元智大學資管系

第三十屆學術類畢業專題頂石課程(二)

期末報告

行得通-智慧共享乘車 APP

1111730 邱育宏、1111751 陳佳佑

實習公司：校內專題

工作代號：ZU2

指導教授：楊錦生 教授

中華民國 114 年 11 月

Nov, 2025

Content

Chapter 1 緒論……...3

Chapter 2 相關技術與研究……...5

2.1 相關技術一(資料查閱)……5

2.2 相關技術二(主要開發工具)……8

2.3 相關技術三(次要開發工具)……8

Chapter 3 研究方法……10

Chapter 4 實驗結果與系統展示……31

4.1 系統展示……31

Chapter 5 結論……..37

5.1 系統回顧……...37

5.2 未來發展……..37

附錄 **A** 專題工作內容……...39

附錄 **B** 專題心得與建議……...40

Chapter 1 緒論

背景說明

近年來，隨著都市化發展與人口集中，交通擁擠、停車困難與環境污染等問題日益嚴重。而共享經濟的崛起，使得共乘（Carpooling）成為替代私人交通工具的重要手段之一。透過數位科技與行動應用的普及，共乘服務可有效整合分散的交通需求，降低個人乘車成本，並對環境永續與能源節約產生正向影響。

傳統的共乘平台多著重於司機與乘客之間的一對一媒合，對於短途或多位乘客共乘的情境支援有限，難以有效促成配對。此外，對於企業接駁（活動主辦提供）的需求，也缺乏整合性的資訊平台。這些現象顯示，現行共乘系統在自由度、彈性與配對智慧性方面仍有改進空間。

研究動機

當代都市因人口密度上升與交通需求增加，尖峰時段的叫車困難與運輸擁擠已成為普遍現象。學生與上班族在上下課/班時段、交通樞紐周邊，或天氣不佳的情況下，常面臨等待時間過長、價格浮動、甚至無法成功叫車的問題。此外，許多活動主辦單位雖提供免費接駁車以提升參與率，但因資訊傳遞仰賴公告或口耳相傳，缺乏統一整合平台，加上人數難以預估與分流，導致車輛資源

未被有效運用。

另一方面，現行共乘平台多仰賴人工配對，缺乏智慧演算法支援，導致使用者常遇到「無法成單」、「時間不合」等問題，影響共乘意願。系統在上車地點的彈性不足，無法自動計算最合理的集合點，也增加使用者之間的協調難度。同時，共乘過程中成員之間的即時溝通與任務變動通知等互動機制尚未完善，進一步降低使用體驗。

以下分為幾種付費情境說明:

1. 大型活動:以演唱會及桃園燈會為例，由於人流集中或地點偏遠，可能會遇到叫不到車，或車費過高的問題，若能事先發起共乘任務，與他人搭乘不僅可分攤費用，也能提高叫車成功率。
2. 研究所考試或大學面試:外地考生最擔心的就是會不會遲到，因此傾向選擇可控性高的叫車方式，若能提前配對共乘，不僅能減少風險，也增加心理安全感。
3. 尖峰時段放學:學生常面臨等公車人潮擁擠的問題，尤其在地理位置較偏的學校（如東吳大學）。透過預約共享乘車，針對「出發地及目的地一致性高」的地點(學校及捷運站)，可以有效節省時間與成本。

而對於某些活動主辦方提供的免費接駁車服務，雖然乘客乘坐無須費用，

但人力及車都需要成本，活動方可透過共享乘車平台發起非付費任務，不僅能讓使用者知曉有該活動也能提升前往參與的意願及接駁使用率。

研究目的

本研究旨在開發一套結合「免費接駁車任務資訊發布」與「付費共乘智慧配對」的行動共乘平台，整合地圖定位、即時通訊與自動推播通知等功能，以滿足使用者多樣化的共乘需求。

研究具體目的如下：

1. 付費車種配對機制: 用戶可透過系統輸入乘車條件，系統將自動媒合條件相近的共乘者，當媒合成功後系統會自動叫車，成功叫車後通知乘客。
2. 建置任務發布與查詢機制：提供使用者發佈免費接駁車任務資訊，讓他人可依時間與地區查詢並加入，並支援即時修改。
3. 整合地圖 API 並找出上下車地點路線：使用 Google Map API 查看經緯度並計算路徑距離，以利於配對系統使用，且能妥善利用其他 API 功能，增加系統的使用體驗。
4. 實作聊天室與通知系統：每個任務對應一個聊天室，供參與者即時溝通，並在任務即將出發前自動推播提醒，提升準時率與互動性。

Chapter 2 相關技術與研究

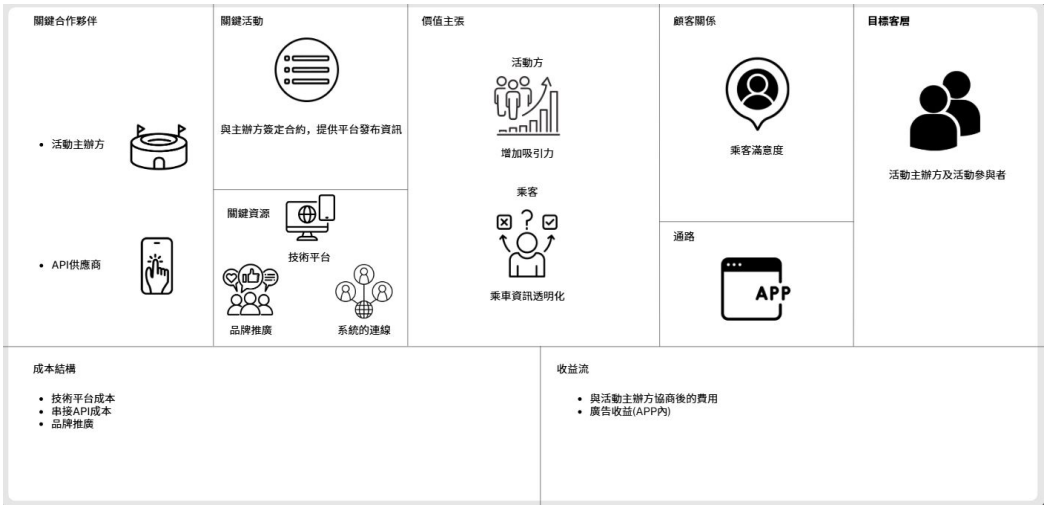
2.1 相關技術一(資料查閱)

1.Uber 商業模式分析

Uber 的商業模式特色在於以數位平台為核心，建立乘客與駕駛的雙邊市場。此模式啟發了本研究共乘系統在平台設計、路線配對與用戶互動上的核心概念，並由此建立了系統的商業模式圖，如下:

圖一(付費車種商業模式圖)

本系統的付費車種商業模式以「平台媒合」為核心，透過整合支付平台商、車行公司及 API 供應商等合作夥伴，建立穩定且具延展性的服務生態。系統主要活動包含金流運作、使用者服務、系統維護與品牌推廣，確保乘客與司機間的配對效率與服務品質。其價值主張在於提供乘客便捷、安全且具成本效益的出行體驗，同時協助司機提升車資收入並增加接單機會。目標客層涵蓋有乘車需求的用戶與具駕駛資格的車行司機。系統成本主要來自技術平台維護、乘車保險、品牌推廣，而收益則來源於訂單抽成及應用程式內廣告收入。



