目錄

---

1. 簡介 **3**

2. 前端 **3**

2.1 App設計 3

2.1.1 目的 3

2.1.2 目標客群 3

2.1.3 使用者體驗 3

2.2 技術 4

2.2.1 螢幕畫面 4

2.2.2 外部服務 9

2.2.3 畫面, 畫面控制器, 其他類別 9

2.2.4 Data models 17

3. 後端及演算法 **18**

3.1 演算法概念 18

3.2 演算法模塊 19

3.2.1 R2R 模塊 19

3.2.2 U2R 模塊 19

3.2.3 U2U 模塊 19

3.2.4 環境變數模塊 19

3.3 推薦系統流程圖 20

3.4 後端 API 節點 21

3.4.1 User_choose 21

3.4.2 Collections 22

3.4.3 Signup 23

4. 資料庫 **24**

4.1. 表格設計 24

4.2. 問卷調查 24

4.3. 餐廳資料蒐集 24

4.4. 伺服器管理 24

4.5. 資料擷取 24

5. 實驗結果 **26**
5.1 資料集與定義 26
5.2 統計結果 26

6. 未來發展 **27**
6.1 前端未來發展 27
6.1.1 標籤系統實作 27
6.1.2 評分系統實作 27
6.1.3 Google地圖串接 27
6.1.4 產品導向方面 27
6.2 演算法未來發展 28
6.2.1 餐廳評價 28
6.2.2 餐廳標籤系統 28
6.3 資料庫未來發展 28

7. 附錄 **28**
7.1 工作分配表 28
7.2 後端 API 筆記 29
7.3 結果統計程式碼 33

1. 簡介

身為一個生活在地球的人類，食物是生存必需品，一日三餐是從古自今根深蒂固的觀念，同時也是一件擾人的事情。「今天吃什麼？」、「 待會下課吃什麼？」、「 晚餐吃什麼？」、「 你想吃什麼？」，這些問句充斥在我們一天的生活當中，因此我們為了解決每天不知道吃什麼的問題，決定製作一款「能夠告訴你，你想吃什麼」的APP。

秉持著上述的精神，這款APP我們命名為「食物占卜師（Food diviner）」，我們期望在持續使用的情況下，最終使用者一打開APP出現的第一間餐廳即是使用者最想吃的那一間，就像一次準確無比的占卜。實作上我們將演算法分為四大區塊，透過大量的相似度計算及使用紀錄，調整所有餐廳對某個使用者的分數，並將分數最高的前十名作為推薦名單發送至APP上。

以下將分「前端」、「後端及演算法」、「資料庫」三部分介紹。

2. 前端

2.1 App設計

2.1.1 目的

Food Diviner是一個可以解決每天都會遇到的問題(尤其是針對台灣的大學生)，那個問題就是”我們午餐/晚餐要吃什麼？”。我們希望提供最準確的推薦及預測給使用者，透過精心設計的機制觀察使用者使用App的行為！當使用者越常使用Food Diviner，我們的預測將會越精確。

2.1.2 目標客群

目前，我們的目標客群是時常在公館附近用餐的學生，因為後端餐廳資料現階段只提供在公館商圈的餐廳資料。

2.1.3 使用者體驗

1. 使用Email或是Facebook註冊及登入
2. 有一間餐廳會顯示在主畫面中
3. 向左/右/上滑來決定你喜不喜歡這家餐廳，甚至是現在就要去吃！
4. 幫你自動收集曾經喜愛過的餐廳
5. 提供進階搜尋，讓我們更精準推薦餐廳給您

2.2 技術

2.2.1 螢幕畫面

- 登入/註冊頁面

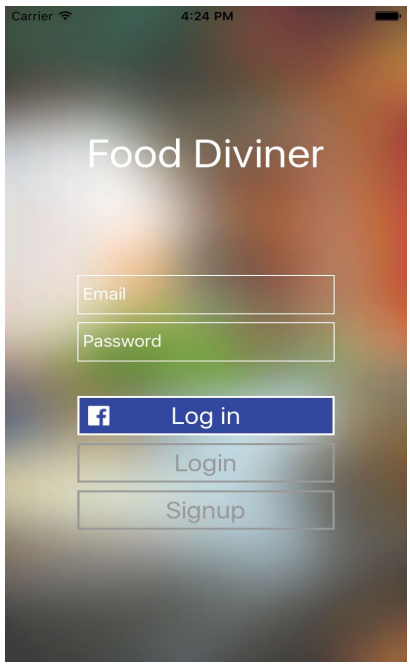

圖 2-1 登入/註冊頁面螢幕截圖

- 主頁面

圖 2-2 主頁面螢幕截圖

- 詳細餐廳資訊頁面(從主畫面進入)

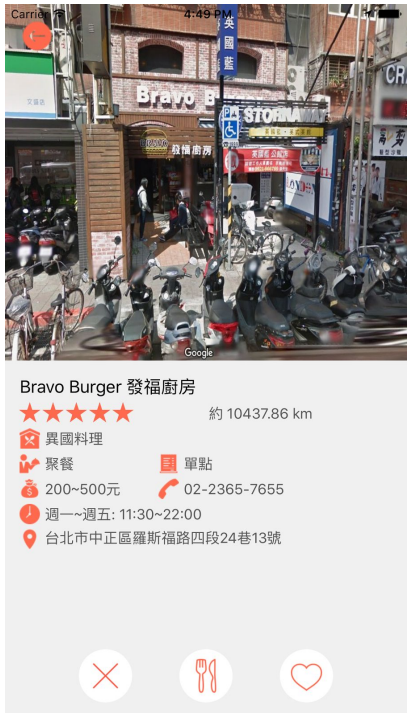

圖 2-3-1 詳細餐廳資訊頁面螢幕截圖

- 詳細餐廳資訊頁面(從曾喜愛餐廳頁面進入)

圖 2-3-2 詳細餐廳資訊頁面螢幕截圖

- 詳細餐廳資訊頁面(從曾去過餐廳頁面進入)

圖 2-3-3 詳細餐廳資訊頁面螢幕截圖

- 進階搜尋頁面

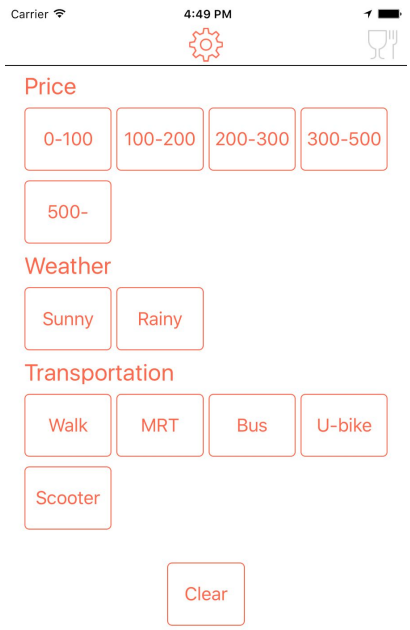

圖 2-4 進階搜尋螢幕截圖

- 曾喜愛餐廳頁面

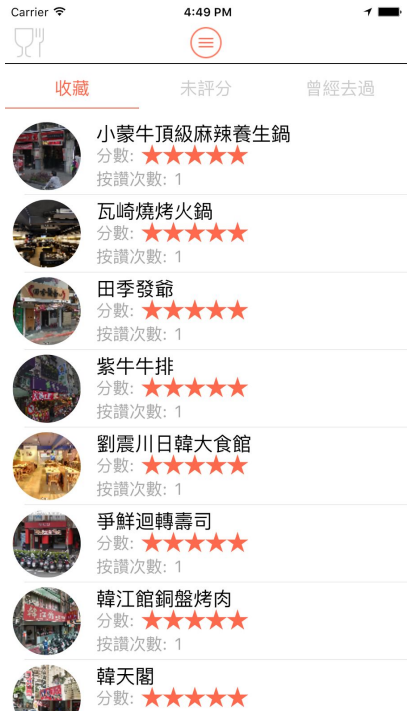

圖 2-5 曾喜愛餐廳頁面螢幕截圖

- 未評分餐廳頁面

圖 2-6 未評分餐廳頁面螢幕截圖

- 曾去過餐廳頁面

圖 2-7 曾去過餐廳頁面螢幕截圖

- 評分頁面

圖 2-7 評分頁面螢幕截圖

2.2.2 外部服務

- Pods
    - TETinderPageView
    - AFNetworking
    - SwiftyJSON
    - HCSStarRatingView
    - NVActivityIndicatorView
    - RealmSwift
    - DeviceKit
    - ZLSwipeableViewSwift
    - Firebase/Core
    - Firebase/Database
    - Firebase/Auth

