計算機演算法期末Project
旅遊規劃王

學號	姓名
106362501	簡浩庭
106362505	何宗龍
106362512	王志嘉

目錄：

專案網站連結 (Deploy to GCP App Engine)

應用程式建立影片 講解連結
網站 Demo 影片 講解連結

一. 動機與目的

1. 行程路線障礙者良藥
   出遊玩時總是會計劃著許多地區的景點，但總是難以規劃出較適合之路徑，造成通勤時間與成本的增加。
2. 為了避免花過多的時間與成本在交通上，程式取得多個景點位置後將規劃出一條通車時間最短之路徑。
3. 規劃好景點後，天氣總是不盡人意
   如:下雨天造成原本計畫泡湯、空氣不佳導致吸入過多髒空氣、熱指數過高容易造成中暑...等，所以需要一個平台整合自訂的景點即時資訊
4. 為了讓使用者能更快更方便的獲得天氣或空氣品質等資訊，減少反覆查詢到各個資訊平台查詢之困擾，且程式會藉由這些資訊判斷是否適合前往。
5. 經由程式規劃與推薦後，就能達到節省交通時間與降低敗興而歸的情況，藉此提升使用者旅遊品質與家庭和諧。

這個服務對於有一堆想去的地方，卻不知如何安排行車流而焦頭爛耳，也為您的家庭和諧提供的優良服務阿!!

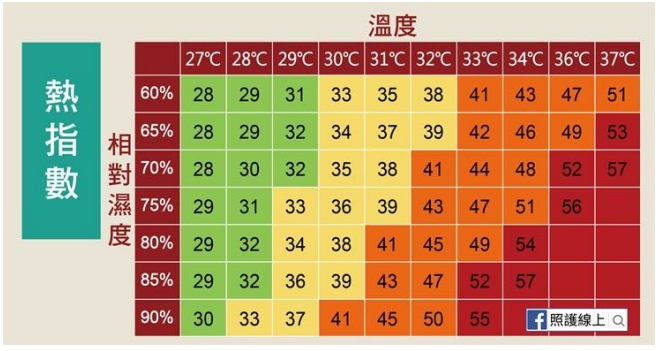
二. 預計完成功能

1. 主要頁面分為兩頁：
   - A. 景點選擇：
     使用者可任選多個景點(10個以內)，並設定出發地，
     此頁面中主要區分為兩個區塊：
     a. 提供全台灣景點列表與基本介紹或相關資訊等等
     b. 顯示已選擇的項目清單
   - B. 行程安排推薦結果：
     a. 顯示預計花費總里程
     b. 安排順序Tabale與項目資訊
     c. Google map 路線標記

2. 程式串接資訊開放平台的景點資料至Mongodb Atlas，並使用mongoengine SDK 製作ODM 模型，取得天氣資料後，抓出雨量、空氣品質指標等資訊，顯示於(B)介面上。
3. 介面上顯示出等級時間內是否有雨量累積(毫米)。
4.判斷空氣品質指標AQI:0~50顯示良好，51~100顯示普通，100以上顯示不健康。

	AQI	0-50	51-100	101-150	151-200	201-300	301-500
AQI指標	對健康的影響	良好	普通	對敏感族群不良	對所有族群不良	非常不良	有害
	代表顏色	綠	黃	橘	紅	紫	橘紅
	人體健康影響	空氣品質為良好，污染程度低或無污染	空氣品質普通；但對非常少數之極敏感族群產生輕微影響	空氣污染物可能會對敏感族群的健康造成影響，但是對一般大眾的影響不明顯	對所有人的健康開始產生影響，對於敏感族群可能產生較嚴重的健康影響	健康警報：所有人都可能產生較嚴重的健康影響	健康威脅達到緊急，所有人都可能受到影響