圖二(非付費車種商業模式圖)

本系統的非付費車種商業模式，以透過與主辦方簽約合作，讓乘客能在 APP 上查看並加入共乘行程，提升活動吸引力與交通便利性。主要目

標客群為活動主辦方及活動參與者，收益來源包含與主辦方合作的簽約費及應用程式內廣告收入，形成具社會效益的免費共乘運作模式。

2.社群媒體共乘社團或拼車發布

目前在 Facebook、Dcard 等社群媒體上有許多以共乘為主題的社群或是文章，例如「高鐵站共乘」與「上下課共乘資訊交流區」等，透過貼文與留言的方式讓乘客與駕駛進行媒合。雖然這類社群具有高度的互動性與彈性，但缺乏自動化配對、距離判斷與即時通知等功能。本系統參考其社群互動特性，整合會員制度與即時配對通知機制，提升使用者之間的互動與配對效率，並兼顧系統化與便利性。

3.相關 app 參考

(i) Tripool

Tripool 是台灣的交通平台，主打「點對點接送、計時包車與共乘」服務。使用者可輸入出發地與目的地，即時獲得報價並完成預約，系統會自動派遣合適車輛與司機接送。Tripool 也支援智能配對行程相近乘客的共乘模式，讓使用者以更低成本共享車輛，適合旅遊、

機場接送與跨縣市長途出行需求。

系統比較:

	行得通 APP	Tripool
定位	付費 + 免費共乘拼車	付費包車服務
配對機制	系統自動根據地點、時間與人數進行智慧匹配	AI 派車調度
收益來源	免費+付費共乘	包車
付費模式	按乘客人數自動分攤扣款	固定報價，整車費用
聊天室	內建聊天室與即時通知	僅提供乘客與司機聯絡
使用場景	通勤、市區短程	長途包車、機場接送
安全機制	乘客驗證碼、聊天室記錄	司機認證
價格	可依人數與距離動態調整	固定透明報價

(ii) dauding

Dauding 是台灣在地的共乘媒合平台，主打「返鄉、通勤、旅遊及順路包裹」等多元出行場景。使用者可在網站上發起或搜尋共乘貼文，車主與乘客能依出發地、目的地及時間自行媒合，共同分攤

油資與交通費。平台與地方政府合作推動如「共乘東海岸」等專案，提供更便利的智慧交通服務。Dauding 強調社群互動與共享經濟理念，致力於降低出行成本、減少交通碳排並促進共乘文化的普及。

系統比較:

	行得通	Dauding
定位	智慧共乘平台，自動配對	共乘資訊發布平台
配對方式	系統自動根據地點、時間與人數進行智慧匹配	使用者自行發文或搜尋貼文，車主與乘客自行媒合
核心功能	自動配對	共乘貼文發布
安全機制	具驗證碼上車、聊天室紀錄與通知功能	使用者信任與評價機制
使用場景	通勤、市區短程	旅遊、順路共乘
技術	App 整合式系統，自動化後端演算法匹配	網頁平台導向，偏社群式操作

2.2 相關技術二(開發工具)

1.React Native： 前端使用者介面

本系統使用 React Native 作為主要的前端開發框架，透過 TypeScripts 撰寫 APP。前端開發過程中，以 Expo 為建構工具，搭配 React Navigation 實現畫面間的切換功能，並使用 Axios 與後端進行 HTTP 請求及資料傳輸。

2.Fast API：後端 API（伺服器邏輯處理層）

接收前端的請求，進行邏輯處理，並與資料庫溝通。

主要功能有:使用者登入帳號密碼的驗證，及將前端資料寫入資料庫中與資料回傳給前端。

3.MySQL：資料庫（儲存資料）

用來儲存各種資料，包含:使用者資訊、記錄每筆乘車的訂單、非付費車種清單等資訊

透過與 FastAPI 整合，資料能即時更新並正確回傳至前端介面。

4.Android Studio：模擬實際 APP 使用情況

為了在開發階段進行測試與除錯，我們使用 Android Studio 提供的虛擬模擬器，模擬真實 Android 裝置操作情境，檢查前端畫面顯示與功能互動是否符合預期。

5.Figma：使用者介面設計工具

在系統開發初期，我們先使用 Figma 完成整體 UI/UX 設計，便於規劃畫面流程與元件設計。根據設計所產出的視覺配置與元件樣式，再透過 React Native 將畫面進行具體實作，確保實際應用與設計一致，並提供良好的使用者體驗。

6.Google map API：Google 地圖的功能使用

由 Google 提供的一套地圖服務介面，透過它在行動應用程式中整合地圖功能，例如顯示地圖、標示地點、計算路線與距離等。在本專案中，我們使用 Google Maps API 的 Geocoding 與 Directions 功能，將使用者輸入的地址轉換為經緯度，並計算起點與終點之間的實際距離，以協助共乘配對與路線規劃。

Chapter 3 研究方法

一、需求分析:

(一)尋找目前市面上與共乘相關功能的 APP，確認是否有開發之必要性

(二)透過詢問消費者的需求，確認付費和免費車種皆有市場需求

二、收益模式分析:

(一) 非付費共乘

1.假設情況: 新竹巨城之固定班次的接駁車行駛

2.運作模式 : 一次性簽約 10 萬元，期限為 1 年，期限內將可以在平台上發布相關接駁資訊

(二) 付費共乘

計算情境 : 一趟車資為 200 塊之訂單

平台抽成 : 20%

車費增加比例 : 基礎車費增加 50% + (總人數-2) * 20%

	乘客平均每人車費	司機利潤	結果比較	平台利潤
非共乘情況	200	200	-	-
共乘 2 人情況	200*1.5/2=150	200*1.5*0.8=240	乘客省下:25% 司機利潤增加:20%	60
共乘 3 人情況	200*1.7/3=113	200*1.7*0.8=272	乘客省下:44% 司機利潤增加:36%	68
共乘 4 人情況	200*1.9/4=95	200*1.9*0.8=304	乘客省下: 52% 司機利潤增加:52%	76

三、系統設計:

