巨量資料與金融科技實務

聊天機器人_(Calendar**)**

組員:

05170132 巨資四A 魏嗣宸

05170182 巨資四A 王彥儒

06170211 巨資三B 李育綺

06170225 巨資三B 陳昱昇

06156222 資三B 陳韻安

目錄

一. 聊天機器人簡介

聊天機器人已充斥在現代人的日常生活中，透過自然語言模擬人類，進而與人類溝通，而聊天機器人根據功能的不同可分為4個種類，分別是問答系統、任務導向型對話系統、閒聊系統以及推薦系統。

1. 問答系統

基於用戶提供的問題，給予固定的回答，不涉及到多輪的回答，通常提供使用者操作上的問題解決或使用說明。問答系統的聊天機器人幫我們省去閱讀與理解資訊的過程，而直接給我們專屬於自己，想要的答案。

2. 閒聊系統

日常生活中的使用者隨意地和聊天機器人對話，而機器人會根據後台設定給予使用者相對應的回答，閒聊系統可讓使用者使用起來更親近於真人，體驗更具親切，且具擬真的效果。日常生活中的微軟小冰、小米小愛同學、蘋果Siri都有搭載閒聊功能。

3. 推薦系統

推薦功能應用，通常都會搭載在通訊軟體上，讓客戶可以即時接收到相關資訊。透過通訊軟體上的聊天機器人，店家可以確保顧客即時接收商品資訊，且也可透過後台數據追蹤其行銷成效，是目前網路行銷的新興手法之一。日常生活中LINE、Messenger 上的官方帳號，便屬此類型。

4. 任務導向型對話系統

同時也是本次專案之聊天機器人的類別。

任務導向型對話系統可以透過使用者的要求，完成使用者指定的任務。生活中蘋果的Siri就是其典型的代表，它可以幫助我們搜尋、預定機票、設定鬧鐘等等。

二. 主題

本次專案之聊天機器人類別為任務導向型對話系統，可協助使用者完成特定功能，主題定為提供行事曆功能的聊天機器人，我們將它取名為Calendar。顧名思義，這是一個具有新增、修改、刪除、查詢代辦事項的私人時間管理助理。不同於市面上常見的行事曆APP，在這基礎架構上，我們添加並優化了一些功能。

像是由於我們是部屬在Line上的服務，因此能滿足不論是IOS系統，抑或是Android系統的手機用戶，一來不僅能避免潛在使用者因為系統差異而無法享有此服務，二來假設未來此聊天機器人能上市，也能因為不受系統限制而擴大用戶範圍，進而增加商機。

此外我們觀察到，雖然現有的電子版行事曆都具備"行程提醒"功能，但並非每個用戶都會單純希望在特定時間以前收到有限次數的貼心小提醒，舉例來說，假設某位使用者在新增代辦事項的當下設定提前一天通知，然而我們卻無法保證在使用者收到提醒之後，到行程當天這段期間內，可以有充裕的時間準備可能需要用到的物品。因此我們在每次用戶進行"查詢"此功能時，都會依據剩餘時間，由少到多開始顯示已新增的每筆代辦事項，如此一來就可以無意間達到不斷提醒的功效，讓使用者有緩衝時間為行程做準備，更可以讓用戶依照事情緩急來安排先後順序，幫助他們做好資源分配。

最後，與一般行事曆不同之處在於，使用者可以在"查詢"的當下自行選擇是否查看過往歷史資料(此處亦是按照時間發生先後排序)，如此一來使用者就能依照自身需求，快速選擇是要調閱過去的預定行程還是未來某個時間點的代辦事項。

三.　動機

隨著科技的發達，傳統紙本行事曆逐漸被各類行事曆APP取代，架上行事曆APP五花八門，功能也不盡相同，像是較為人所知的Google日曆、TimeTree...等。雖然

