

# Team03 Project Report

## Main idea

我們注意到了在同學間，有許多人都會不小心收到超速罰單，因此我們希望藉由這個網站來提高交通安全意識，幫助大家了解測速照相機的位置，從而減少因超速造成的財產損失。同時，這樣的工具可以提供便利，讓大家更容易查詢相關資訊，避免因不熟悉路況而收到罰單，希望可以用技術手段來改善交通秩序。

## Dataset

資料集：測速執法設置點

資料來源：政府資料開放平台

資料介紹：提供有關全台灣測速執法位置的詳細資訊，涵蓋測速執法區域，幫助相關單位監控交通違規行為並改善道路安全。這份資料對公眾使用至關重要，能讓駕駛者了解執法區域，進而調整駕駛習慣。

## Data Table Schema

Table: camera

Field	Type	Null	Key	Default	Extra
Addr	varchar(255)	YES		NULL	
Direct	varchar(255)	YES		NULL	
Limits	int	YES		NULL	

Table: ps

Field	Type	Null	Key	Default	Extra
CityName	varchar(255)	YES		NULL	
RegionName	varchar(255)	YES		NULL	
Addr	varchar(255)	YES		NULL	
DeptNm	varchar(255)	YES		NULL	
BranchNm	varchar(255)	YES		NULL	

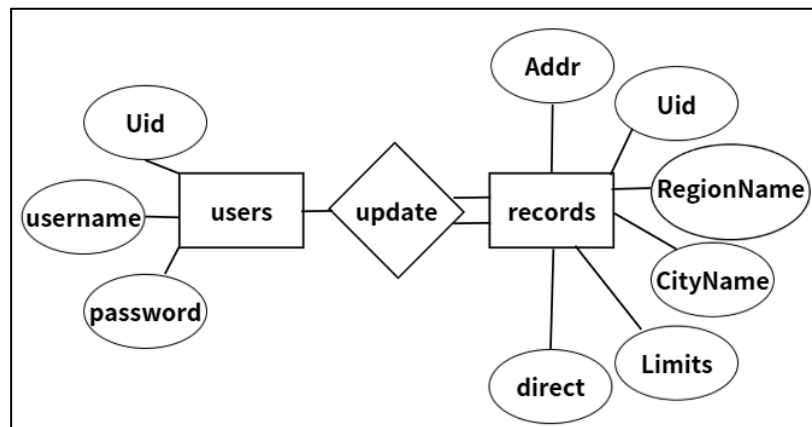
Table: records

Field	Type	Null	Key	Default	Extra
Uid	int	YES		NULL	
CityName	varchar(255)	YES		NULL	
RegionName	varchar(255)	YES		NULL	
Addr	varchar(255)	YES		NULL	
Direct	varchar(255)	YES		NULL	
Limits	int	YES		NULL	

Table: users

Field	Type	Null	Key	Default	Extra
Uid	int	NO	PRI	NULL	auto_increment
username	varchar(255)	NO	UNI	NULL	
password	varchar(64)	NO		NULL	

## ER Diagram



此ER Model包含了users以及records兩個tables, 並且用一對多的relationship來連結, 因為一筆record只會由一個user建立, 但一個user可以有許多筆records。

users:

用來記錄用戶所設置帳號的資料

Uid: 每個用戶獨有的編號, 用以區分各用戶, 並且用以做資料連結

username: 用戶創建帳號時自行輸入的名稱

password: 用戶創建帳號時設置的密碼

records:

用來記錄用戶對資料做的變更(更新速限、新增相機、刪除相機)