(一) 非付費共乘功能設計

1. 任務加入

使用者能夠透過加入任務來獲取搭車相關資訊。

2. 搜尋

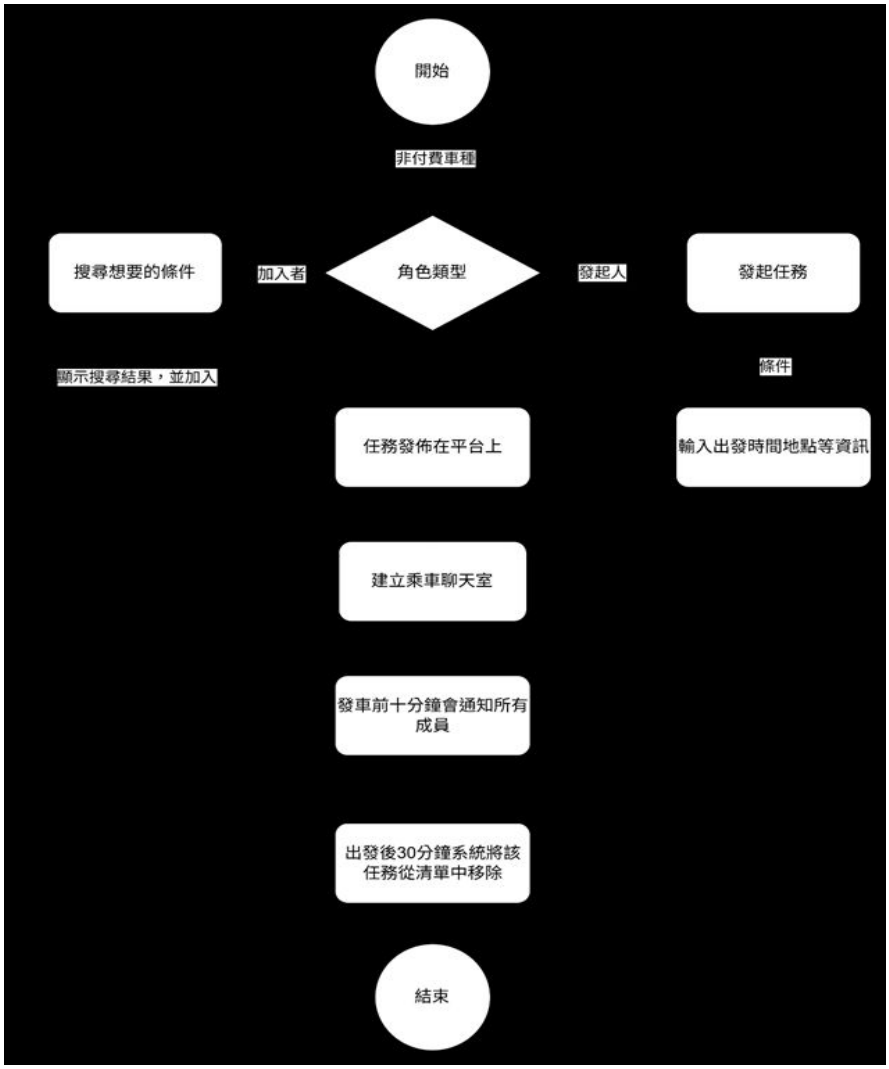
使用者可以透過搜尋框，並根據地區、相關性來搜尋符合需 求的共乘任務。

3. 聊天室建立

使用者加入後，可以進到屬於該任務的聊天室中，與也在該任務內的成員討論。

4. 通知功能

系統在出發前十分鐘會發送通知給每位共乘人員。



(圖三)非付費流程圖

流程圖說明

一開始選擇身分為 A2.a 或 A2.b，如果是 A2.b 的身分可以發起任務後會進入到 A4，A2.a 的使用者可以透過搜尋找到想要加入的選項。加入任務後，該任務成員會有一個專屬聊天室(A5)，可以一起討論相關事宜等，出發前十分鐘會通知每位成員(A6)，當出發後 30 分鐘系統會將任務從清單移除掉(A7)。

(二)付費共乘功能設計

1. 共乘任務發起

   每位用戶都可以根據出發到達地點、出發時間、共乘人數發起共乘配對。

2. 匹配共乘者

   在出發時間 30 分鐘前，系統都會持續匹配是否有符合的配對資訊，若是在離出發時間 30 分鐘時仍沒有成功匹配到足夠的符合條件的配對人數，系統將會依據現有的人數成立訂單。

3. 查看配對狀態

使用者可以在 app 中看到自己發起的配對是否匹配成功。

4. 聊天室

   當有訂單成立時，系統會建立一個專屬於該筆訂單成員的聊天室，提供他們聯絡的空間。

5. 查看目前訂單

   使用者可以在訂單成立後，查看目前未結束的訂單，以及該訂單的詳細內容。

6. 查看歷史訂單

   使用者可以在該筆訂單已經完成後，查看過去成立過的訂單及詳細內容。

7. 通知功能

   當系統完成某件事，且必須通知使用者時，系統將會新增通知內容在 app 中，例如:配對成功、配對失敗等。

8. 尋找車輛

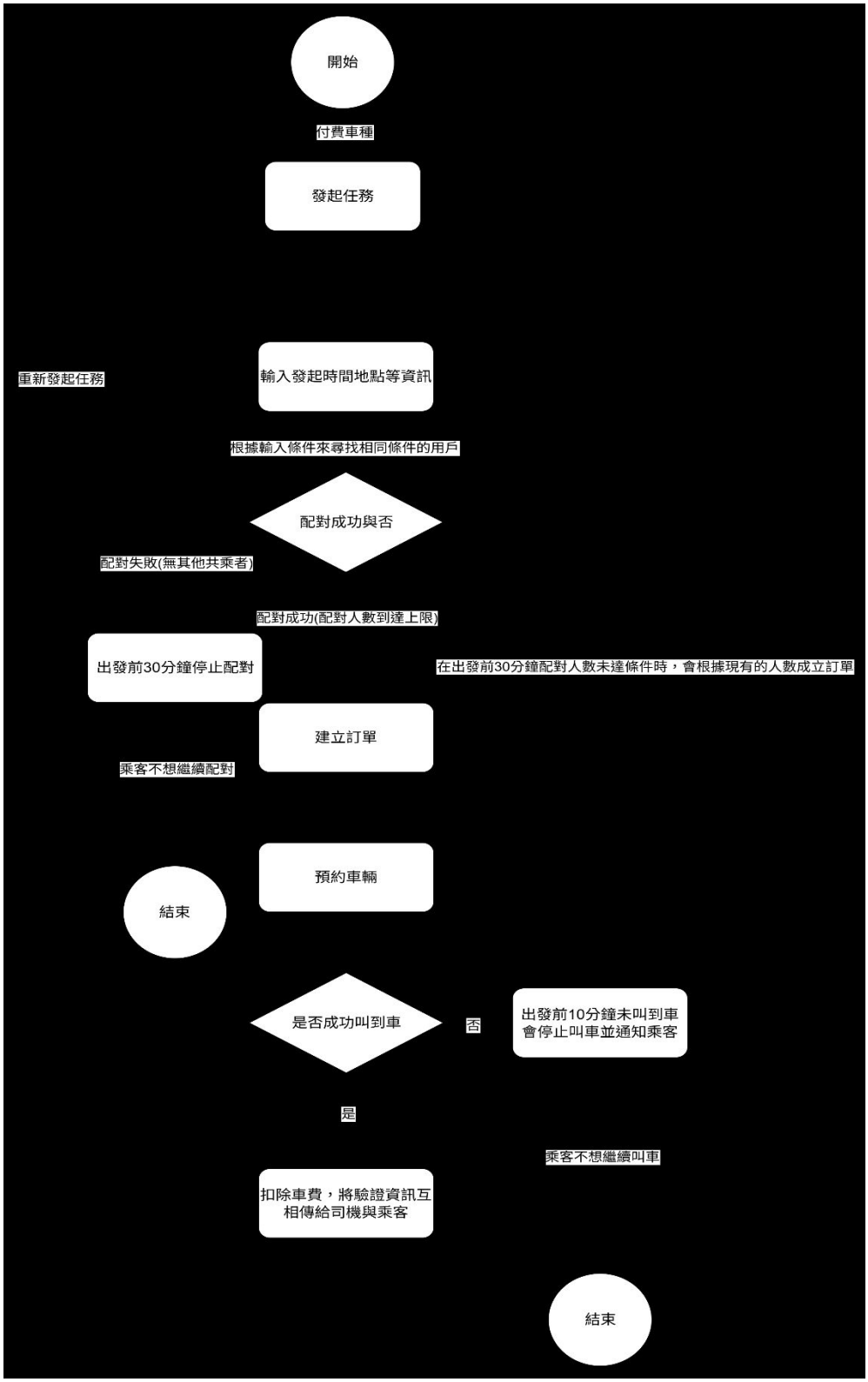
   當有一筆訂單成立後，系統會持續尋找是否有車輛，直到出發時間前 10 分鐘，若是找不到，則取消該筆訂單。