市面上已存在的APP發展的相當完善，然而，並不是每位新用戶都願意特地下載一個APP來幫助他完成"時間管理"的需求，且無可避免的是不同手機版本間可能無法相容(像是特定APP只對Andriod手機提供服務)如此一來勢必無法滿足所有用戶的需求。此外，在忙碌緊湊的生活步調中，即便已經用行事曆記下行程，行事曆的複雜度仍會隨著工作量增大而不斷提升，有時甚至會多到讓重要行程混雜在其他項目中，增加被無意間遺漏的風險。此時若有一個能讓使用者一目了然的查詢介面，就能大大降低此機率。

於是，我們想到了嫁接在Line上的聊天機器人，由於Line本身就提供一個簡潔的使用者介面，因此所有資料都能藉由"訊息"方式呈現，在我們的設計下，查詢結果能依照預定行程之時間發生先後順序顯示，用戶完全不用擔心遺漏任何一筆代辦事項。此外，使用者更無須開多個應用程式，一切的操作都可以在單一畫面上完成，非常方便。

四. 功能介紹

查詢、刪除、修改流程

當使用者按下查詢後，首先流程會先判讀使用者是否有資料，並且分成以下兩種情況：

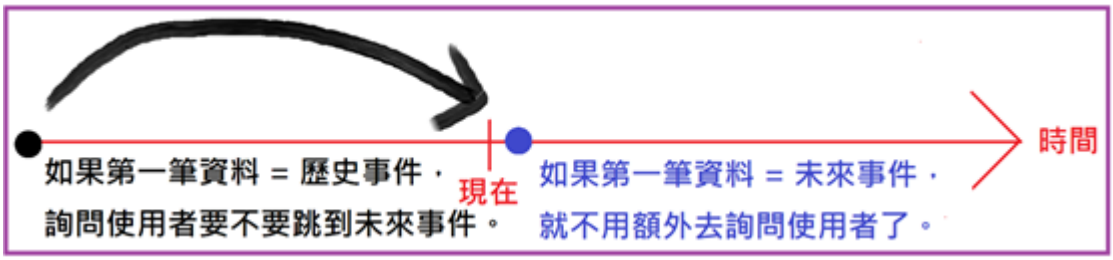
1. 沒有資料(有可能是從來沒新增，或者是將所有的資料都刪除了)，聊天機器人會發送一段提醒文字，並且詢問使用者要不要新增資料。

2.有資料：由於在新增的步驟，我們已將資料按照時間大小進行排序，故之後的決策又會分成兩種情況：

2-1.第一筆資料為歷史事件，我們的聊天機器人會詢問使用者是否要跳過歷史事件，直接查看未來事件。如果使用者選擇【是】，那麼就會直接跳到2

-2的流程，反之則會一筆一筆查看歷史事件。這樣的設計便可避免使用者不想查詢歷史事件，但是卻又花很多時間在一筆一筆查看的窘境。

2-2.第一筆資料為未來事件，由於已經按照時間已經排序過，所以每次出現的第一筆資料必定為最快發生的代辦事項，可以達到提醒使用者的效果。



以下為使用者在查詢時會看到的介面:

下一筆：讓使用者查看下一筆資料

刪除：和一般在刪除檔案的情況一樣，按下之後，會再詢問一次使用者是不是真的要刪除，使用者雙重確認之後，才會真的刪除檔案，如使用者按了刪除，又按了否，則會跳到下一筆資料。