Addr: 做變更的相機地址

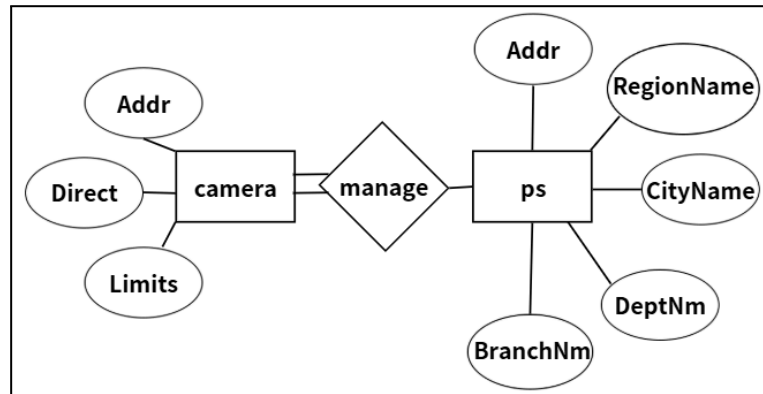
Uid: 用以與users table做連結

RegionName: 做變更的相機所在地區

CityName: 做變更的相機所在縣市

Limits: 做變更後的相機速限(若為刪除則為null)

Direct: 做變更後的相機拍攝方向



此ER Model包含camera及ps(警局)兩個tables, 並且用一對多的relationship來連結, 因為一個警局可以管理多個相機, 但一個相機只隸屬於一個警局。

camera:

用來記錄各相機的資訊

Addr: 相機所在地址

Direct: 相機拍攝方向

Limits: 相機速限

ps:

用來記錄各警局的資訊

Addr: 用以與camera做連結

RegionName: 警局所在地區

CityName: 警局所在縣市

DeptNm: 管轄警局名稱

BranchNm: 管轄分局名稱

## Database

Database selected:

我們這次採用的database為教授在上課時所推薦的MySQL。MySQL是一種關聯式資料庫管理系統(RDBMS), 它使用了SQL作為其主要的數據操作語言。MySQL是一種開源

的關聯式資料庫管理系統，所以任何人都可以免費使用，是Web開發中最常用的資料庫之一。

Maintain：

新增資料：

使用 /add\_camera 路由來新增資料，接收來自用戶的 JSON 格式請求。  
將資料插入 camera 和 ps 資料表，並記錄到 records 表中以追蹤操作。  
例如，新增攝影機的資訊，包括地址 (Addr)、方向 (Direct)、速限 (Limits) 等。

更新資料：

使用 /update\_camera 路由來更新資料，接收來自用戶的 JSON 格式請求。  
更新 camera 資料表中的速限 (Limits)，同時在 records 表中記錄操作。  
確保只有登錄用戶能執行更新操作。

刪除資料：

使用 /delete\_camera 路由來刪除資料，接收 JSON 格式請求。  
刪除 camera 資料表中的指定記錄，並在 records 表中記錄刪除操作的詳情。

查看歷史紀錄：

使用 /get\_update\_history 路由來查詢歷史更新紀錄。  
從 records 表中提取特定用戶的操作歷史，包含修改的縣市 (CityName)、地區 (RegionName)、地址 (Addr) 等資訊。

Database connection：

資料庫設定：

在程式開頭定義 db\_config，其中包含資料庫的連接參數：  
host: 資料庫伺服器位置 (localhost)。  
user: 資料庫使用者帳號 (dbuser)。  
password: 使用者密碼 (0420)。  
database: 資料庫名稱 (final)。

```
db_config = {  
    'host': 'localhost',  
    'user': 'dbuser',  
    'password': '0420',  
    'database': 'final'  
}
```

連接函數：

使用 mysql.connector 與資料庫建立連線

```
def get_db_connection():  
    return mysql.connector.connect(**db_config)
```

查詢處理流程：

### 1. 從應用接收請求

後端通過 **Flask** 接受用戶發出的 **GET** 或 **POST** 請求，並解析其數據  
如 /add\_camera 使用以下程式接受 JSON 格式的請求

```
data = request.json  
city = data.get('CityName')  
region = data.get('RegionName')  
addr = data.get('Addr')  
limits = data.get('Limits')  
direction = data.get('Direct')
```

## 2.資料庫操作

根據請求內容生成SQL查詢。

先建立資料庫連線：

```
conn = get_db_connection()
cursor = conn.cursor()
```

然後執行query：

```
insert_query_camera = """
INSERT INTO camera (Addr, Direct, Limits)
VALUES (%s, %s, %s)
"""
```

```
cursor.execute(insert_query_camera, (addr, direction, limits))
```