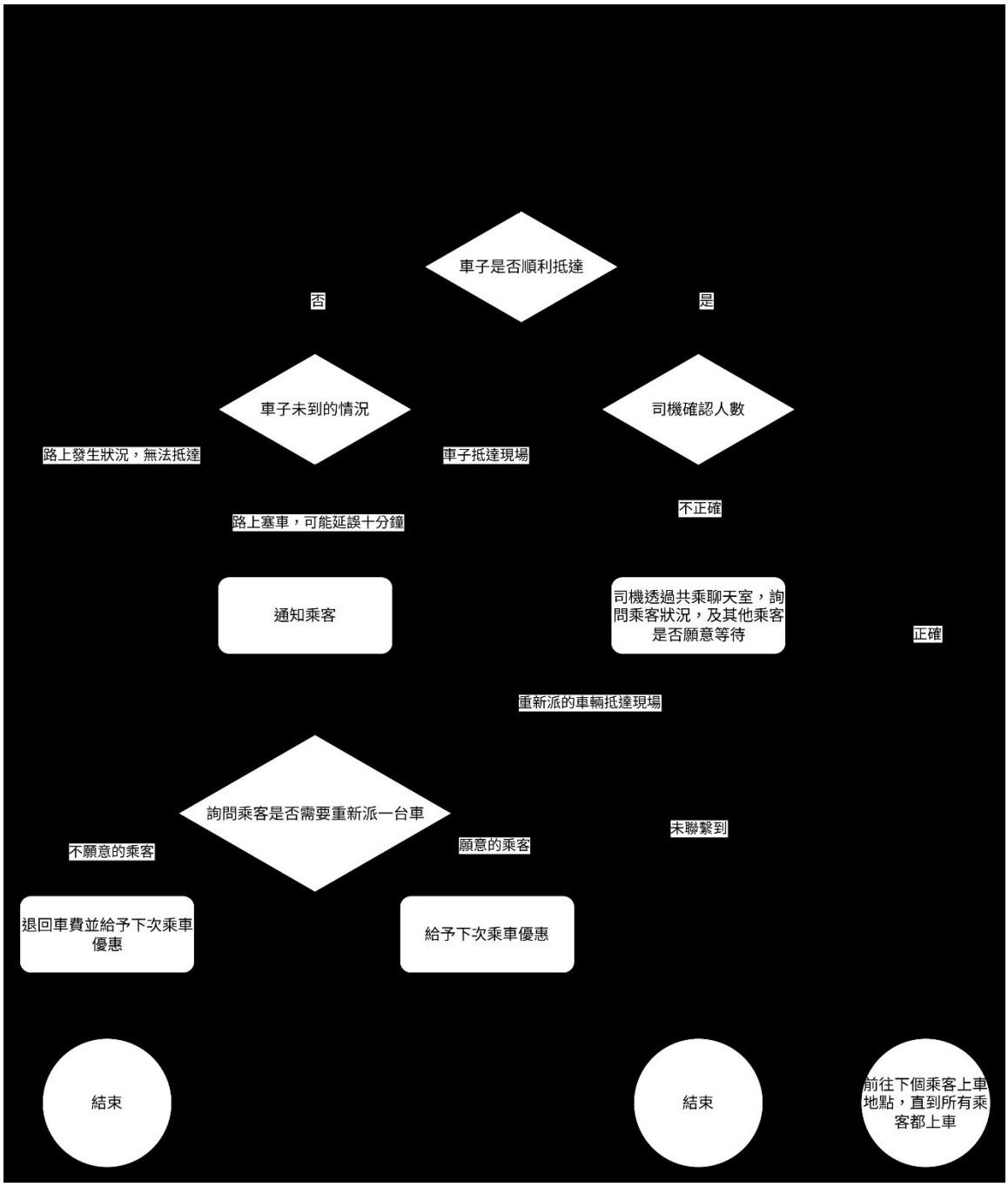
9. 扣款機制

   當系統叫到車後，會自動扣除該筆訂單所需的金額。

10. 乘客驗證機制

系統會在扣款完畢後，發送每位乘客一組驗證碼做為驗證資訊給司機，上車前司機將會逐一確認每位乘客的驗證碼。


開始
付費車種
發起任務
輸入發起時間地點等資訊
重新發起任務
根據輸入條件來尋找相同條件的用戶
配對成功與否
配對失敗(無其他共乘者)
配對成功(配對人數到達上限)
出發前30分鐘停止配對
在出發前30分鐘配對人數未達條件時，會根據現有的人數成立訂單
建立訂單
乘客不想繼續配對
預約車輛
結束
是否成功叫到車
否
出發前10分鐘未叫到車
會停止叫車並通知乘客
是
乘客不想繼續叫車
扣除車費，將驗證資訊互
相傳給司機與乘客
結束



(圖四)付費共乘流程圖

流程圖說明

使用者發起任務並輸入乘車的資訊(B1、B2)，選擇 4 或 7 人當湊滿人數時就會直接建立訂單，如果在出發前 30 分鐘配對人數未達條件時，會根據現有的人數成立訂單並且預約車輛(B3、B5、B6)，反之在出發時間前 30 分鐘未配對到其他共乘者就會停止配對(B4)。訂單成立後若在出發前 10 分鐘未叫到車就會停止叫車並通知乘客(B9)，反之叫到車就會扣除車費並將驗證資訊傳給司機與顧客雙方(B8)。車子若順利抵達，司機會確認人數，正確就會前往下個乘客上車地點，直到所有乘客上車即可出發各自目的地(B9->B9.a)，若乘客未到達會聯繫該乘客的情況。如果車子遇上情況時，像是路上塞車可能延誤，就會通知乘客(B11.a)。但是發生的情況是車禍或路上拋錨等問題導致無法抵達現場時，這時會通知乘客是否需要重新發派一台車，願意的乘客會給予下次乘車優惠並車資依舊依照共乘人數平分的車資計算，不願意的乘客則將返回這次的車資並且也給予下次乘車優惠(B11.b、B12、B13)。