2.2.3 畫面, 畫面控制器, 其他類別

- 畫面
    - RestaurantView
    - RateTableViewCell
    - CollectionTableViewCell
    - BeenTableViewCell
    - CustomizedButton
    - MainButton
    - TextField
    - StickerImageView
    - MultilineLabel

- 畫面控制器
    - 畫面控制器階層圖

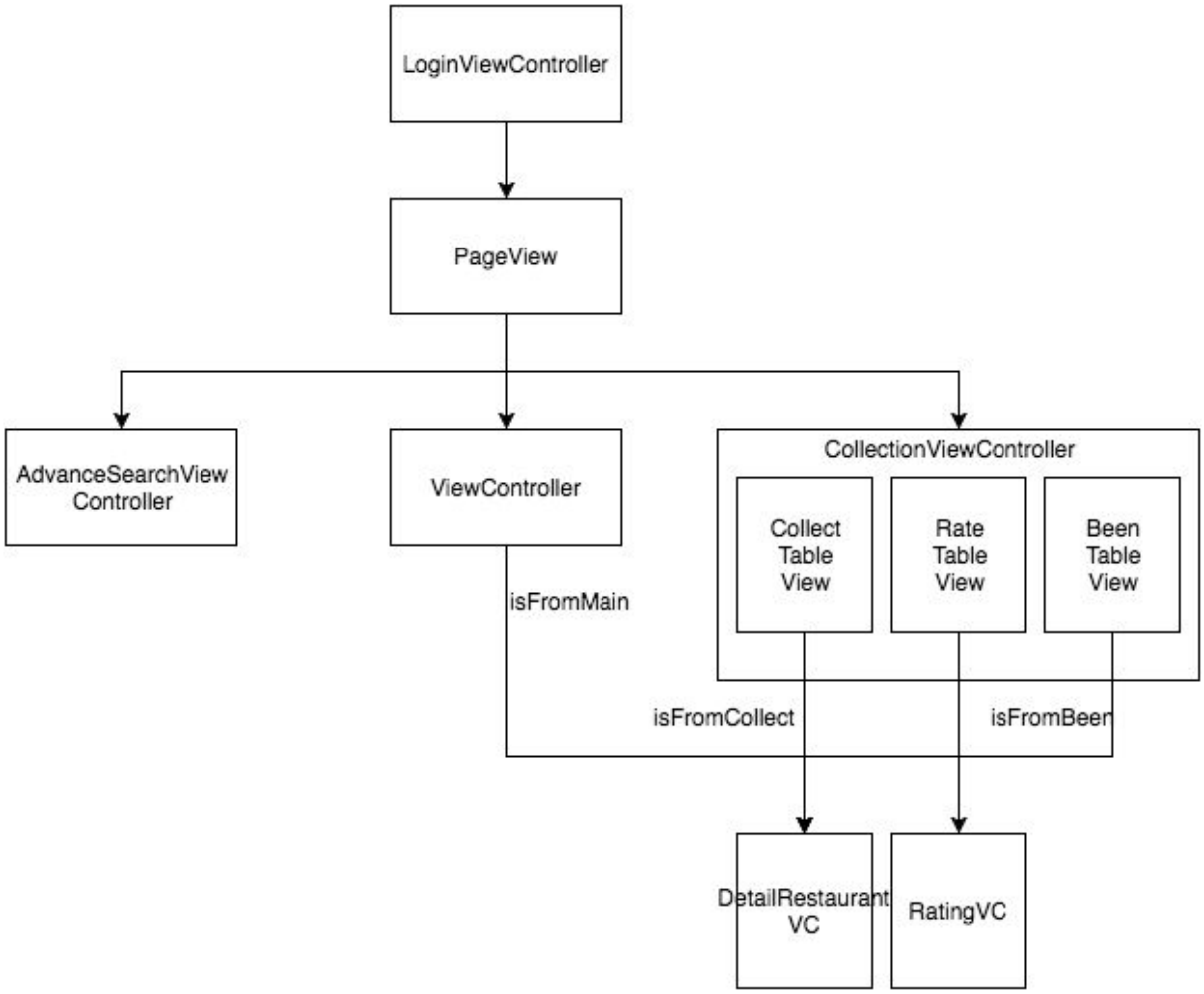

圖 2-8 畫面控制器階層圖

- LoginViewController : 整合Email及臉書登入，驗證系統是由Firebase進行實作。

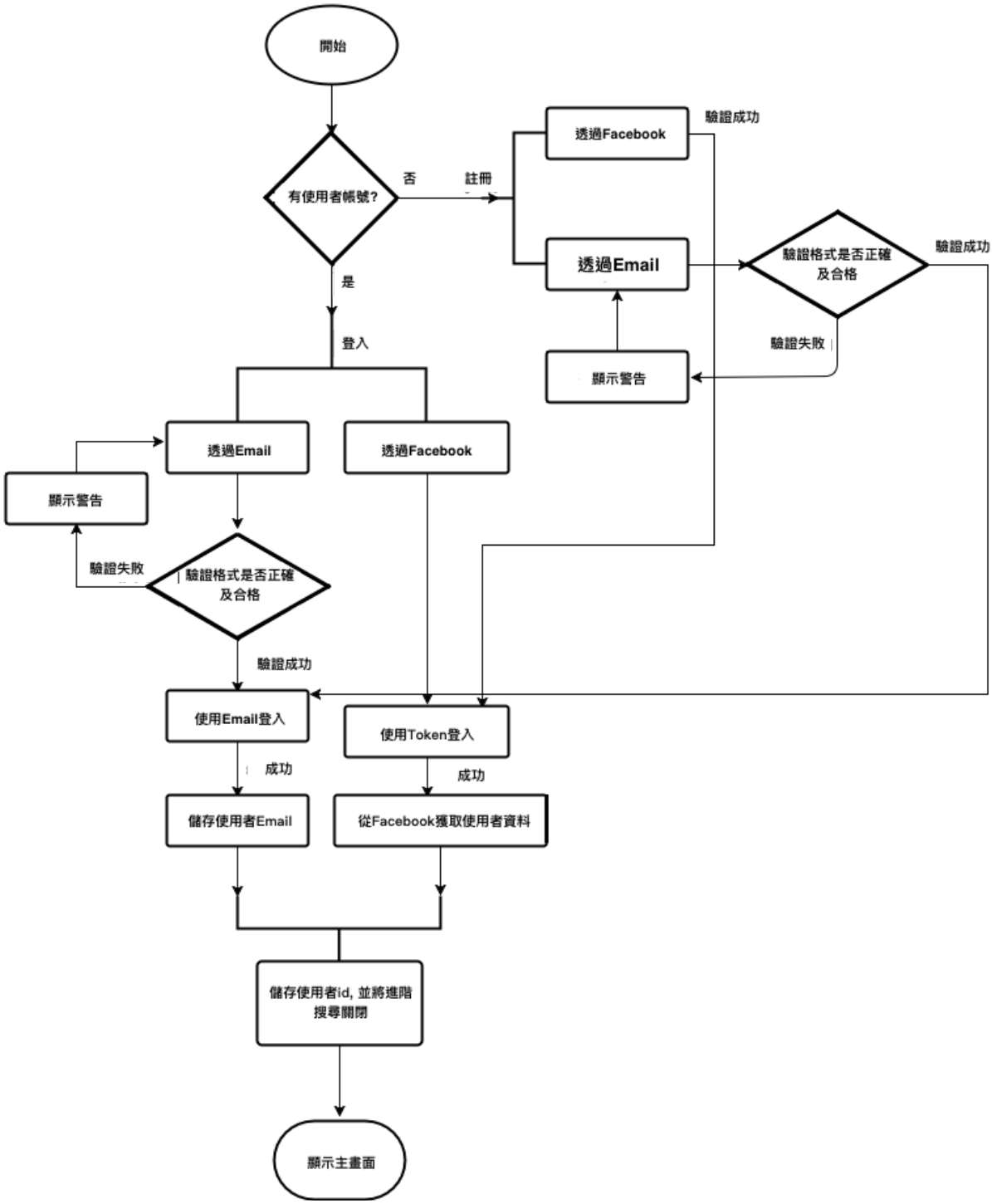

圖 2-9 LoginViewController流程圖

- ViewController : Food Diviner使用最主要的畫面控制器。餐廳的卡片會在這個控制器裡被推薦，然後使用者可以在這個頁面滑左/右/上，或由螢幕下方三個按鈕來對選擇餐廳。如果使用者從未登入，前五張永久存在本地端的餐廳卡片會用來幫助我們初始化使用者的模塊，也就是說，前五張餐廳卡片滑完之後，才算完整地完成登入流程。