5.熱指數：藉由溫度與濕度，判斷環境舒適程度，下圖為橘色與紅色區塊熱指數偏高，易中暑，為危險等級區塊。



5.除了顯示天氣狀況外，程式會推薦使用者一條最短路徑(環)，讓使用者節省交通上的時間與成本。

三. 使用語言、平台、開發環境

- 前端應用組件與模板框架
  - HTML
  - CSS
  - JavaScript
  - jQuery
  - Datatable
  - Bootstrap
  - Jinja2
- 後端處理之程式語言
  - Python
- 後端框架與模板引擎
  - Flask
  - Jinja2
- 資料庫與其ODM框架
  - Mongodb Atlas
  - Mongoengine(連結)
- 主要使用模組與SDK
  - google-maps-services-python(連結)
  - Google Maps Directions API(連結)
  - Google Maps Distance Matrix API(連結)
  - Google Maps JavaScript API(連結)
  - DataTables Jquery(連結)
- 部屬平台
  - Google Cloud Platform App Engine
- 開發環境
  - Windows WSL2 Ubuntu 18.04
  - Docker Engine 19.03.12
  - ~~Kubernetes 1.18.3(還沒做大，再拔掉)~~
- API來源:
  - 民生公務物聯網 API
  - 民生公務物聯網-中央氣象局-雨量站 API
  - 民生公務物聯網-環保署-國家空品測站監測資料 API
  - 景點 - 觀光資訊資料庫 API

四. 使用資料結構與演算法分析

(1)使用資料結構:

Hash Table: Python 內建型態 Dictionary 他能幫我們以鍵對值來儲存我們的資料，以及使用nested的方式儲存稍微複雜的資料。

```python
@classmethod
def get(cls, args:dict):
    args = defaultdict(lambda: None, args)
    only_fields = ['Id','Name', 'Toldescribe', 'Add', 'Tel', 'Location']
    raw_query = {
        'Name': args['Name'],
        'Id': args['Id'],
        'Toldescribe__icontains': args['Keyword'],
        'Ticketinfo__icontains': args['Ticketinfo'],
        'Travellinginfo__icontains': args['Travellinginfo'],
        'Add__icontains': args['Add'],
        'Add__not': re.compile('.*(金門縣|澎湖|綠島|小琉球|馬祖|蘭嶼|連江).*'),
        'Region__not': re.compile('.*(金門縣|連江縣|澎湖縣).*'),
        'Location__near': list(map(float, args['Location'].split(','))) if isinstance(args['Location'], str) else None,
        'Location__max_distance': float(args['Distance']) if isinstance(args['Distance'], (str,float)) and isinstance(args['Location'], str)
    }
    query = dict(filter(lambda item: item[1] is not None or False, raw_query.items()))
    if(cls.objects.count() == 0):
        cls.insert_all()
    q_set_json = cls.objects(**query).only(*only_fields).exclude('id').to_json()
    return json.loads(q_set_json)
```

如框選所示，使用字典將 query 參數包起來，以從 mongodb 取得資料

Queue: 我們有 Python 中使用 threading 模組，以使用多個子執行緒去訪問來源資料(API)並存儲到 Mongodb 做紀錄與 thread-safe 的存入 Queue 中(因在做多個子執行緒時無法直接 return 資料，且稍後前端會需要這些值的資訊)，這樣才能照每個 job 完成的順序並將其取出，最後再回送至前端。

```python
def job(args: dict, q: Queue):
    info_dict = {}
    info_dict['ScenicSpotInfo'] = (ScenicSpotInfo.get(args))
    for sta in stations:
        ScenicSpotInfo_Loc = info_dict['ScenicSpotInfo'][0]['Location']
        loc_array = CILocation.get({'Location': ScenicSpotInfo_Loc, 'Station': sta, 'Distance': 0.1})[0]['location']['coordinates']
        sta_info = Datastream.get_station_info(
            sta, ','.join(map(str, loc_array)))
        info_dict[sta] = sta_info
    q.put(info_dict)

for raw_arg in raw_args[Inside]:
    t = threading.Thread(target=job, args=(
        {Inside: raw_arg}, q))      You, an hour ago • fix template
    t.start()
    threads.append(t)

for thread in threads:
    thread.join()

for _ in raw_args[Inside]:
    result_list.append(q.get())
```

使用 threading 搭配 queue 做存儲與取一次值。

程式碼來源:
[Python dictionary implementation]
[cpython-dictobject.c]
[python-多線程]

(2)使用演算法:

比較項目	Brute Force	backtracking	2-opt
優點	一定可以找到最佳解	優化版的暴力法，透過遞迴尋找邊界以壓縮暴力法枚舉的所有可能。	效率高，速度快
缺點	需用較多硬體資源，要探討的情況有多種	不好找到能修改 backtracking 的切入點，該方法效率還是比較低。	只有局部最佳化 是一種隨機性算法，結果可能會不同。

程式碼來源:
https://www.geeksforgeeks.org/traveling-salesman-problem-tsp-implementation/
2-opt求解TSP（旅行商）问题的python实现
Traveling Salesman Problem (TSP) Implementation
https://www2.seas.gwu.edu/~simhaweb/champalg/tsp/tsp.html