四、系統實作:

本系統以 React Native 為前端跨平台開發框架，後端採用 FastAPI 搭配 MySQL 資料庫做為系統架構，並分為非付費共乘及付費共乘兩大功能

(一) 資料庫關聯綱目設計

資料表欄位說明

1. Member:存放所有會員資訊。

   ˙帳號、密碼、姓名、性別、email、電話、出生年月日

   ˙是否擁有開發者權限

2. Driver:司機的資料。

   ˙駕照號碼、電話、性別、司機類型（非付費車種/付費車種）

3. PaidCarList:平台登記的付費車輛清單。

   ˙車種、公司、車牌、狀態

4. RideSharingInformation: 會員發起的共乘需求

   ˙發起會員

   ˙配對開始時間

   ˙搭乘日期、時間

   ˙起訖地點

   ˙搭乘人數

   ˙狀態（Pending / Matched）

   ˙是否有車

5. MatchOutcome: 紀錄已成立的共乘訂單

   ˙訂單/配對 ID

- 對應會員需求
- 對應司機、車輛
- 最終出發時間、到達時間
- 最終起訖地點
- 訂單狀態（待確認、進行中、完成）
- 乘客實際人數
- 訂單金額

6. Payment: 付款資訊。
   - 付款方式
   - 付款金額
7. FreeRideMember Bridge table。
   - 參加會員
   - 所屬免費行程
   - 加入時間
8. FreeRideScheduleList:免費共乘行程
   - 司機、車牌號碼、車名、車輛狀態
   - 出發時間、抵達時間、起點、終點
   - 最大可容納人數、行程狀態

9. ChatRoom: 聊天室，依據訂單建立。

   - 聊天室類型（付費 / 免費）

   - 參與人數

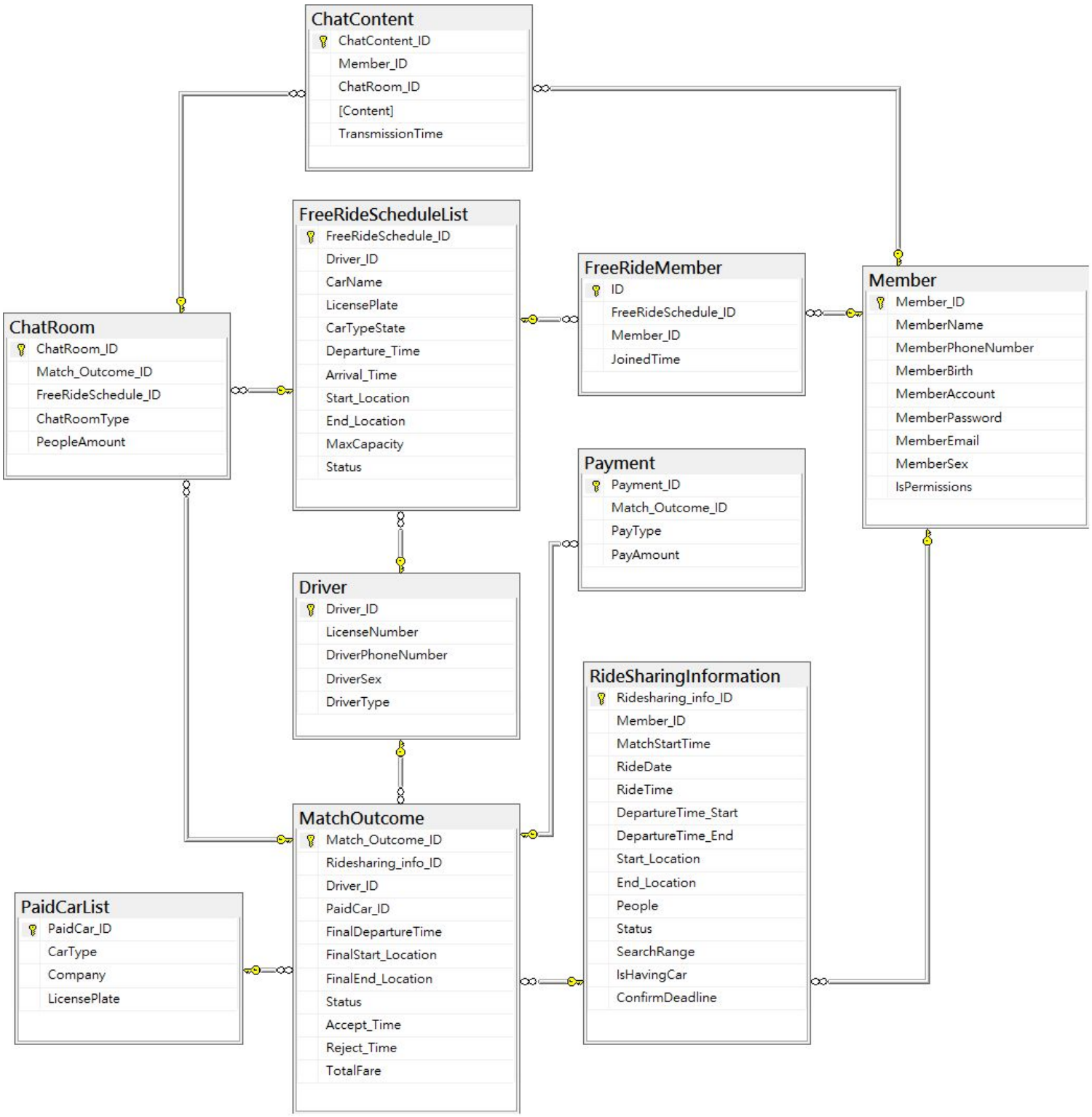
10. ChatContent:聊天室訊息紀錄。

   - 所屬聊天室

   - 發送會員

   - 訊息內容

   - 傳送時間


ChatContent
ChatContent_ID
Member_ID
ChatRoom_ID
[Content]
TransmissionTime
FreeRideScheduleList
FreeRideSchedule_ID
Driver_ID
CarName
LicensePlate
CarTypeState
Departure_Time
Arrival_Time
Start_Location
End_Location
MaxCapacity
Status
FreeRideMember
ID
FreeRideSchedule_ID
Member_ID
JoinedTime
Member
Member_ID
MemberName
MemberPhoneNumber
MemberBirth
MemberAccount
MemberPassword
MemberEmail
MemberSex
IsPermissions
ChatRoom
ChatRoom_ID
Match_Outcome_ID
FreeRideSchedule_ID
ChatRoomType
PeopleAmount
Payment
Payment_ID
Match_Outcome_ID
PayType
PayAmount
Driver
Driver_ID
LicenseNumber
DriverPhoneNumber
DriverSex
DriverType
RideSharingInformation
Ridesharing_info_ID
Member_ID
MatchStartTime
RideDate
RideTime
DepartureTime_Start
DepartureTime_End
Start_Location
End_Location
People
Status
SearchRange
IsHavingCar
ConfirmDeadline
MatchOutcome
Match_Outcome_ID
Ridesharing_info_ID
Driver_ID
PaidCar_ID
FinalDepartureTime
FinalStart_Location
FinalEnd_Location
Status
Accept_Time
Reject_Time
TotalFare
PaidCarList
PaidCar_ID
CarType
Company
LicensePlate