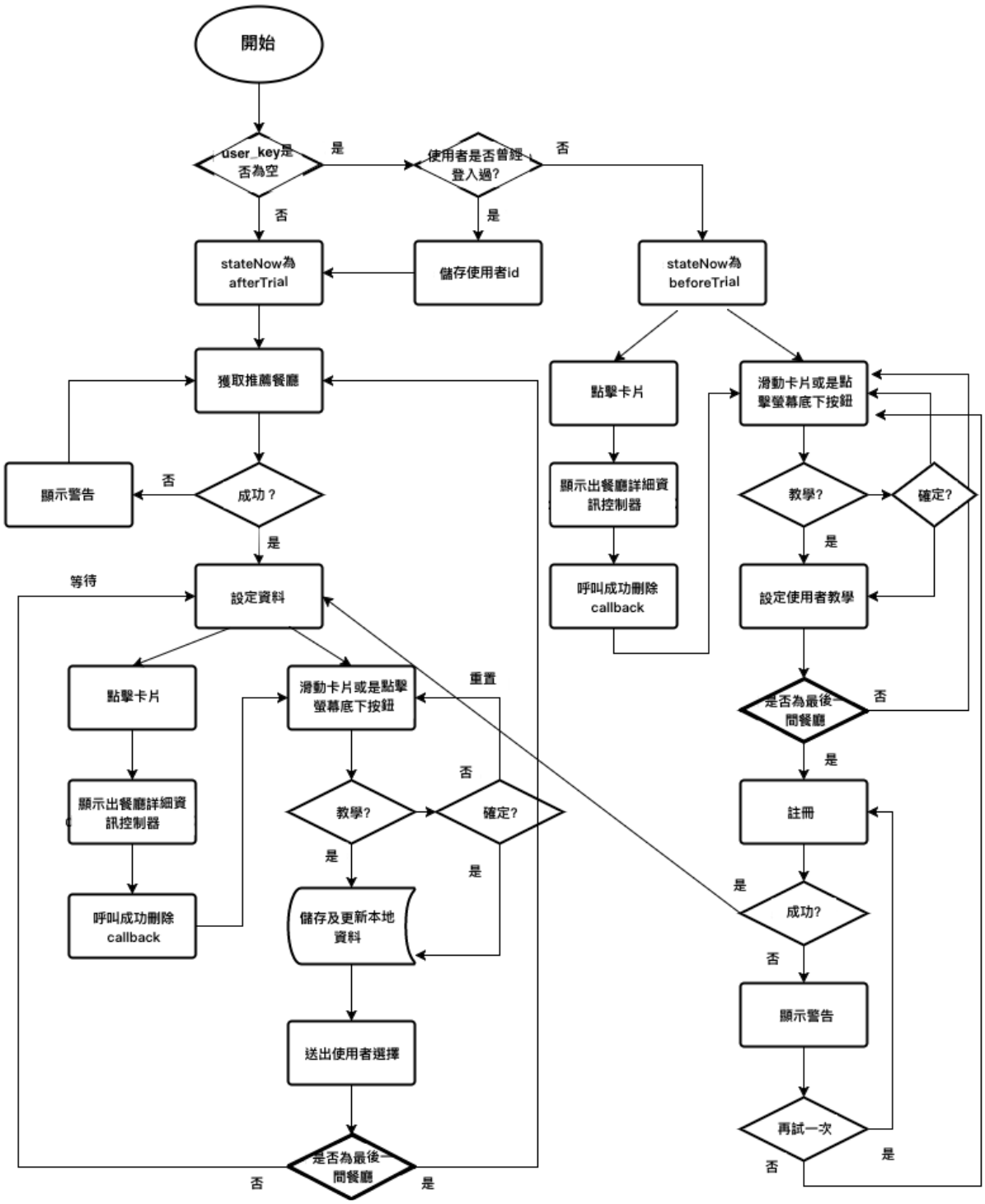

圖 2-10 ViewController流程圖

- DetailRestaurantViewController : 一旦使用者點擊餐廳卡片，或是點擊收藏頁面的餐廳，本畫面控制器就會顯示。我們提供更詳細的資料，例如營業時間或地址和價錢等等。必須一提的是，根據使用者於不同頁面進來本畫面控制器，將會有三種不一樣的按鈕組合於螢幕下方顯示。

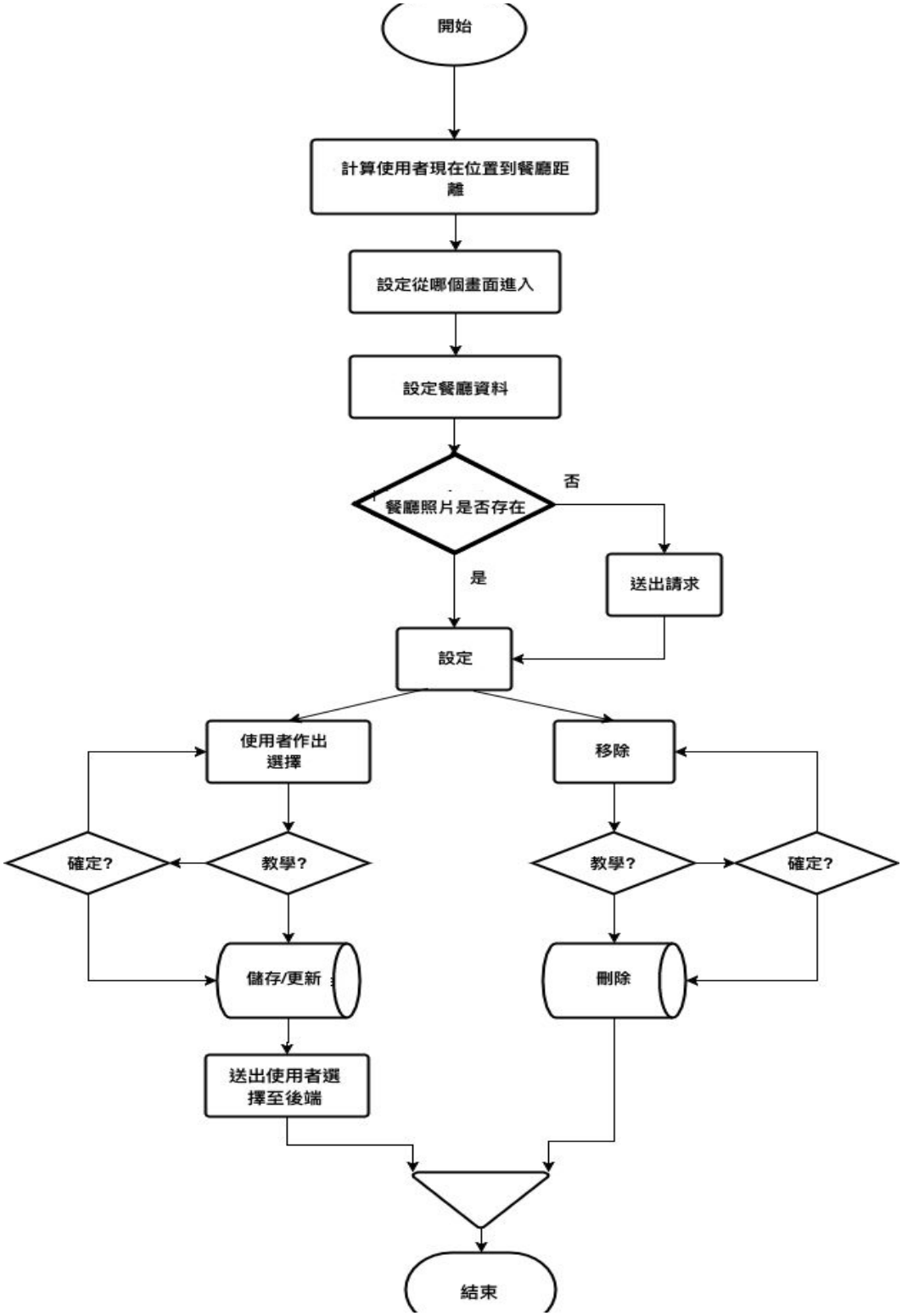

圖 2-11 DetailRestaurantViewController流程圖

- AdvanceSearchViewController : 使用者可以在這裡設置一些篩選條件來幫助他們獲得更好的使用者經驗(因為我們的預測將會變得更準確)！一旦天氣以及交通被選取，你只能透過按螢幕下的清除按鈕才可以清除。進階搜尋的篩選條件將會被儲存在NSUserdefaults()裡面，當使用者離開這個畫面控制器時。