在前端的部分，在第二個顯示Result頁面的時候，因為後端傳回的資料並不是根據我傳入的順序做回傳，所以前端必須根據回傳的項目給定index，再經由Datatable排序此欄位。

```
"columns": [
    {
        data: null,
        render: function (data, type, row, meta) {
            // 因為後端回傳的順序並不是按照傳入的順序，所以需要給index，讓Datatable根據欄位value排序
            Id = data['ScenicSpotInfo'][0]['Id'];
            order = mapping_paths.indexOf(Id);
            return ++order;
        }
    },
```

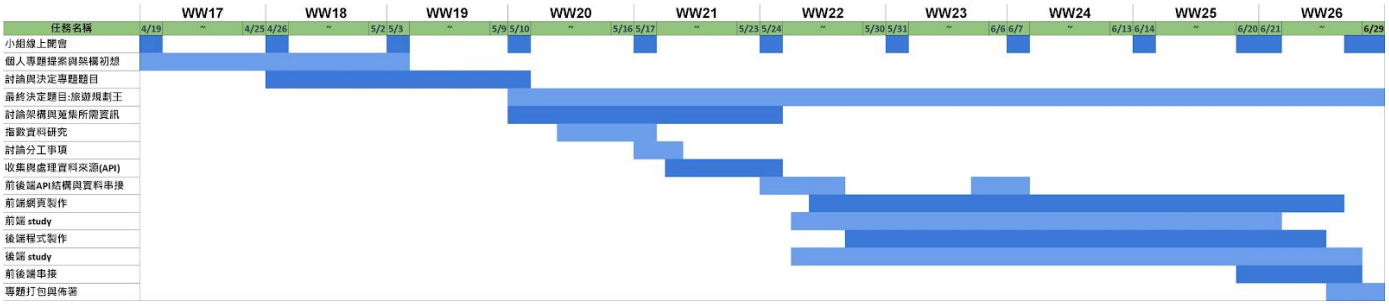
這邊直接利用Javascript Array內建的indexOf函式取得排好序Array的index，設給列的欄位值，針對 String.prototype.indexOf() 做源碼的探討發現
他有定義了

1. LinearSearch
2. BoyerMooreSearch
3. BoyerMooreHorspoolSearch
4. SingleCharSearch
5. InitialSearch

內建會根據傳入的初始data做搜尋演算法的選擇，我們也能理解這種概念的作法，因為再老師的課程中，經由引導性和思考讓我們理解到「沒有最好的演算法，只有最適合的演算法」。

五. 甘特圖 研發時程以及分工情況

甘特圖 Google sheet Link(Google sheet 有紀錄一些備註)



分工情況:

姓名	分工事項
簡浩庭	➢ 前端 (Jinja2、HMTL、CSS、JavaScript、jQuery與內建演算方法應用) ➢ Google Map API (Distance Matrix、Directions、Maps JavaScript) ➢ Web API串接 ➢ 前後端資料串接結構定義 ➢ 演算法與資料結構分析討論 ➢ 基本架構優化與討論
何宗龍	➢ 專案管理(會議主持與紀錄、掌控進度) ➢ 報告整合(甘特圖繪製) ➢ 提供專案想法 ➢ 演算法與資料結構分析討論
王志嘉	➢ 後端 (python Flask Restful Api、 Jinja2 後端服務串接和 Mongodb 等與演算法整合) ➢ 依 OGC 定義的 SensorThings API (民生公務物聯網之來源感測器資料) 設計與拆分出 Mongodb 的 ODM 模型及其索引優化和最後的 pipeline 以倒出想要的資料格式等 ➢ Google Map API (Distance Matrix、Directions) ➢ Web API串接 ➢ 演算法與資料結構分析討論 ➢ Deploy to GCP (App Engine)， 可不下線更新 ➢ 將應用程式容器化與 Infra 架構性能優化

六. 結語 (心得與感想)

(1)簡浩庭

在這次課程的期末專案中，原本就想多使用一些以前沒有用過的服務，例如GCP上的佈署、API的串接與套用或用Docker包裝起來等，所以我就在與夥伴開會時提出想套用看看過去沒使用過的東西，順邊學習一些雲端技術與容器化的概念，但許多服務對於免費仔都有許多限制，所以我們只能在有限的環境下測試本次的專案，而老師在最後一堂課也有剛好提到相同概念的東西，所以也算是有剛好呼應的老師的課程內容，能藉由本次的專案學到新東西。