圖五(資料庫實體關聯圖)

(二) 環境設定

1.前端與後端連線

後端啟動方式為 uvicorn main:app --host 0.0.0.0 --port 8000

```
PS C:\Users\g0905\OneDrive\Desktop\python-fastapi> uvicorn main:app --host 0.0.0.0 --port 8000
```

--host 0.0.0.0:表示任何區域網路內的 IP 都能來存取這服務

--port 8000:表示服務開在埠口為 8000

前端在虛擬機實作時，使用 10.0.2.2 去抓取後端的 API

前端在實體機實作時，使用該電腦的區域網路 IP

將前端改成 http://ip 的位置:8000 即可在虛擬機與實體機切換

```
export const API_BASE_URL = "http://10.0.2.2:8000";
```

2.資料庫與後端連線

(1)連線設定

後端使用 mysql.connector 套件來與 MySQL 資料庫連線，並在程式中寫一個 get_db() 函式

```
def get_db():
    return mysql.connector.connect(
        host="localhost",
        user="root",
```

password="xxxxxx",

database="carpool"

)

說明

host → 資料庫伺服器位置

user → MySQL 使用者帳號

password → MySQL 使用者密碼

database → 指定要連線的資料庫名稱

(2)連線方式

建立連線

每次後端需要存取資料庫（例如查詢會員、建立訂單），就會呼叫 get_db() 建立連線物件。

操作資料庫

透過 cursor =db.cursor() 來執行 SQL 語法

```
db = get_db()
cursor = db.cursor(dictionary=True)
cursor.execute("SELECT * FROM member WHERE MemberAccount = %s", (data.MemberAccount,))
```

3.Google map 顯示

設定 Google Maps API 的金鑰，並建立一個 gmaps 客戶端物件。

```
GMAP_KEY = os.getenv("GOOGLE_MAPS_API_KEY", "AIzaSyDO-V7HOak_Ueu3NM-IdabJJ1X-pvHDH2U")
gmaps = googlemaps.Client(key=GMAP_KEY)
```

googlemaps.Client() 使後續能使用 Geocoding、Distance Matrix 等 API。

Geocoding

```
new_start_lat, new_start_lng = geocode_cached(req.start_location)
new_end_lat, new_end_lng = geocode_cached(req.end_location)
```

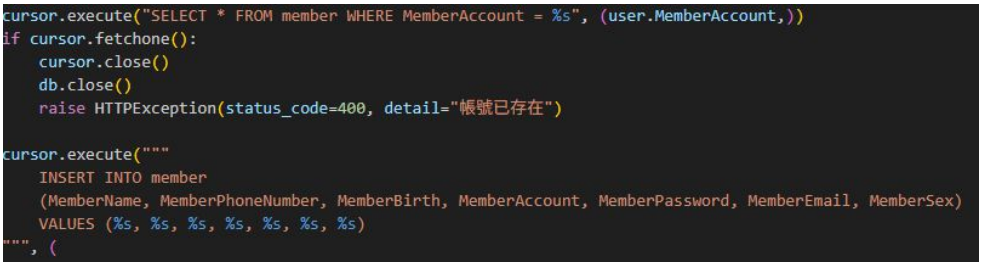
Distance Matrix

```
pick_dist = route_distance_m(new_start_lat, new_start_lng, r_start_lat, r_start_lng, mode="driving")
drop_dist = route_distance_m(new_end_lat,   new_end_lng,   r_end_lat,   r_end_lng,   mode="driving")
```

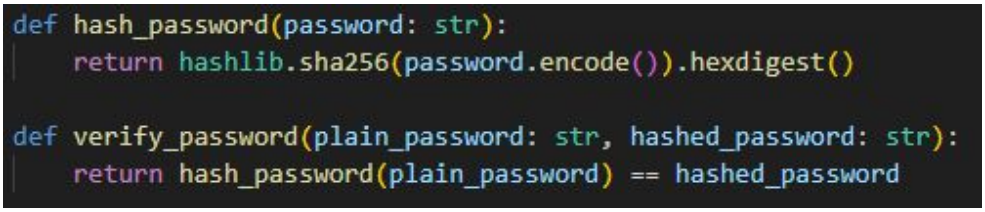
(三)App 功能實作

1. 註冊及登入

在註冊流程中，系統首先接收使用者輸入的基本資料（包含姓名、帳號、密碼、電話、生日、性別與電子郵件），並檢查資料庫中是否已有相同帳號存在；若帳號重複，則回傳錯誤訊息避免重複註冊。若無重複，系統會使用 SHA-256 加密演算法 將密碼加密後寫入資料庫，以確保帳號安全性。



圖六(註冊)



圖七(SHA-256 加密演算法)

登入流程則透過比對帳號與加密後的密碼完成驗證。當使用者輸入帳號密碼時，系統會查詢資料庫中對應的帳號資料，並使用相同的加密方式進行密碼比對。若驗證成功，系統會回傳登入成功訊息及使用者基本資訊（例如會員編號、姓名與電子郵件），作為後續系統操作的身分依據。此設計不僅確保使用者資料的安全，也提升了整體系統的穩定性與可維護性。

圖八(帳密比對)

2. 非付費共乘任務功能

使用者在選擇欲加入的免費共乘任務時，系統會先檢查

FreeRideMember 資料表中是否已有相同的參與紀錄，以避免重複加入同一任務。若未重複，系統便會將使用者的會員編號 (Member_ID) 與任務編號 (FreeRideSchedule_ID) 新增至資料表中，並回傳「加入成功」的訊息，表示參與登記完成。

```
cursor.execute("""
    SELECT 1 FROM FreeRideMember
    WHERE Member_ID = %s AND FreeRideSchedule_ID = %s
""", (data.Member_ID, data.FreeRideSchedule_ID))
if cursor.fetchone():
    cursor.close()
    db.close()
    raise HTTPException(status_code=400, detail="已加入此任務")

cursor.execute("""
    INSERT INTO FreeRideMember (FreeRideSchedule_ID, Member_ID)
    VALUES (%s, %s)
""", (data.FreeRideSchedule_ID, data.Member_ID))
```

圖九(非付費共乘任務加入)

當使用者選擇退出任務時，系統會透過 DELETE 指令刪除該會員在該任務中的對應紀錄，並確認是否有實際刪除資料。若查無紀錄則回傳錯誤訊息，提示使用者該任務並未參與。此設計確保資料一致性與操作安全性，使使用者能即時加入或退出任務，提升整體系統的互動性與使用體驗。

```
cursor.execute("""
    DELETE FROM FreeRideMember
    WHERE FreeRideSchedule_ID = %s AND Member_ID = %s
""", (data.FreeRideSchedule_ID, data.Member_ID))
```

圖十(非付費共乘任務退出)