修改：類似刪除的流程，按下之後，會再詢問一次使用者是不是真的要修改，使用者雙重確認之後，會先將此筆資料刪除，再帶使用者來到新增的流程。

不看了：即時的讓使用者中斷查詢的流程。

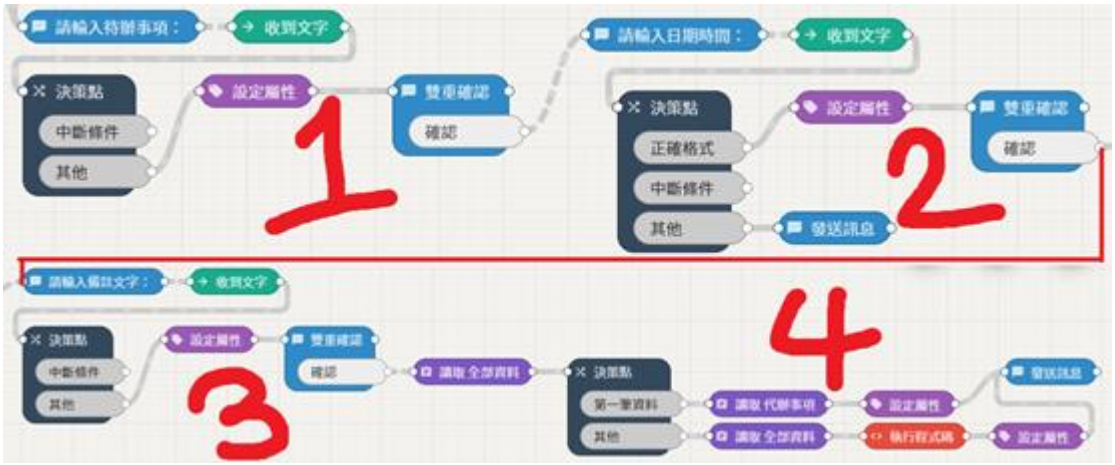
以上為查詢、刪除、修改的流程圖。

五. 製作方法

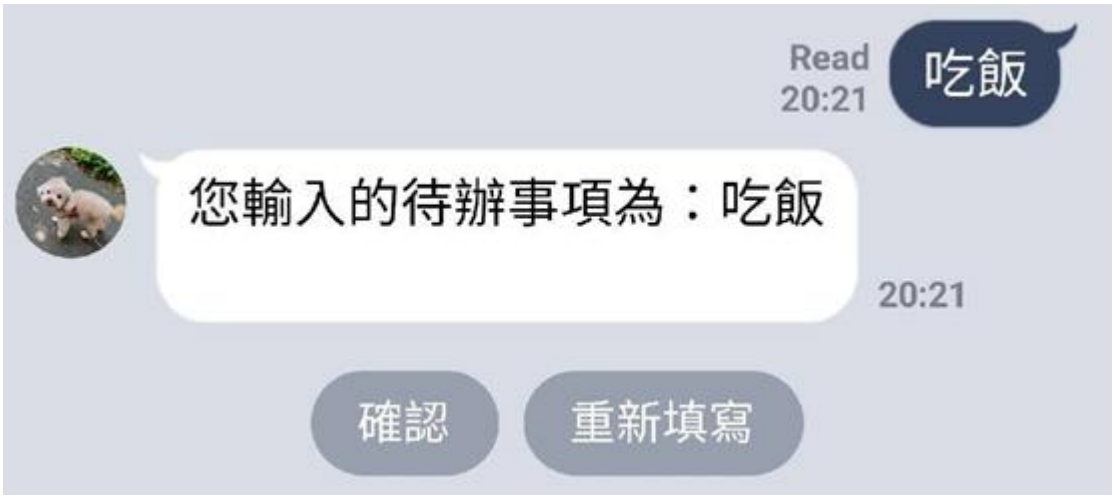
- 新增

  在拉流程的時候，我們發現可以為每個使用者建立多種屬性，且存放進屬性的資料，是不會被清除的！我們決定用設定屬性的方式，來儲存使用者的資料。Gosubar的【設定屬性】節點中，有三種資料結構：數字、文字、列表。用程式語言的方式理解，就是int、string、array。

  讓使用者依序輸入代辦事項、日期時間、備註文字，除了日期時間需要固定格式(這樣才可以用程式碼轉換成時間格式，並且進行後續時間大小的判斷、排序)，其餘兩個，我們並不會去要求使用者填的文字，只要不跟本聊天機器人的指令衝突即可。此外，我們也有簡單的防呆機制，如果使用者輸入的日期時間格式是錯誤的，或者是使用者打錯字的話，我們可以讓使用者重新輸入。(以下為新增的流程)