## 3.回應用戶請求

query執行完成後，向用戶返回結果

若成功：

```
flash("新增成功！", "success")
```

若失敗：

```
flash(f"新增失敗：{str(e)}", "danger")
```

例外處理：

每一個資料庫的操作(如新增、更新、刪除等)，都會被包在try-except-finally中以/update\_camera為例：

會先試著執行try區塊中的指令

```
try:
    update_query = """
    UPDATE camera
    SET Limits = %s
    WHERE Addr = %s
    """
    cursor.execute(update_query, (new_limit, addr))

    log_query = """
    INSERT INTO records (Uid, CityName, RegionName, Addr, Direct, Limits)
    VALUES (%s, %s, %s, %s, %s, %s)
    """
    cursor.execute(log_query, (session['user_id'], city, region, addr, direct, new_limit))

    conn.commit()
    flash("更新成功", "success")
    return redirect("/update")
```

若是執行錯誤或無法執行，則會執行rollback，並且彈出錯誤訊息

```
except Exception as e:
    conn.rollback()
    flash(f"更新失敗：{str(e)}", "danger")
    return redirect("/update")
```

最後會執行finally區塊中的指令

```
finally:  
    cursor.close()  
    conn.close()
```

防範不預期操作：

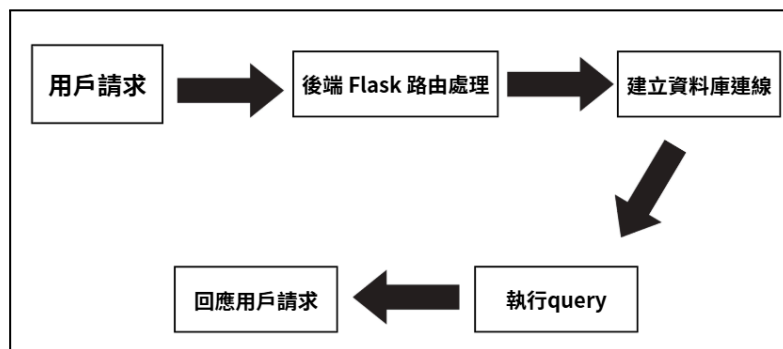
為避免用戶在未進行登入的情況下就做新增刪除等操作，我們在每個route中都會檢查用戶Session：

```
if 'user_id' not in session:  
    flash("請先登入", "warning")  
    return redirect("/update")
```

我們會驗證用戶的輸入資料是否完整，如在/get\_camera中檢查請求是否包含必要的欄位，若資料不完整，則返回錯誤訊息：

```
if not city:  
    return jsonify({"error": "請選擇縣市(區域)"}), 400
```