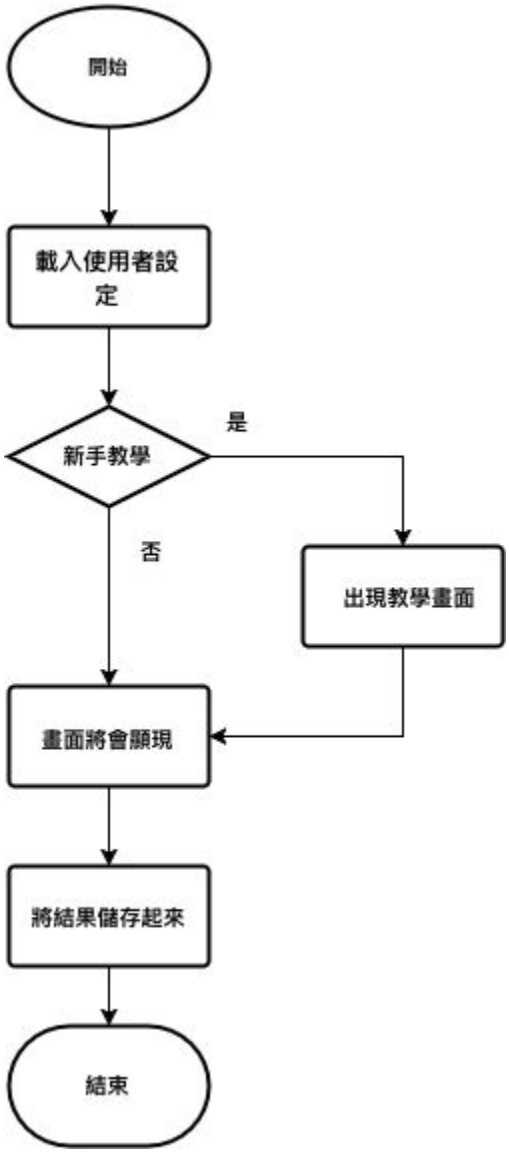

圖 2-12 AdvanceSearchViewController流程圖

。CollectionViewController : 在本畫面控制器裡同時存在了三個 TableView ， 分別是CollectTableView, JudgeTableView以及 BeenTableView。CollecTableView的資料來源是使用者曾經喜愛過的餐廳。JudgeTableView的資料來源是使用者曾經去過但未評分的餐廳。BeenTableView的資料來源則是使用者去過且完成評分的餐廳。

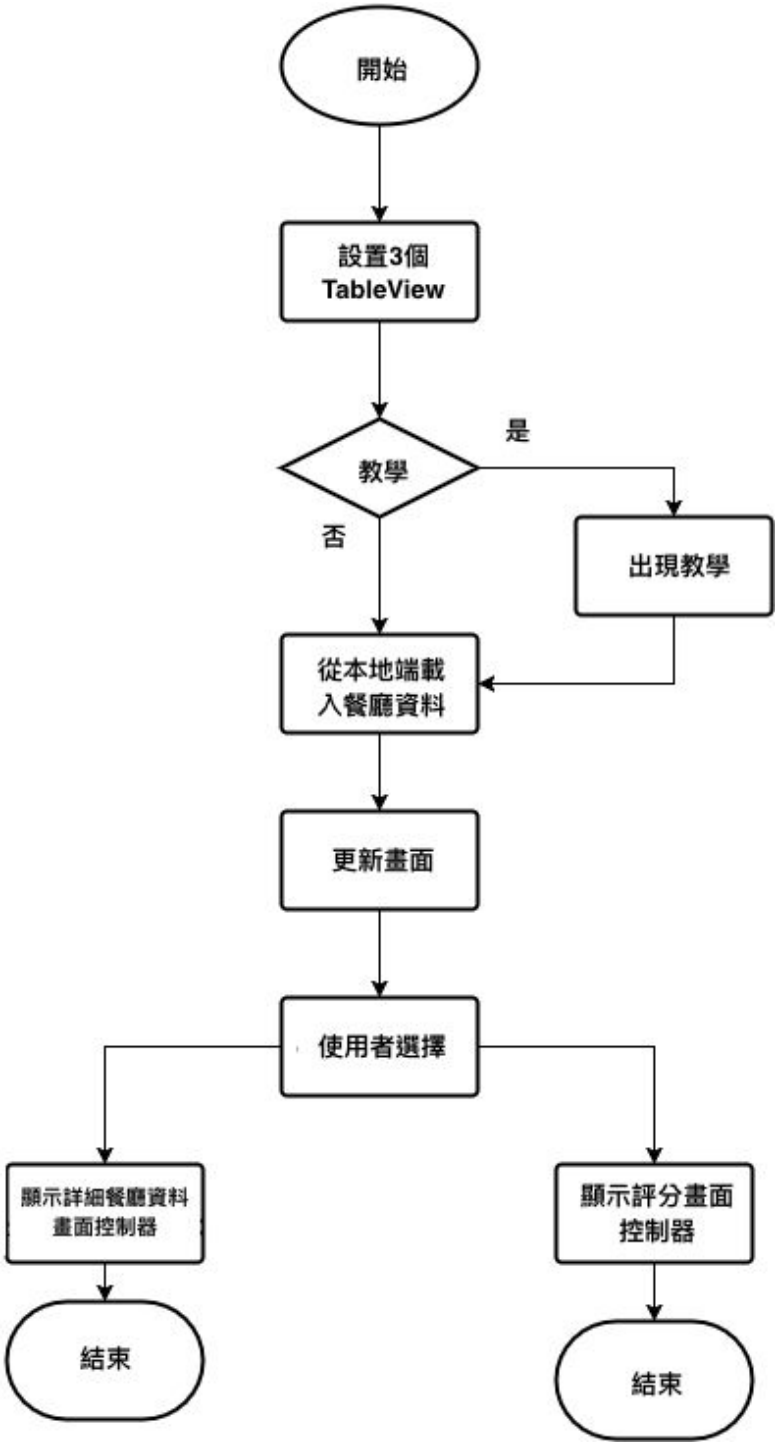

圖 2-13 CollectionViewController流程圖

- RatingViewController : 當使用者點擊尚未評分餐廳時，本畫面控制器即會顯示並提供使用者評分相關機制。

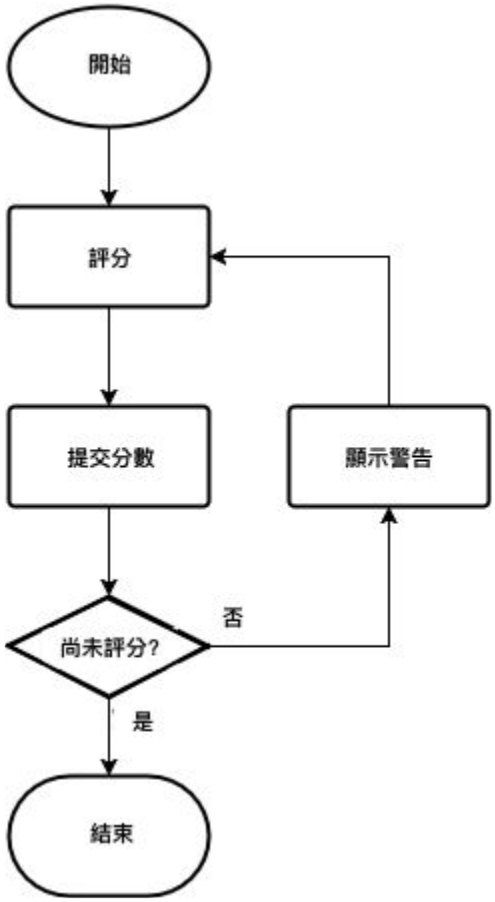

圖 2-7 RatingViewController流程圖

- Other Classes

  - APIManager : 使用Alamofire 及 SwifyJSON外部套件。Food Diviner所有的HTTP請求都是被APIManager所集中實現，除了與FIRAuth相關的請求。提供一組Delegates用來傳送HTTP請求後的結果到畫面控制器，使畫面控制器能根據請求回覆結果，執行相對應動作，例如更新畫面或者是重新傳送該次請求。
  - TrialHelper : 檢查所有Food Diviner團隊認為應該要有教學的動作是否為第一次被使用者觸發，若為第一次觸發，即會跳出針對該動作相關說明及教學。
  - RealmHelper : 使用RealmSwift外部套件，儲存及更新所有在本地端資料。餐廳的照片不會被儲存在本地端，本地端只儲存照片編號，使用照片編號去後端要求相關影像資料。
  - DeviceHelper : 檢查螢幕相關的資訊，回傳該螢幕大小是屬於哪個螢幕族群。

2.2.4 Data models

- Restaurant
    - var address: String!
    - var cuisine: String!
    - var name: String!
    - var order: String!
    - var phone: String?
    - var price: String!
    - var scenario: String!
    - var time: String?
    - var restaurant_id: NSNumber!
    - var tags: String!
    - var photo: NSData?
    - var image_id: String?
    - var avgRating: NSNumber!
    - var userRating: NSNumber!
    - var lastBeenDate: NSDate!
    - var collectTime: NSNumber!
    - var beenTime: NSNumber!
    - var status: NSNumber!