因為之前有用Laravel做過全端的開發經驗，而夥伴沒有類似的經驗，所以我就負責接下了前端的工作，一開始先弄一個能夠做基本測試的環境能讓其他人能簡單的測試，自己再專心的研究前端的模板套用，過程中跟夥伴討論資料流如何串接、前端預期功能、頁面排版設計等等事項，也因為需要用到MongoDB，所以也刷了許多MongoDB university 的教學，發現自由度比過去的RDBMS高上許多，也學習使用好用的工具Postman，在跟隊友討論完基本的資料定義與來源後，我們討論可能會使用到的演算法，那根據上課的內容第一個想到的就是Dijkstra Algorithm，但是發現我們每個點都可以相通，只不過時間及距離成本不同而已，所以也Google了許多的資料有查到 Traveling Salesman Problem(TSP)的相關資訊，但這會計算環形的shortest path跟我一開始的設計理念稍有出入，但在跟團隊討論後還是決定先暫時使用這種演算法。

我們在討論中確定各自需要用到的工具、語言、SDK或框架等等，而前端除了基本的HTML、CSS、JS，也花時間研究如何套用Bootstrap framwork快速完成前端樣板的頁面，利用 Datatable 的基本樣式與功能列出我們的資訊，使用Ajax做資料的取得，也藉由這次的機會使用很就以前就想試用看看的Google Maps API，後端的夥伴也很給力，把flask的服務快速的建立起來以及部署，讓我能夠做Ajax資料串接的測試，否則只能用Postman塞給前端假資料。

自從學期前半段得知我們可能需要在期末時作出一個架構或一個服務時其實我還蠻期待的，因為我還蠻享受在這過程中慢慢看到專案的進展與最後的成果，就會有滿滿的成就感，而最後在網站展示的結果，讓還沒接觸過類似雲服務的我蠻滿意的，但因為時程的安排，以及需要上班+上課，能用來做專案的時間實在太零碎分散，導致進度容易slide，這次也花了太多時間在處理JS的event，以及思考路線演算法的部分，導致有一些想做的功能還沒完成(但不影響整個架構)，而且在這開發的過程中體悟到在團隊開發中要完成一個簡單的網站需要許多的預先設計定義與資料串接結構細節，否則後期會浪費許多工程在解決一開始沒討論好的事情，在這次專案中嘗試了許多以前想用的東西，真的自學到了不少東西，這真的讓這學期充實很多。

(2)何宗龍

透過實際專題設計，促使我們更深入的思考老師上課教過的知識，一開始我們花了很多時間思考題目，以及題目中能使用的資料結構與演算法，重一開始的「台灣股票市場分析」，我蒐集了如何爬取取得台灣證交所的程式，但實作時發現要將這些數據轉換成股票的指標，實作上較困難，故考慮另一個專題「社群平台回應整合服務」，於是我們又實作了一段時間，開始研究如何將各種平台(如:FB、YT、Reddit、Twitter)結合於一個平台，方便使用者回覆，但實作一段時間後發現FB Developer權限不易取得，於是又再度取消此方案，接著討論專題「Ubike調度建議」，希望能取得Ubike站點單車個數後，給予一條路徑，建議每站該放下幾台，或是載幾台上車，但難以判斷該行經那些站，且難以判斷每站該載上幾台或放下幾台單車，種種問題讓我們又放棄了此方案。此時我和組員已經有些崩潰，因為時間約只剩一個月，已經很緊張，擔心再度失敗後就繳不出專題，接著我再討論「旅遊規劃王」，因上一個專案已經有研究過google API了，故轉到此專案上能更快上手，我們討論了整體架構，我幫忙組員分析了這個專案中會用到哪些資料結構與演算法後，發覺這個專題可行度較大，於是我開始將組員分工誰實作前端、誰作後端程式，而我負責開始撰寫報告與幫忙蒐集程式中可能用的到的程式給組員參考，隨著時間一天一天過，我們也越來越緊張，擔心到了期末這個專題又再失敗一次，所以我每天都緊盯並督促著組員的進度，並要求他們將自己遇到的問題記錄下來，開會時再一起討論。花了許多天，常常我們忙到三更半夜討論將前端與後端作結合，最後還是有將專題做出來，也讓我鬆了一大口氣。