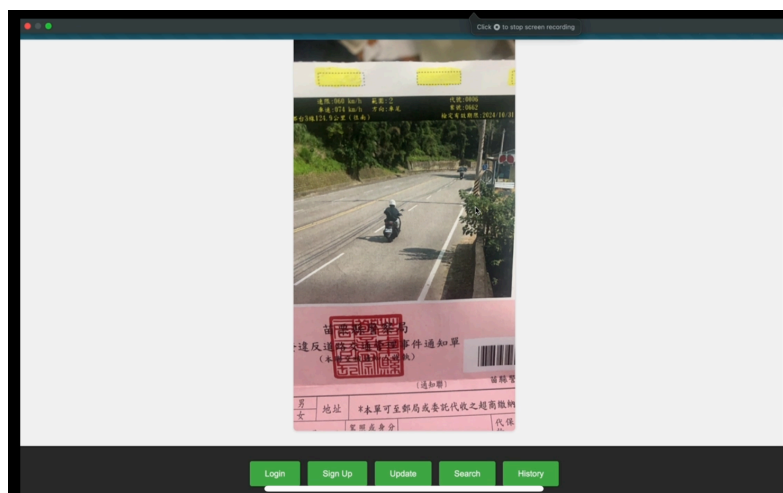
流程圖示：



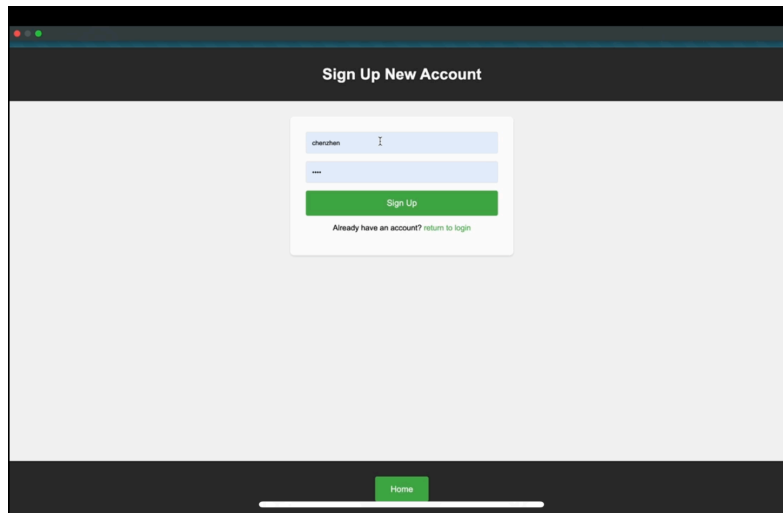
## Application

Interface：

Home page: 可以連接到sign up, login, search, update, history等頁面

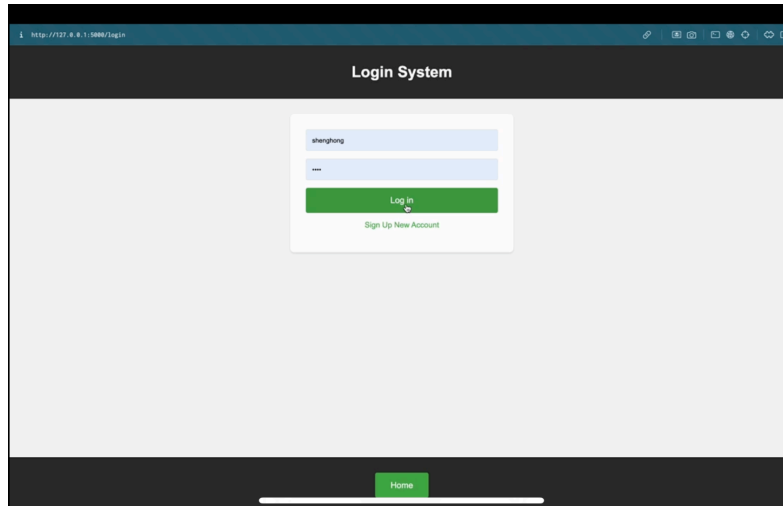


Sign up page: 可讓用戶進行註冊



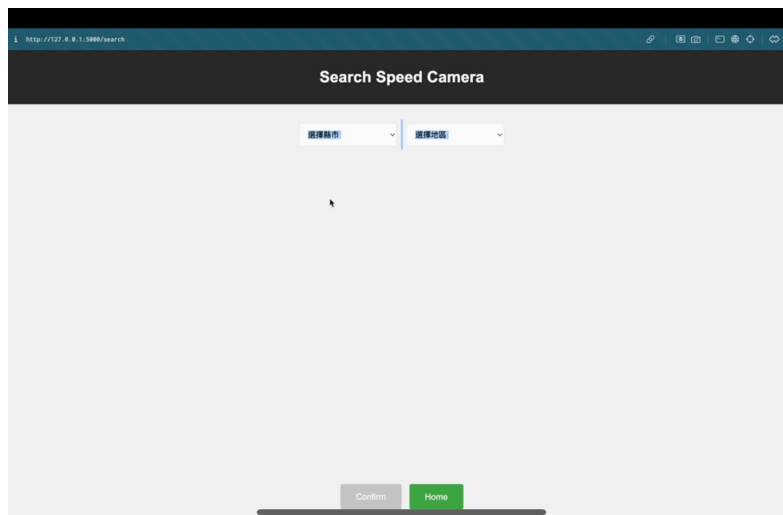
The image shows a mobile application interface for signing up a new account. The title bar at the top is dark blue with the text "Sign Up New Account" in white. The main content area is light gray. In the center, there is a white card with a light blue border. Inside the card, there are two input fields: the first contains the text "chenchen" and the second contains "1234". Below the input fields is a green button with the text "Sign Up" in white. Underneath the button, there is a link that says "Already have an account? return to login". At the bottom of the screen, there is a dark blue bar with a green button labeled "Home" in white.

