的基礎，在練習歷屆試題時容易處處碰壁。

無論是分區賽，還是全國賽，都建議自備鍵盤帶過去，分區賽可能會遇到拿青軸，專門干擾其他選手的注意力，得保持專注，不要被外界影響。