本系統設計了自動化排程機制，用以定期清除已出發 30 分鐘之免費共乘任務，以維持資料庫的整潔與運作效率。此功能透過 APScheduler 實作，建立名為 delete_expired_rides() 的函式，負責刪除 FreeRideScheduleList 資料表中出發時間已超過 30 分鐘的任務紀錄，並加入到歷史訂單中。當排程執行時，系統會自動執行 DELETE 指令清除過期資料，並輸出刪除筆數以供監控。

```
def delete_expired_rides():
    """定期刪除已過期的免費任務（出發超過 30 分鐘）"""
    db = get_db()
    cursor = db.cursor()
    cursor.execute("""
        DELETE FROM FreeRideScheduleList
        WHERE Departure_Time < NOW() - INTERVAL 30 MINUTE
    """)
    db.commit()
    print(f"🧹 已清除 {cursor.rowcount} 筆過期免費任務")
    cursor.close()
    db.close()
```

圖十一(自動刪除過期任務)

排程器設定為每分鐘執行一次，確保資料能即時更新、避免累積無效任務。此設計不僅降低資料冗餘與查詢負擔，也確保使用者在前端查詢時僅能看到仍有效的任務資訊，提升系統穩定度與資料一致性。

```
scheduler.add_job(delete_expired_rides, 'interval', minutes=1)
```

圖十二(排程器)

3. 共乘匹配

本系統的核心功能之一為「共乘配對機制」，其主要目標是根據乘客輸入的出發地、目的地與時間，自動比對符合條件的乘客，以達成共乘配對。此功能使用 Google Maps API 的 Geocoding 與 Distance Matrix 服務，透過實際「路徑距離（Driving Distance）」的方式，進行精確的配對判斷。

```
# ---- Geocode 新單 ----
try:
    new_start_lat, new_start_lng = geocode_cached(req.start_location)
    new_end_lat, new_end_lng = geocode_cached(req.end_location)
except Exception as ge:
    raise HTTPException(status_code=400, detail=f"地點解析失敗: {ge}")
```

圖十三(Google Map API Geocoding 服務)

在配對流程中，系統首先接收使用者的請求（包含會員編號、日期、時間、起訖地點及人數），並呼叫 Google Maps API 取得對應的經緯度座標。接著，將新訂單以 pending 狀態寫入 RideSharingInformation 資料表，並自動建立一則「配對請求已送出」的通知。

```
cursor.execute("""
    INSERT INTO RideSharingInformation (
        Member_ID, MatchStartTime, RideDate, RideTime,
        Start_Location, End_Location, People, Status
    ) VALUES (%s, %s, %s, %s, %s, %s, %s, 'pending')
""", (
    req.member_id, now, req.date, req.time,
    req.start_location, req.end_location, req.people
))
```

圖十四(插入一筆新的配對資訊)

系統隨後搜尋同日期、同人數、尚未配對完成的待配對訂單，逐筆比對是否滿足以下條件：

(1) 時間窗限制：出發時間需在 ±15 分鐘內；

```
r_dt = parse_db_time_to_dt(str(r['RideDate']), r['RideTime'])
if abs((new_ride_time - r_dt).total_seconds()) > TIME_WINDOW_S:
    continue
```

```
TIME_WINDOW_S = 15 * 60
```

圖十五(時間窗限制)

(2) 路徑距離門檻：兩筆不同配對資訊的起點之間與終點之間的路徑距離皆需小於等於 1 公里。

```
# 3-3) 「路徑距離」群聚
pick_dist = route_distance_m(new_start_lat, new_start_lng, r_start_lat, r_start_lng, mode="driving")
drop_dist = route_distance_m(new_end_lat,   new_end_lng,   r_end_lat,   r_end_lng,   mode="driving")
pass_radius_rule = (pick_dist <= PICK_RADIUS_M and drop_dist <= DROP_RADIUS_M)
```

```
PICK_RADIUS_M = 1000
DROP_RADIUS_M = 1000
```

圖十六(路徑距離門檻)

(3) 樞紐比對機制：若兩筆訂單的終點皆為車站、轉運站、捷運站等樞紐類型地點，且距離小於 1 公里，也視為可配對。

```
pass_hub_rule = False
if new_hub_center is not None:
    hub_dist_r = route_distance_m(r_end_lat, r_end_lng, new_hub_center[0], new_hub_center[1], mode="driving")
    if hub_dist_r <= HUB_RADIUS_M:
        pass_hub_rule = True
if not pass_hub_rule and looks_like_hub(r['End_Location']):
    hub_dist_new2 = route_distance_m(new_end_lat, new_end_lng, r_end_lat, r_end_lng, mode="driving")
    if hub_dist_new2 <= HUB_RADIUS_M:
        pass_hub_rule = True

if pass_radius_rule or pass_hub_rule:
    same_route_ids.append(r['Ridesharing_info_ID'])
```

```
HUB_HINT_WORDS = ["車站", "捷運", "轉運站", "火車站", "高鐵", "機場"]
HUB_RADIUS_M   = 1000
```

圖十七(樞紐比對機制)

若配對成功且達到設定人數，系統會自動將所有符合條件的訂單狀態更新為 matched，並將配對結果寫入 MatchOutcome 資料表，同時向所有成團乘客發送「配對成功」通知；若未達人數，則維持 pending 狀態並等待其他乘客加入。

```
if len(same_route_ids) >= int(req.people):
    placeholders = ','.join(['%s'] * len(same_route_ids))
    cursor.execute(f"""
        UPDATE RideSharingInformation
        SET Status = 'matched'
        WHERE Ridesharing_info_ID IN ({placeholders})
    """, same_route_ids)

    for ride_id in same_route_ids:
        cursor.execute("""
            INSERT INTO MatchOutcome (
                Ridesharing_info_ID, PaidOrder_ID, FinalDepartureTime,
                FinalStart_Location, FinalEnd_Location, Status
            ) VALUES (%s, NULL, %s, %s, %s, 'matched')
        """, (ride_id, now, req.start_location, req.end_location))
```

圖十八(更新配對狀態及寫入配對結果)

本系統設計了自動化排程功能 force_match_rides()，用於定期檢查即將出發的共乘訂單，確保配對流程順暢並減少人工等待。此函式會定期執行，搜尋資料庫中所有狀態為 pending（待配對）且出發時間在 30 分鐘內 的共乘請求。當系統偵測到多筆符合相同條件的訂單（包含出發地、目的地、日期與乘車人數皆相同）時，會自動將這些訂單的狀態更新為 matched，表示系統已強制成團，無須等待其他使用者加入。

```
cursor.execute("""
    SELECT * FROM RideSharingInformation
    WHERE Status = 'pending'
      AND TIMESTAMPDIFF(MINUTE, NOW(), CONCAT(RideDate, ' ', RideTime)) <= 30
""")
rides = cursor.fetchall()

for ride in rides:
    cursor.execute("""
        UPDATE RideSharingInformation
        SET Status = 'matched'
        WHERE Start_Location = %s AND End_Location = %s
          AND RideDate = %s AND People = %s AND Status = 'pending'
    """, (ride['Start_Location'], ride['End_Location'], ride['RideDate'], ride['People']))
```

```
scheduler.add_job(force_match_rides, 'interval', minutes=1)
```