【雙重確認】：為了防止使用者不小心錯誤，所以每次使用者輸入完一個部分後，我們的聊天機器人會再向使用者確認一次他輸入的文字是否正確。使用者點選【確認】之後，才能繼續輸入下一個部分，反之則會重新輸入。



【決策點】中斷條件：當使用者輸入【查詢、新增、說明、圖文選單、重新填寫】，會強制中斷目前的流程，切換到使用者輸入的流程，如此可避免無窮迴圈的情況，也可以讓使用者強制中斷目前的流程。

正確格式：當輸入日期時間的格式為YYYY-MM-DD HH:mm(Ex:2020-04-05 12:36)且為未來時間，才會讓使用者接著輸入備註文字，否則就重新輸入。

實現按時間日期順序排序的程式碼:

假設原列表 = [約會,2020-05-31 12:00,和女友]，使用者新增了一筆資料 = 吃飯,2020-05-29 18:00,和家人，由於5月29日比5月31日還要早，所以這筆資料就會新增到列表的最前面，所以新的列表 = [吃飯,2020-05-29 18:00,和家人,約會,2020-05-31 12:00,和女友]。如此一來，便可確保每次在查詢的時候，第一筆資料總會是時間最小的。

```
var det = 0
for(i=1;i<=tmp.all.length;i+=3){
    if(Date.parse(tmp.time)<=Date.parse(tmp.all[i])){
        tmp.all.splice(i-1,0,tmp.activity,tmp.time,tmp.ps)
        det = 1
        break
    }
}
if (det == 0){
    tmp.all.push(tmp.activity)
    tmp.all.push(tmp.time)
    tmp.all.push(tmp.ps)
```

註:

根據我們的研究，Gosubar執行程式碼的節點，底層架構是JavaScript語法。JS提供豐富的函式庫，如Date.parse可以將時間格式轉換成數值，如此便可比較時間的大小。Splice則是可以移除特定index區間的資料，並且同時新增指定的資料進列表。

六. 技術問題

1. Gosubar提供了連結Google Sheet表單的方法，因此原本我們想使用Google Sheet的方式建資料庫，並將使用者輸入的代辦事項、日期時間、備註文字，一併上傳到資料庫儲存，再藉由執行API的方法，將整份Google Sheet的內容抓下來，存放到tmp變數提供開發者使用，來完成"查詢"的流程(概念如下圖:)

然而，當我們實際測試的時候發現新增完的資料，並不會馬上被查詢到，需要等個1~5分鐘再查詢。也就是要執行API，請求整份表單的這個步驟，並不會去抓最即時的表單內容。我們推測，有可能是網站為了減輕負擔，所以第一次查詢的時候，就會先把請求下來的資料先暫時存放，下一次查詢，如果時間沒有間隔太久，就會直接把暫時存放的資料回傳，如此便可減少一次執行API請求資料的時間。

因此，若是可以解決無法及時抓取Google Sheet表單內容的問題，說不定能使Gosubar更加完整。

2. 我們的流程不管使用者目前輸入到代辦事項、日期時間或備註文字，只要使用者輸入錯誤要按下【重新填寫】，一律就是回到代辦事項從頭輸入(概念如下圖)，這樣的設計不夠友善。因為我們希望達到的目標為：使用者在輸入備註文字的時候，如果輸入錯誤，那麼只要針對備註文字再重新輸入即可。

我們發現Gosubar對於【收到文字】節點的設計，可以選擇是否為額外事件，(額外事件代表，當使用者執行到此節點的時候，才會開始偵測事件。)而當勾選額外事件之後，又必須指定此節點的可執行的次數，以及此節點的執行模式為併發還是獨佔。獨佔代表當此節點被觸發之後，之後相同類型(收到文字)的節點都不會運作，反之就是併發。