Login page: 可讓用戶進行登入



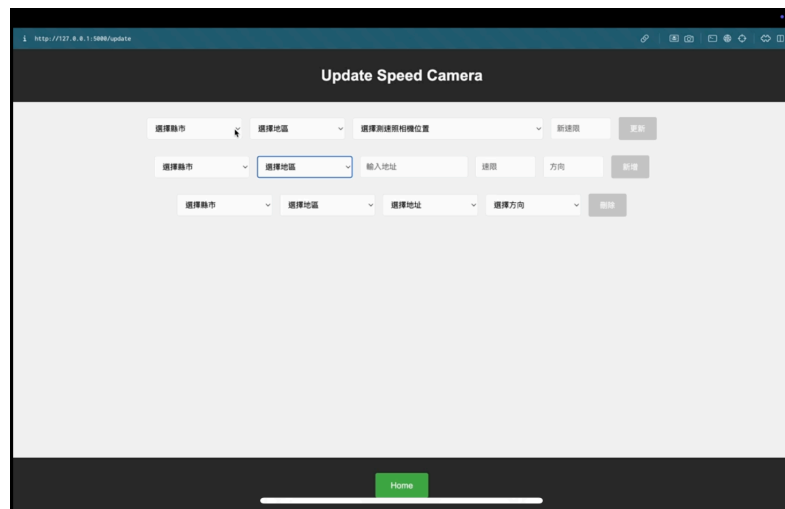
The image shows a mobile application interface for logging in. The title bar at the top is dark blue with the text "Login System" in white. The main content area is light gray. In the center, there is a white card with a light blue border. Inside the card, there are two input fields: the first contains the text "shenghong" and the second contains "1234". Below the input fields is a green button with the text "Log in" in white. Underneath the button, there is a link that says "Sign Up New Account". At the bottom of the screen, there is a dark blue bar with a green button labeled "Home" in white.

Search page: 可讓用戶針對縣市(及地區)查詢照相機位置

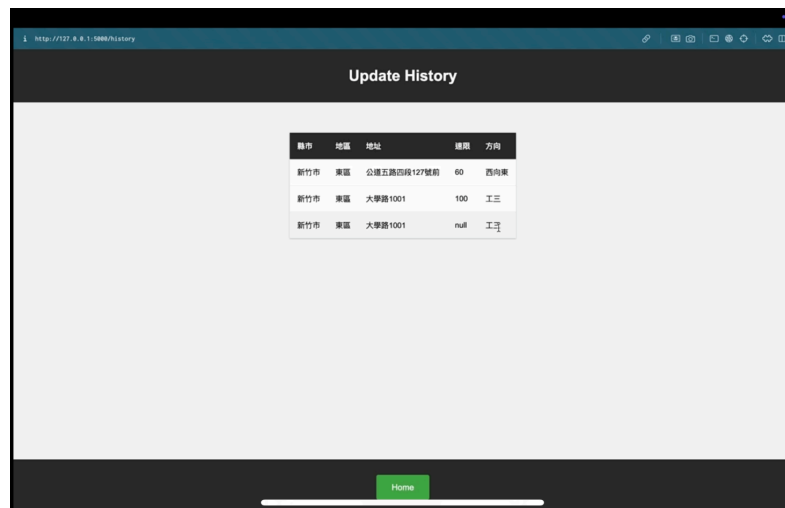


The image shows a mobile application interface for searching speed camera locations. The title bar at the top is dark blue with the text "Search Speed Camera" in white. The main content area is light gray. In the center, there are two dropdown menus: the first is labeled "選擇縣市" and the second is labeled "選擇地區". Below the dropdown menus, there is a small mouse cursor icon. At the bottom of the screen, there is a dark blue bar with two buttons: a gray button labeled "Confirm" and a green button labeled "Home" in white.