3. 後端及演算法

演算法實作部分，分為四大區塊，分別為「使用者紀錄（R2R）」、「與餐廳比較（U2R）」、「與其他使用者比較（U2U）」、「環境變數（context）」，各大區塊分別計算餐廳分數，並根據不同的百分比，最終加總後排序出前十名的餐廳，傳送餐廳資訊至前端APP。詳見圖3.1 演算法概念圖。

3.1 演算法概念

R2R 模塊根據以往接受或拒絕的餐廳特性，增加或是減少此使用者對各餐廳之喜好程度。

U2U 模塊計算任意兩使用者喜好參數的相似度，選擇與自己最相似的N個使用者，根據相似使用者的喜好紀錄做為計算各餐廳分數之依據。

U2R 模塊利用使用者喜好參數與餐廳特性計算相似度，並依照權重比計算各餐分數。

Context 模塊目前僅作紀錄使用。

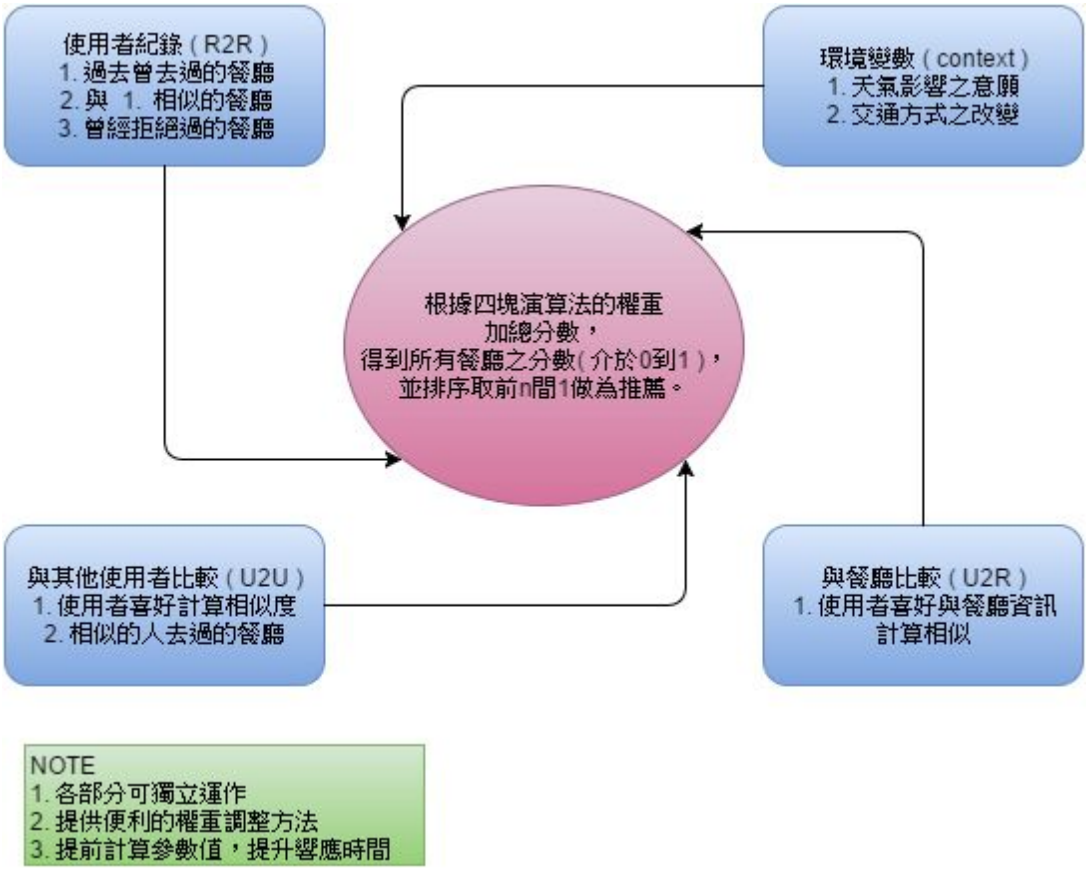

圖 3-1 演算法概念圖

3.2 演算法模塊

以下分為四塊 module 介紹各模組之輸出入與計算公式。

演算法中大量使用兩向量夾角的 cos 值做為相似度，其公式為：

$$\vec{a}\cdot\vec{b} = |\vec{a}|\,|\vec{b}|\cos\vartheta$$

圖 3-2 相似度 cos 公式

以下篇幅中 cos (x1, x2) 表示對 x1 及 x2 此兩個長度相同之向量計算夾角取 cos 值。

3.2.1 R2R 模塊

用於計算任意兩餐廳之間的相似度。

輸入 : distance , priceA , orderingA , cuisineA , priceB , orderingB , cuisineB
(任意兩餐廳之參數資訊)
距離相似度公式 : distance_sim = 1 - distance^2 （小於0則為0 ）

輸出 : sim = distance_sim + cos (priceA, priceB) +
cos (orderingA, orderingB) + cos (cuisineA, cuisineB)

3.2.2 U2R 模塊

用於計算使用者喜好與餐廳之相似度。

輸入 : U_price , U_ordering , U_cuisine ,
R_price , R_ordering , R_cuisine
(U為使用者，R為餐廳)

輸出 : sim = cos (U_price, R_price) + cos (U_ordering, R_ordering) +
cos (U_cuisine, R_cuisine)

3.2.3 U2U 模塊

用於計算任意兩使用者喜好之相似度。

輸入 : priceA , orderingA , cuisineA , priceB , orderingB , cuisineB
(A,B兩使用者喜好)

輸出 : sim = cos (priceA, priceB) + cos (orderingA, orderingB) +
cos (cuisineA, cuisineB)

3.2.4 環境變數模塊

目前環境變數部分以紀錄為主，搜集足夠資料後再進行實作及參數調整。

3.3 推薦系統流程圖

推薦系統主要函數之流程圖，前端將資料傳送至後端，將先檢查資料是否正確，與是否開啟進階功能，再將餐廳分數初始化，從各模塊中取得各餐廳分數，加總排序之後取分數前10名作為此次推薦名單。