首先，我們必須把新增流程的三個【收到文字】節點設定為額外事件，如果沒有設定為額外事件，那麼此節點就會變成起始點，意思就是，當收到使用者傳來的任何文字訊息，會同時從三個起始點進入流程，這樣就會造成無窮迴圈的情況。

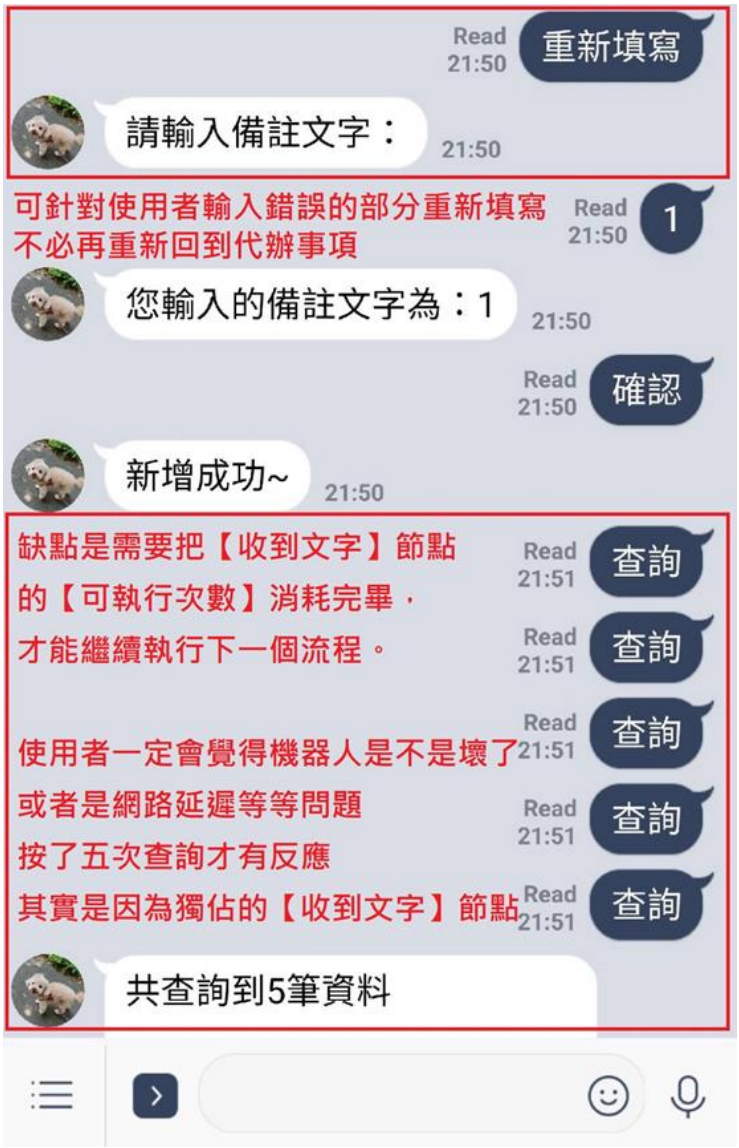
因此我們將那三個節點設為額外事件，但又因為必須設定【可執行次數】，然而【可執行次數】在我們的專案下，由於不能限制使用者最多可以輸錯幾次，因此沒有實際的價值。再來又有一個問題，這三個【收到文字】節點，要設為併發還是獨佔呢？如果設為獨佔，那麼使用者就無法臨時切換到其他流程(例

如：查詢、說明)，必須要等到這個【收到文字】的節點的執行次數被消耗完畢，或者是下一個【收到文字】的節點被觸發之後，這個節點才會停止獨佔。

假設我們將這三個【收到文字】節點設定為【併發】，就會出現這個問題：

由於【收到代辦事項文字】的節點已經被觸發，加上又不是獨佔，所以它就會變成開放式的起始點，任何文字訊息都會從這邊進入流程，同時也會從【收到日期時間文字】的節點進入流程，導致迴圈的情況產生，故併發也是不可行。