這次專題中讓我深深了解到了團隊合作的重要性，一個好的專案在開發中會遇到許多的問題，每個人擅長的部份不盡相同，每個人會思考到的問題也不同，雖然前面失敗了許多次，但組員們還是每週不離不棄，勇敢面對這個作業，讓身為此專案經理的我看了很感動。最後，我非常感謝我們組員合作的這幾個月都非常配合我要求的進度，且不惜犧牲假日與睡眠一起完成此專案。

(3)王志嘉

老實說，這次實際專案開發所能用的時間並不算很充裕。平常公司下班到家後都晚上九點十點了，週六更是從早上上課到晚上十點。平日晚上也有選修數堂其它課程，導致能利用的時間不多。那利用週日及半夜的時間並依照團隊的各式需求開發到這樣的結果也算很勉強了，從零開始看 OCG API 與 MongoDB 與其 ODM 模型設計，這裡也花了些時間從 SQL 轉到 No-SQL 的層面去看他是如何取資料以及其關聯的效果等等。

MongoDB 與一般的 RDBMS 有非常大的差別是他不用外鍵與Join，MongoDB 是採用 Reference 與 Embedding 來處理各個 Document 的東西，這導致我們不能採用 ORM 模型得改使用 ODM 模型並需要把 Embedded Document 給拆開來做，且 Query 的方法也會有所不同，倒出資料時也得使用 Aggregate Pipeline 將我們想要的資料格式給一層一層的倒出來。另外，原本後端預期是使用 Flask 框架單純傳送 Restful-API 與向來源端發送 Restful-API 取資料並存入 MongoDB 然後回傳資訊還有使用者認證等等，但由於團隊沒有使用前端框架經驗，所以說我這邊只好把 views 的部份給簡單的做起來，之後我會再看看前端框架並搞好他。

老師有在最後一節課說到雲端與大數據等等，其中虛擬化及平行化是趨勢，所以說我們也盡量照這個方向去建立應用程式與部署至Google Cloud Platform。那接續上一段的問題，以我目前所看到的，若前後端不分好的話，在往後平行化擴展容器時得連後端的部分組件一起擴展，這對 Cluster 上的 Node (虛擬或實體主機) 是一個負擔，所以能分最好(不花錢，但花時間)。關於這個應用的架構上因考量到成本問題，所以說我把原本要使用的 Memory Cache 與 kubernetes 給拔了，加上這些東西會讓原本的預算變 3 倍以上且我們的網站沒有流量以及"沒有更求"好的體驗(例如loading第一頁資料的時間從五秒便瞬間等)。另外我也有將 MongoDB 給拆分出來，不得不說 MongoDB atlas 還蠻佛心的，它提供了免費的 Database as a Service 即用服務給我們使用，我們不用出錢不用管理基礎設施即可使用他們由 Google Cloud Platform 的台灣資料中心下的 Cluster 分配的 MongoDB。我們這邊只須要管好權限與資料庫的使用容量使否有到上限可，基礎維運的東西我們都不用管(令人頭痛的資料碎片問題呀等等)，只要知道怎麼用 MongoDB 就好了，真的非常省時與省力。

那回到演算法的部分，最初是想採用 Dijkstra Algorithm，但後來發現我們的 case 是每個景點的位置都有通，所以並不適用 Dijkstra Algorithm 這種在有限制路徑條件下尋找路徑，最後我們改用了 Traveling Salesman Problem 來實現這個專案。往 Traveling Salesman Problem 的方向後還發現了 2opt 與 backtracking 等方法來實作，但最後我們選了 Brute Force，因為他能很簡單的紀錄最佳路線且目前我們有限制使用者只能選 10 個地點，所以不會花上太多時間即可計算完成。

原本我是想利用民生公務物聯網的資料來製作智慧推薦(依照即時的天氣與空氣狀況核災害等條件)與利用推薦下的結果做最佳路徑安排或是路徑安排調整等，但礙於時間不夠，所以團隊先暫時做出單純的最短路徑與氣象資訊參考等等。

透過這次專案真的學習了很多，無論是學習與指導，我都能以目前的 SRE 與雲端架構師的實習工作中拉出一些東西來使用，像是提供利用 Postman 當 Mock Server 的解決辦法以及 Infra 的改善和應用程式效能如何優化等等。關於前端的部分也從團隊理學到了很多，像是 Jquery 有 datatable 這種玩意能為我省下很多導入資料的步驟及方法還有Bootstrap樣式等等。就決論來說，這個真的是一份非常有價值與非常充實的專案~