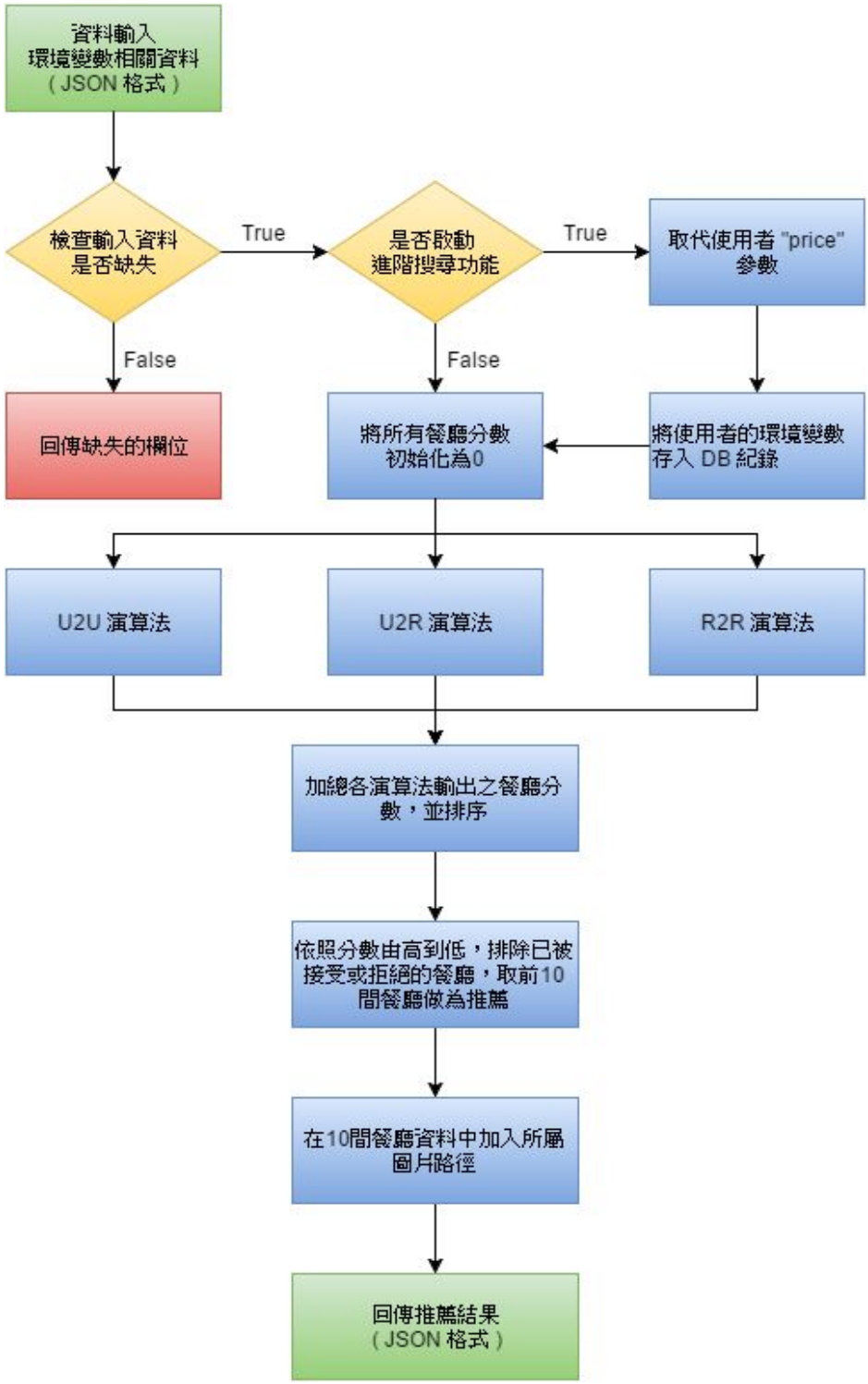

圖 3-3 推薦系統流程圖

3.4 後端 API 節點

後端節點共有10個，但大部分節點僅為內部測試使用，本節介紹三個主要節點之流程圖與概念。所有節點的 API 文件均列於附錄中，可參照查閱。

3.4.1 User_choose

此節點為使用者對某餐廳做出喜歡或拒絕的決定時，前端回傳資料給後端所使用的節點。首先做輸入檢查，接著將記錄存入資料庫，最後再更新資料庫中此使用者的喜好參數。

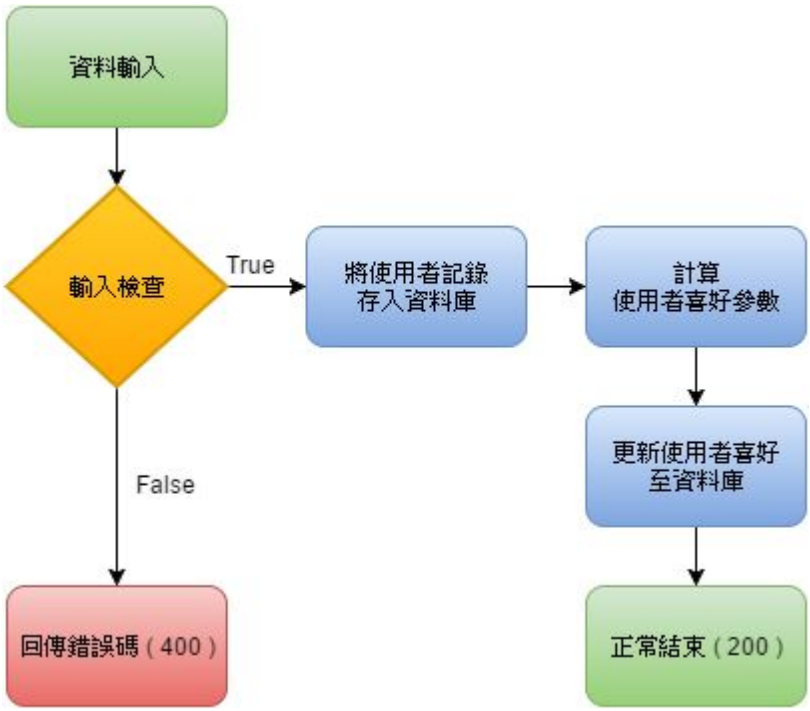

圖 3-4-1 User_choose 節點流程圖

3.4.2 Collections

此節點提供兩個請求方法（requests method），分別是「GET」、「POST」，若為前者後端將回傳此使用者所收藏之餐廳資訊，後者則代表使用者收藏某一餐廳，後端將其收藏資訊存入資料庫。