下圖為如果將新增的三個【收到文字】的節點設為【獨佔】，會發生的情況：


Read 21:50
重新填寫
請輸入備註文字：
21:50
可針對使用者輸入錯誤的部分重新填寫
不必再重新回到代辦事項
Read 21:50
1
您輸入的備註文字為：1
21:50
Read 21:50
確認
新增成功~
21:50
缺點是需要把【收到文字】節點
的【可執行次數】消耗完畢，
才能繼續執行下一個流程。
使用者一定會覺得機器人是不是壞了
或者是網路延遲等等問題
按了五次查詢才有反應
其實是因為獨佔的【收到文字】節點
Read 21:51
查詢
共查詢到5筆資料


起始點
事件類型
接收到文字訊息
額外事件
可執行次數
1
存活時間(秒)
存活對象
使用者
執行模式
併發
在等待使用者回應的過程中，變數資料將會重置。
如有需要保留的資料，可以透過設定屬性結點將資料儲存
你將會有以下變數可以使用
profileId : 3
確定

雖然Gosubar在【收到文字】的【額外事件】也有提供【存活時間】，但是我們不能得知使用者輸入這三個東西需要花費多久時間，我們也不可能去限制使用者必須在幾秒內輸入完畢，否則流程就會終止，這樣的行事曆太不實用了。

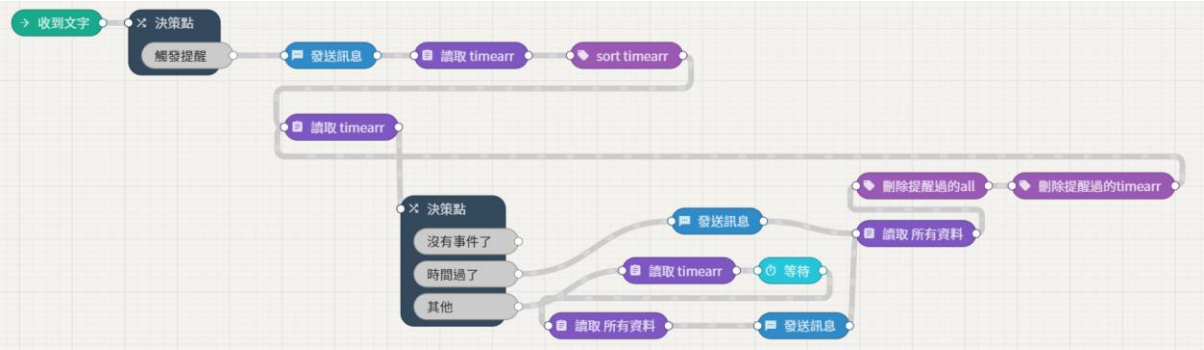
下圖統整出新增的流程，三個【收到文字】節點的狀態，分別的優缺點：

	額外事件(併發) 可執行次數>1次	額外事件(獨佔) 可執行次數>1次	額外事件(獨佔/併發) 可執行次數=1次 最終採用的
優點	無	當使用者已經到步驟3的時候，如果不小心輸入錯誤，只需從步驟3重來，不必再回到步驟1	可即時轉換、中斷流程
缺點	無窮迴圈	無法即時轉換、中斷流程(雖然可以透過設定可執行次數，但是妳沒辦法確定使用者會輸錯幾次)	當使用者已經到步驟3的時候，如果不小心輸入錯誤，必須從步驟1重來，不能直接回到步驟3

可以看到，額外事件(獨佔)跟額外事件(1次)其實是剛好互補的，兩者都只能解決其中一個問題，但我們最後選擇的是【額外事件1次】。雖然流程會讓使用者覺得不太方便，但是至少不會有Bug產生。如果Gosubar在【收到文字】的【額外事件】能有更多的彈性讓開發者選擇的話，那麼開發者們製作出來的聊天機器人將會更多元、更符合使用者的需求。