Update page: 可讓用戶執行新增、更新、刪除照相機等動作



History page: 用戶可以看到他曾對哪些做過更動



縣市	地區	地址	速限	方向
新竹市	東區	公庫五路市段127號前	60	西向東
新竹市	東區	大學路1001	100	工三
新竹市	東區	大學路1001	null	工三

Funtion:

Create:

- 1.新增用戶(signup)
- 2.新增照相機(add\_camera)
- 3.新增更新紀錄

Read:

- 1.找出所選縣市(及地區)的照相機, 並顯示相關資訊(get\_cameras)
- 2.可查看自己對照相機做過的更動紀錄(get\_update\_history)

Update:

- 1.使用者可更新現有照相機的速限(update\_camera)



Delete:

1.刪除現有照相機(delete\_camera)

Function implement:

**signup:**

用戶會先透過 /signup route 發送 POST 請求, 包含用戶名與密碼, 然後密碼再經過 SHA-256 加密後存入資料庫。

SQL query:

```
INSERT INTO users (username, password) VALUES (%s, %s)
```

設計原因:

使用參數化查詢 (%s) 防範 SQL 注入。  
為避免用戶名重複, 資料表設有唯一鍵約束。

例外處理:

當 IntegrityError 發生(如違反唯一鍵約束), 返回錯誤訊息給用戶, 並保持用戶在註冊頁面。

**get\_cameras:**

用戶透過 /get\_cameras 路由發送 GET 請求, 包含城市與區域資訊。  
根據城市或區域資訊查詢照相機數據。

SQL query:

當同時提供縣市及地區時:

當僅提供縣市時:

```
if city and region:
    query = """
    SELECT
        c.Limits,
        c.Direct,
        c.Addr,
        p.DeptNm,
        p.BranchNm
    FROM
        camera c
        JOIN ps p ON c.Addr = p.Addr
    WHERE
        p.CityName = %s AND p.RegionName = %s
    ORDER BY
        c.Addr desc
    """
    cursor.execute(query, (city, region))
    results = cursor.fetchall()
    return jsonify(results)
```

```
else:
    query = """
    SELECT
        c.Limits,
        c.Direct,
        c.Addr,
        p.DeptNm,
        p.BranchNm
    FROM
        camera c
        JOIN ps p ON c.Addr = p.Addr
    WHERE
        p.CityName = %s
    ORDER BY
        c.Addr desc
    """
    cursor.execute(query, (city,))
    results = cursor.fetchall()
    return jsonify(results)
```

設計原因:

根據用戶所提供的地區範圍不同, 進行兩種不同的查詢方式, 可確保不會查詢到錯誤的範圍。

例外處理：

使用以下程式碼來確保用戶至少有選擇縣市

```
if not city:
    return jsonify({"error": "請選擇縣市(區域)"}), 400
```

查詢中若出現錯誤

```
except Exception as e:
    return jsonify({"error": str(e)}), 500
```

**add\_camera:**

用戶透過 /add\_camera 路由提交 JSON 格式請求，包含照相機的地址、方向、速限、城市與區域資訊，然後將數據插入 camera 與 ps 資料表，並記錄在 records 表中

SQL query:

分別以以下指令插入 camera, ps, records 的 table 中

```
insert_query_camera = """
INSERT INTO camera (Addr, Direct, Limits)
VALUES (%s, %s, %s)
"""
```

```
insert_query_ps = """
INSERT INTO ps (CityName, RegionName, Addr, DeptNm, BranchNm)
VALUES (%s, %s, %s, %s, %s)
```

```
log_query = """
INSERT INTO records (Uid, CityName, RegionName, Addr, Direct, Limits)
VALUES (%s, %s, %s, %s, %s, %s)
```