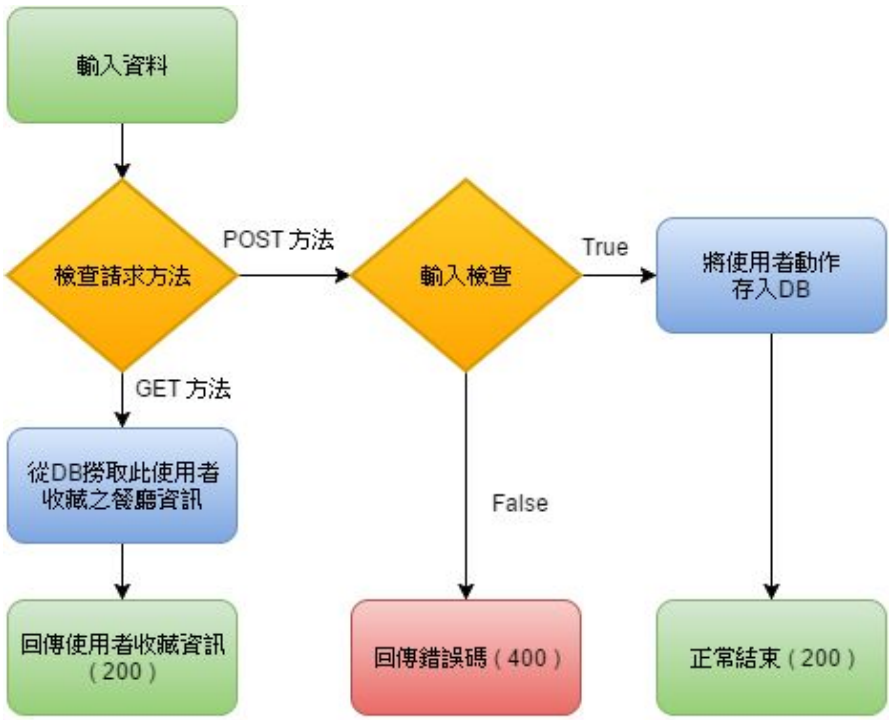

圖 3-4-2 Collections 節點流程圖

3.4.3 Signup

此節點為新使用者註冊節點，首先檢查資料是否有缺失，再嘗試新增使用者，此時若資料庫中已有此使用者會回傳錯誤，新增完畢後再更新使用者喜好參數以及U2U相似度矩陣。

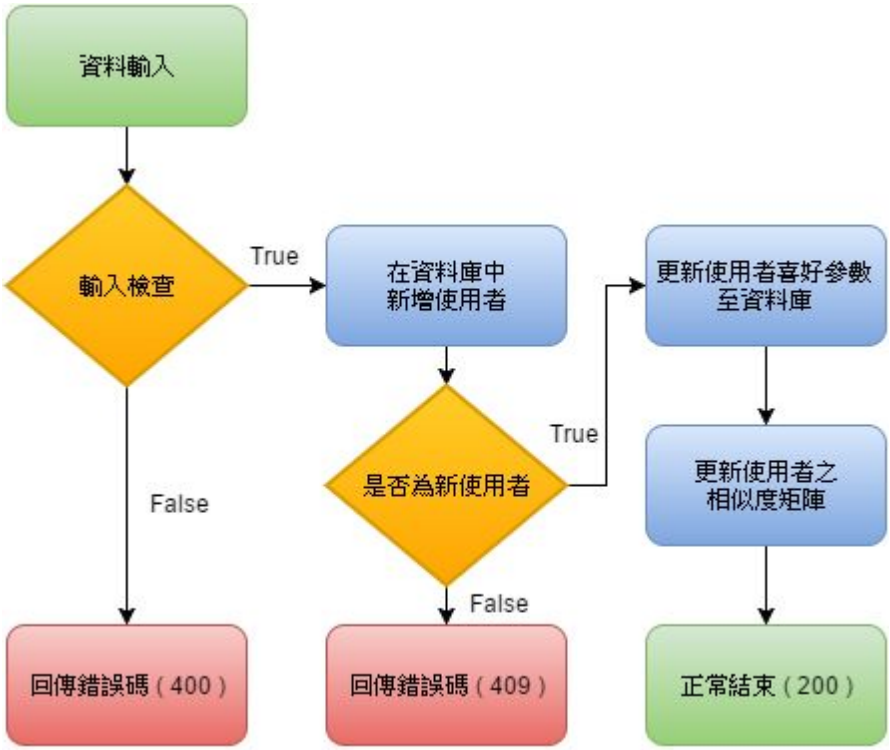

圖 3-4-3 Signup節點流程圖

4. 資料庫

資料庫的部分主要是視前後端的需求，建立所需的表格並撰寫程式以擷取儲存的資料，以下分別介紹表格設計、問卷調查、餐廳資料蒐集、伺服器管理、資料擷取五部分。

4.1. 表格設計

以關聯式資料庫的架構建立多個表格，主要的數個表格設定主鍵後，再由其他表格設定外鍵進行參考，建立表格時必須先考慮清楚想要記錄那些資料，並預先建立對應的欄位，若之後不再使用的話，撰寫程式時不擷取該欄位即可，目前資料庫綱要如圖4-1。

4.2. 問卷調查

設計問卷詢問學生對於餐廳菜式、價格及各種會影響選擇餐廳的因素後，根據收到的回應計算比例，將其作為使用者喜好程度的起始值。例如100份問卷中有60人喜好中式料理，就將使用者對於中式料理喜好程度的起始值設定為0.6，另外根據每則回應的喜好菜式不同，可計算出菜式兩兩間的相似度以建立其相似矩陣。

4.3. 餐廳資料蒐集

人工蒐集台大與台科大周邊100多間餐廳的店名、地址、價格區間、菜式、營業時間等資訊，並拍攝店面照片，作為候選餐廳。

4.4. 伺服器管理

根據設計好的表格綱要，在資料庫伺服器上實際建立表格並設定相依性，可使用圖像化的介面(phpMyAdmin)進行管理會較命令列容易，待表格建立完畢後蒐集到的資料匯入伺服器。

4.5. 資料擷取

使用Python程式語言撰寫資料存取介面，由於每次與資料庫的連線需花費較多時間，因此撰寫時必須盡可能在一次的資料庫連線中將資料全部擷取下來，再一一進行處理，轉換為需求的格式，避免延遲時間過長。


restaurantInfo
restaurant_id
name
address
price
ordering
cuisine
scenario
special
phone
hours
hours_num
remark
restaurantPrice
restaurant_id
price_id
has
price
price_id
name
userPrice
user_id
price_id
has
userInfo
user_id
name
gender
user_key
password
restaurantOrdering
restaurant_id
ordering_id
has
ordering
ordering_id
name
userOrdering
user_id
ordering_id
has
userAverage
id
price
ordering
cuisine
restaurantDistance
restaurant_id1
restaurant_id2
distance
orderingSimilarity
ordering_id1
ordering_id2
similarity
U2USimilarity
user_id1
user_id2
similarity
restaurantCuisine
restaurant_id
cuisine_id
has
cuisine
cuisine_id
name
userCuisine
user_id
cuisine_id
has
userModel
id
user_id
price
ordering
cuisine
timestamp
R2RSimilarity
restaurant_id1
restaurant_id2
similarity
cuisineSimilarity
cuisine_id1
cuisine_id2
similarity
userAdvance
id
user_id
price
weather
lat
lng
transport
timestamp
tags
id
user_id
restaurant_id
tag
type
userCollection
user_id
restaurant_id
timestamp
userActivity
id
user_id
restaurant_id
run
result
timestamp
weight
id
user_id
R2R
U2R
U2U
context
R2R_distance
R2R_price
R2R_ordering
R2R_cuisine
U2R_TFIDF
U2R_price
U2R_ordering
U2R_cuisine
U2U_tag
U2U_price
U2U_ordering
U2U_cuisine
context1
context2
context3
context4
timestamp

圖 4-1 資料庫綱要圖

5. 實驗結果

5.1 資料集與定義

資料集 : 由於部分使用者在測試時有全選接受或不接受的極端值狀況，因此從資料庫中刪除上述雜訊後輸出成CSV格式檔案，在統計各項數值計算正確率。

分類 (prediction)定義 : 在推薦時我們一次推薦十間餐廳，因排名較前者分數較高，所以理論上使用者會較為喜歡，因此定義若排名前6的餐廳使用者選擇「接受」，表示我們預測正確，為 True positive，而後4間若使用者選擇「拒絕」，同樣表示預測正確，為 True negative。

使用者選擇 (choose) 定義 : 在收到餐廳資訊後，APP上使用者有三種選擇「接受 / 拒絕 / 喜歡」，「接受」表示決定前往此餐廳，「喜歡」表示暫時不前往但喜歡此餐廳，「拒絕」表示不前往也不喜歡，因此我們將「接受 / 喜歡」認定為 True，「拒絕」為 False。

5.2 統計結果

根據前節定義將資料集製成以下結果。程式碼列於附錄。

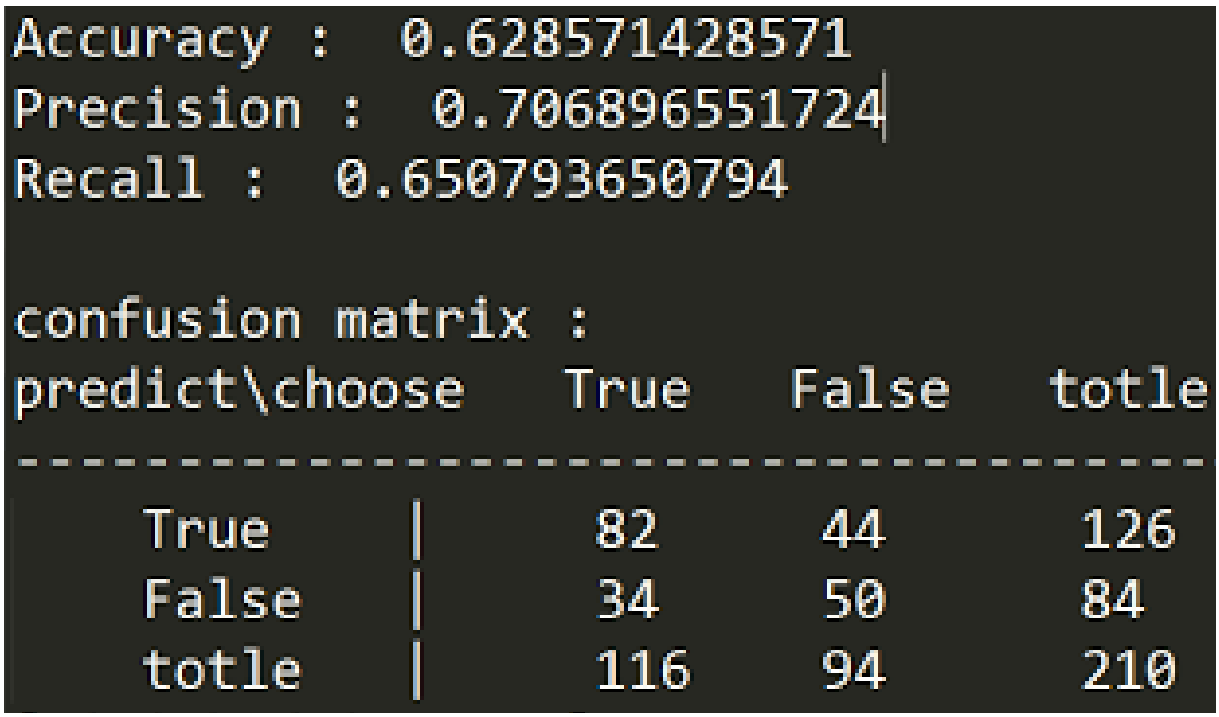

圖 5-2 實驗統計結果

6. 未來發展

6.1 前端未來發展

6.1.1 標籤系統實作

希望未來在餐廳詳細資料裡面，能夠顯示出大量被使用者重複標籤餐廳的標籤提供使用者進行更進一步的特色參考，並且能夠於評分時，新增自己的標籤，或是對已存在的標籤進同意或確認的動作。