3. 時間流程的問題。

此為時間流程的架構圖



當收到文字，判斷執行該流程以後，把timearr (使用者的屬性之一，儲存所有事件的發生時間點，是一個陣列) 裡的時間由小到大sort過一遍，再於第二個決策點判斷如果timearr長度為零，則跳出迴圈，如果這一筆時間已經過了，就推送「遲來的提醒」，否則就設定等待時間，時間一到就立刻推送「準時的提醒」通知。之後再於all(使用者的屬性之一，儲存所有事件的資料，是一個陣列)刪除該時間的那筆提醒，和timearr的該筆時間。接著回到第二個決測點，判斷timearr的下一筆時間。

其中遇到的問題有：

1. GOSUBAR無法同時執行多個流程，因此只能用一個迴圈流程去設定提醒

   一開始，我在新增流程結束以後，安排推送提醒的流程，在使用者輸入的時間當下，即推送訊息提醒。此刻，若馬上執行其他流程，前一個「新增」流程尚未完成(因為必須隔一段時間，才會提醒)，而新的流程便覆蓋掉前面的流程，造成先前輸入的那筆資料，就永遠不會被提醒到了。

   因此，我用「迴圈」的方法解決這個問題(參見上圖流程)。凡是有進行修改使用者all屬性的流程，例如：新增、修改、刪除，結束以後，觸發「報時迴圈」的流程，讓它會一直抓下一筆資料，不會被中斷。而這種做法，雖然解決了流程覆蓋問題，不過也衍伸出以下的問題。

2. 迴圈時間拉太長，產生穩定性問題

   迴圈的做法，可以想像成為，在使用者不使用該聊天室功能的期間，不停地執行該報時提醒的迴圈，直到把陣列裡面所有的事件都提醒完，並且清空了以後，才會結束。因此，這個迴圈可能會執行好幾個小時，甚至好幾天，其中可能遇到許多讓流程中斷的因素，造成後半段的流程都沒有被提醒到。

3. 提醒會lag的問題

   當使用者沒有開啟Line的時候，Line提醒會延遲，那麼此機器人就無法在正確的時間點，提醒使用者。不過，同樣的問題也困擾著其他的行事曆app。例如：微軟的Outlook行事曆，也有推送提醒的功能，偶爾也無法在正確時間提醒。因此，這個問題，可能是很多app的通病。

   而為了彌補這個問題，我也在上圖中的第二個決策點，增加「時間過了」的判斷項目，一旦流程發現時間已過了，流程會馬上推送「遲來的提醒」，提醒使用者該筆資料。

4. 跑不動

這是報時提醒功能無法完成的致命傷。該流程一個小時前測試，可以準確無誤地執行，而一個小時後，再度測試時就發現出了狀況。它會跳過推送提醒的步驟，直接執行後面刪除陣列裡的資料。對於這個問題，我只能想出一個原因：「推送提醒」與「新增、修改、刪除、查詢」相比，需要執行更多CPU-bound的程序，代表需要處理較大量的資料運算(包括：使用者屬性的排序、刪除)。而那些流程背後的程式碼，都存在GOSUBAR平台後端的伺服器裡，運作流程時，必須連接到那雲端的伺服器抓程式碼，經過一連串的過程，最後才到使用者端推送提醒。因此，電腦真實做的事情，比流程上呈現的多很多。所以伺服器吃的運算過重，才會產生如此的問題。

七. 工作分配表

聊天機器人製作	陳韻安,陳昱昇
書面報告製作	魏嗣宸,李育綺,陳昱昇
PPT製作	李育綺,王彥儒
口頭報告	李育綺

八.參考資料

- 原來聊天機器人可以做到這些？四種功能，七大案例，五分鐘認識各大企業推出的有趣 Chatbot！
- 「NLP-ChatBot」我們熟悉的聊天機器人都有哪幾類？