例外處理：

當操作失敗時

```
except Exception as e:
    conn.rollback()
    flash(f"新增失敗: {str(e)}", "danger")
```

**update\_camera:**

用戶透過 /update\_camera 路由提交請求，包含新的速限與地址資訊。  
更新 camera 表中速限數據，並記錄於 records 表

SQL query:

更新速限：

紀錄更新操作：

```
update_query = """
UPDATE camera
SET Limits = %s
WHERE Addr = %s
"""
```

```
log_query = """
INSERT INTO records (Uid, CityName, RegionName, Addr, Direct, Limits)
VALUES (%s, %s, %s, %s, %s, %s)
"""
```

設計原因：

- 1.更新操作針對地址進行條件篩選，確保唯一性
- 2.操作記錄包含新速限與地址，方便後續追蹤

例外處理：

缺少必要欄位時：

```
if not addr or not new_limit:
    flash("請提供地址和新速限", "danger")
    return redirect("/update")
```

更新失敗時：

```
except Exception as e:
    conn.rollback()
    flash(f"更新失敗: {str(e)}", "danger")
    return redirect("/update")
```

**delete\_camera:**

用戶透過 /delete\_camera 路由提交 JSON 格式請求，包含城市、區域、地址及方向資訊，然後從 camera 資料表中刪除指定記錄，並記錄刪除操作到 records 表中。

SQL query:

刪除照相機記錄：

```
delete_query = """
DELETE FROM camera
WHERE Addr = %s AND Direct = %s
"""
```

紀錄刪除操作：

```
log_query = """
INSERT INTO records (Uid, CityName, RegionName, Addr, Direct)
VALUES (%s, %s, %s, %s, %s)
```

設計原因：

利用 Addr(地址)與 Direct(方向)作為條件篩選，確保刪除的精確性。  
若刪除的攝影機與方向不匹配，SQL 不會執行操作。

例外處理：

確保要刪除的紀錄存在：

```
if cursor.rowcount == 0: # 確認是否有刪除成功
    flash("刪除失敗：找不到指定地址", "danger")
```

若刪除出現錯誤：

```
except Exception as e:
    conn.rollback()
    flash(f"刪除失敗: {str(e)}", "danger")
```

**get\_update\_history:**

用戶可以透過 /get\_update\_history，查詢其對照相機數據進行的新增、更新或刪除操作的歷史紀錄，並返回 JSON 格式的數據，包含操作的詳細內容。

SQL query:

```
query = """
SELECT
    uh.CityName,
    uh.RegionName,
    uh.Addr,
    uh.Direct,
    u.username,
    uh.Limits
FROM
    records uh
    JOIN users u ON uh.Uid = u.Uid
WHERE
    uh.Uid = %s
```

設計原因：

利用 JOIN 將 records 表與 users 表聯結，顯示執行操作的用戶名 (username)。

篩選條件使用 `uh.Uid = %s`，確保返回的紀錄僅屬於目前登入用戶。

返回字段包括城市、區域、地址、方向以及速限，提供全面的操作記錄。

例外處理：

查詢過程出錯時：

```
except Exception as e:
    return jsonify({"error": str(e)}), 500
```

## Others

Progress：

下圖為我們project progress大致上的示意圖，比我們所預計的要晚了一點開始，但整體的進度都是有按部就班地完成各個function的功能



Problem encountered：

我們在製作專題的過程中，所遭遇的最大的問題就是，曾有一次我們在測試成功後，還未將測試完成的版本更新到github上面，就開始設計下一個function，這導致我們用來設計的程式碼是舊版的程式碼，由於不只有一個程式碼的更動，所以這導致了我們在測試新的功能時，發生了前一個function的功能出了問題，我們的解決方法是回溯到上一個版本，並重新修改，這導致我們花費了不少的時間，我們也學到了教訓，在每一次測試完成後，都會確認更新到github上面，並且在submit時添加版本備註，如此一來我們之後也便沒有再遇到相同的問題發生。

Links：

[project code link](#)

[presentation video link](#)