6.1.2 評分系統實作

區隔使用者個人評分以及大眾綜合評分，目前這部分僅限於實作出畫面，於顯示區隔上，以及背後與後端串接真正功能，需要再多投入時間完成。

6.1.3 Google地圖串接

在詳細餐廳的資訊畫面上，由於提供地址太不直觀，因此希望一點進來的時候，可以串接Google地圖，直接顯示使用者與餐廳距離以及路線，幫助使用者更方便找到餐廳位置。

6.1.4 產品導向方面

就產品導向發展而言，簡單來說針對資料收集這部分提出的疑慮，因為團隊內並無設計師以及行銷人，因此我們不知道這樣的App交互方式或是我們所提供的服務，到底有沒有真正符合使用者需求，若有的話，這樣的交互方式是不是為人所接受，這是我們面臨到第一個關於使用者體驗及需求的問題，這方面的話可能得交由設計師進行一個更深入且全面的使用者訪談才能更進一步確認。另一部分則是建立在我們提供的是使用者需要的服務前提之下，對市場進行一些刺激以及商業開發，接觸到更多使用者之後，後端才有更多的資料可以進行進一步的分析與優化演算法。

6.2 演算法未來發展

6.2.1 餐廳評價

目前餐廳評價的部分從前端回傳之後，將會儲存進資料庫，並未參與餐廳分數的計算，或是權重更新。未來可增加在R2R 模塊的部分，作為此部分計分的依據，或是將評價較好的餐廳推薦給其他使用者，並標記為「其他人覺得很棒的！」，增加互動性及可信度。

6.2.2 餐廳標籤系統

目前在 U2R 模塊中標籤系統並未實作，僅預留參數權重，未來可將餐廳標籤加入前端，讓餐廳資訊更具有參考性，也可提升APP互動性，餐廳推薦的可信度。演算法部分可以根據使用者喜好的標籤與餐廳標籤計算相似度，在長期使用此系統的情況下，可以逐漸提高標籤權重作為推薦依據。

6.3 資料庫未來發展

目前餐廳僅有100多間，且限定在公館地區，期望未來能夠自動擷取美食網站上方的餐廳資料，增加候選餐廳的數量，並蒐集該餐廳的使用者評論，在標籤系統中預先新增數項標籤。

7. 附錄

7.1 工作分配表

組員	工作項目
許雲翔	後端演算法模型及開發、伺服器部署及維護、資料蒐集
張志遠	後端伺服器架設及開發、資料蒐集
廖慶麟	UI設計、iOS 應用程式開發、資料蒐集、外部測試者蒐集
王奕棋	資料庫伺服器架設、管理、撰寫資料存取介面
樓俊豪	資料庫伺服器管理及匯入、資料蒐集

7.2 後端 API 筆記

POST: /user_trial // 送回答案
POST: /user_choose // 使用者決定是否要這個推薦
POST: /users/{user_id}/recommendations // 取得推薦
POST: /users/{user_id}/ratings // 評價餐廳
POST: /users/{user_id}/collections // 收藏餐廳
POST: /signup
POST: /test // 測試該fb用戶是否註冊過

GET: /restaurants/{restaurant_id}/ratings (uid) // 取得餐廳的評價及tag
GET: /users/{user_id}/collections/ // 取得使用者收藏列表
GET: /images/{restaurant_id } // 取得圖片

DELETE: /users/<user_id>/caches // 清除使用者暫存紀錄
DELETE: /users/<user_id>/delete // 清除使用者所有資料

節點	方法	參數	回傳結果
/user_choose	POST	{ “user_id":”123456”, “restaurant_id":”1234123”, “decision": “accept”	“decline” “run”: (number), // 推薦的順位 “distance”: 123 // 使用者跟餐廳的距離（公尺） }	200: 正確 400: 傳送參數名稱有誤或不合規範的值
/users/{user_id}/recommendation	POST	{ "advance": [false] | true, **// Required when advance is true** "prefer_prices": [0, 1, 1, 1, 0], // 價格向量	[{ 'restaurant_id': 1, 'name': '甘丹鐵板燒', 'hours': '週一~週五: 10:30~14:30 17:00~22:30\n週六~週日: 10:30~22:30',

		"weathe"r: “raining” | “sunny” | “cloudy”,
		"transport": “”,
		// Optional，純記錄用
		"lat": “”,
		"lng": “”,
		}
			'scenario': ['一般'],
			’tags': {'老饕系列’: 1, '九層塔蛋’: 3, '免費湯’: -2, ‘飲料’: 0},
			‘remark’: [‘備註’, '事項'],
			'price': '100~300元',
			'ordering': ['單點'],
			'cuisine': ['燒烤'],
			'phone': '02-2362-1183',
			'address': '台北市中正區汀州路3段208號’,
			‘images’: [123, 1234]
			},
			{...}, ...]

/users/{user_id}/ratings	POST	{
		“restaurant_id”: 123
		“rating": 1~5 // 暫時沒有
		“tags": {“Good”: 1,
		“Tasty”: -1, …} // 所有使用者加入的tag陣列，key為tag名稱；value為1或-1，表示使用者喜不喜歡
		}
			200: 正確
			400: 傳送參數名稱有誤或不合規範的值

/users/{user_id}/collections // 取得使用者收藏	GET	None	[{ 'restaurant_id': 1, 'name': '甘丹鐵板燒', 'hours': '週一~週五: 10:30~14:30 17:00~22:30\n週六~週日: 10:30~22:30', 'scenario': ['一般'], ’tags': ['老饕系列', '九層塔蛋’, '免費湯', '飲料'], ‘remark’: [‘備註’, '事項'], 'price': '100~300元', 'ordering': ['單點'], 'cuisine': ['燒烤'], 'phone': '02-2362-1183',

			'address': '台北市中正區汀州路3段208號', ‘images’: [123, 1234] }, {...}, ...]

/users/{user_id}/collections
// 收藏餐廳
	POST	{
		“restaurant_id”: 123,
		“run”: (number)
		}
			200: 正確
			400: 傳送參數名稱有誤或不合規範的值

/images/{restaurant_id}	GET	None	binary data

/signup	POST	{
		“user_key": “user_key from front-end"
		“user_trial": {
		“rid”: true|false
		} // 接不接受測試的餐廳
		“name”: [string],
		“gender”: “M” | “F”,
		
		}
			{
			“user_id”: 123456
			}

/restaurants/{rid}/ratings	GET	**optional:**
		user_id=“123456” //如果有帶則會有yours，也就是自己對這間餐廳的評價
			{
			“average_rating”: (real)
			"like": {
			"tag": (count)
			},
			"disklike": {
			"tag": (count)
			},

			"yours": {
			"like": {
			"tag": (count)
			},
			"disklike": {
			"tag": (count)
			}
			}
			}

/test	POST	{“user_key”: 123432412 }	{“user_id”: 123456 } or 404 // 使用者尚未註冊
/users/<user_id>/caches	DELETE	None	{ "success":True }
/users/<user_id>/delete	DELETE	None	{ "message": "delete UserInfo success!" }

Food diviner Backend API note (http://ppt.cc/ZrTi7)

7.3 結果統計程式碼

```
# -*- coding: utf8 -*-
import csv

with open("project_data.csv","r") as df:
    TP, FP, FN, TN = 0.0, 0.0, 0.0, 0.0
    totle = -1 #header is not data
    for row in csv.reader(df):
        totle += 1
        if row[3] == "T":
            if row[2] == "1":
                TP += 1
            elif row[2] == "-1":
                FN += 1

        if row[3] == "F":
            if row[2] == "1":
                FP += 1
            elif row[2] == "-1":
                TN += 1



    print "Accuracy : ", (TP+TN) / totle
    print "Precision : ", TP / (TP+FP)
    print "Recall : ", TP / (TP+FN)
    print ""
    print "confusion matrix :"
    print "predict\choose    True   False   totle"
    print "-------------------------------------"
    print " True    |     %d     %d      %d" % (TP, FN, TP+FN)
    print " False   |     %d     %d      %d" % (FP, TN, FP+TN)
    print " totle   |     %d     %d      %d" % (TP+FP, FN+TN, totle)
```

圖 7-3 結果統計程式碼