圖十九(排程器實作自動化配對人數未達標自動成單)

此機制能有效避免因時間逼近而導致行程無法成團的情況，同時提升使用者的搭乘體驗與系統運作效率。系統在執行後會輸出

成團筆數紀錄，以便日後監控與除錯。透過此自動化邏輯，平台能兼顧即時性與穩定性，確保共乘任務能在出發前即時完成配對。

4. 配對狀態查看

在本系統中，使用者可透過 查詢配對狀態 API 了解目前共乘任務的配對進度。此功能以 FastAPI 實作，透過 SQL 查詢 RideSharingInformation 資料表，篩選出 Status = 'pending'並且屬於該會員的未出發行程。系統會依據 MatchStartTime 進行排序，讓使用者能優先查看最新的配對紀錄。

```
cursor.execute("""
    SELECT Start_Location, End_Location, People, Status, RideDate, RideTime, MatchStartTime
    FROM RideSharingInformation
    WHERE Member_ID = %s
      AND Status = 'pending'
      AND CONCAT(RideDate, ' ', RideTime) >= NOW()
    ORDER BY MatchStartTime DESC
    LIMIT 20
""", (member_id,))
```

圖二十(查看配對狀態)

查詢結果中包含上車地點、下車地點、乘車人數、配對狀態、日期與時間等資訊。為確保資料呈現一致性，程式中對 MySQL 傳回的時間格式進行修正：RideDate 轉換為「YYYY-MM-DD」格式，而 RideTime 轉換為「HH:MM」格式，使前端在顯示時能

保持統一格式與易讀性。此設計不僅提升使用者查詢便利性，也確保資料在資料庫與前端之間的時間格式同步與正確性。

```
for row in rows:
    # RideTime -> "HH:MM"
    if hasattr(row["RideTime"], "seconds"):  # MySQL TIME -> timedelta
        seconds = row["RideTime"].seconds
        hours = seconds // 3600
        minutes = (seconds % 3600) // 60
        row["RideTime"] = f"{hours:02d}:{minutes:02d}"
    else:
        row["RideTime"] = str(row["RideTime"])
    # RideDate -> "YYYY-MM-DD"
    if hasattr(row["RideDate"], "strftime"):
        row["RideDate"] = row["RideDate"].strftime("%Y-%m-%d")
    else:
        row["RideDate"] = str(row["RideDate"])
```

圖二十一(時間格式修正)

5. 目前訂單

本系統設計了「目前訂單顯示功能」，使使用者能即時查詢已成功配對的付費共乘行程。透過查詢 RideSharingInformation 資料表取得使用者的當前訂單資料。系統僅篩選狀態為 matched 且尚未出發的訂單，依照出發日期與時間排序後回傳，讓使用者能清楚了解近期的乘車安排。

```
cursor.execute("""
    SELECT Ridesharing_info_ID, Start_Location, End_Location, RideDate, RideTime
    FROM RideSharingInformation
    WHERE Member_ID = %s
      AND Status = 'matched'
      AND CONCAT(RideDate, ' ', RideTime) >= NOW()
    ORDER BY RideDate ASC, RideTime ASC
""", (member_id,))
```

圖二十二(目前訂單顯示)

此外，系統提供「目前訂單查看功能」，使用者可進一步點選查看該筆訂單的共乘名單。此功能透過查詢 MatchOutcome 資料表取得配對群組的共乘資訊（包含出發時間、起訖地點），再根據該群組條件查詢對應的所有乘客資料，包含姓名與電子郵件。若該筆訂單尚未配對成功，系統會回傳提示訊息。此設計使使用者能清楚掌握共乘對象與行程細節，提升透明度與使用便利性。

```
cursor.execute("""
    SELECT FinalDepartureTime, FinalStart_Location, FinalEnd_Location
    FROM MatchOutcome
    WHERE Ridesharing_info_ID = %s
""", (ridesharing_info_id,))
group_info = cursor.fetchone()
```

圖二十三(目前訂單查看功能)

```
cursor.execute("""
    SELECT m.Member_ID, m.MemberName, m.MemberEmail
    FROM MatchOutcome mo
    JOIN RideSharingInformation r ON mo.Ridesharing_info_ID = r.Ridesharing_info_ID
    JOIN Member m ON r.Member_ID = m.Member_ID
    WHERE mo.FinalDepartureTime = %s
      AND mo.FinalStart_Location = %s
      AND mo.FinalEnd_Location = %s
""", (
    group_info["FinalDepartureTime"],
    group_info["FinalStart_Location"],
    group_info["FinalEnd_Location"]
))
```

圖二十四(目前訂單顯示共乘人)

6. 歷史訂單

在「歷史訂單（免費共乘）」部分，系統會根據使用者的會員

編號查詢 FreeRideScheduleList 資料表，僅顯示出發時間已經過去的任務。查詢結果包含任務主鍵、起訖地點、出發時間、車輛資訊及當次共乘人數。使用者亦可透過「歷史訂單詳情（免費）」進一步查看單筆任務的詳細資料，如出發與抵達時間、車牌號碼、最大乘客數及任務狀態。

```
cursor.execute("""
    SELECT
        s.FreeRideSchedule_ID,                -- 主鍵
        s.Start_Location,
        s.End_Location,
        s.Departure_Time,
        s.CarName,
        s.LicensePlate,
        (SELECT COUNT(*) FROM FreeRideMember fm
         WHERE fm.FreeRideSchedule_ID = s.FreeRideSchedule_ID) AS PassengerCount
    FROM FreeRideScheduleList s
    JOIN FreeRideMember m ON s.FreeRideSchedule_ID = m.FreeRideSchedule_ID
    WHERE m.Member_ID = %s AND s.Departure_Time < NOW()
    ORDER BY s.Departure_Time DESC
""", (member_id,))
```

圖二十五(免費共乘歷史訂單顯示)

```
cursor.execute("""
    SELECT
        s.FreeRideSchedule_ID,
        s.Start_Location,
        s.End_Location,
        s.Departure_Time,
        s.Arrival_Time,
        s.CarName,
        s.LicensePlate,
        s.MaxCapacity,
        s.Status,
        (SELECT COUNT(*) FROM FreeRideMember fm
         WHERE fm.FreeRideSchedule_ID = s.FreeRideSchedule_ID) AS PassengerCount
    FROM FreeRideScheduleList s
    WHERE s.FreeRideSchedule_ID = %s
""", (schedule_id,))
```

圖二十六(免費共乘歷史訂單詳情)

在「歷史訂單（付費共乘）」部分，系統會從

RideSharingInformation 資料表中篩選出狀態為 matched 且出發時間早於目前時間的紀錄。結果顯示包括訂單主鍵、起訖地點、乘車日期、時間與乘客人數。若使用者查詢「歷史訂單詳情（付費）」，系統則會同時結合 MatchOutcome 資料表，取得該筆訂單的最終出發時間與群組資訊，並計算同團乘客總數。

```
cursor.execute("""
    SELECT
        Ridesharing_info_ID,                    -- 主鍵
        Start_Location,
        End_Location,
        RideDate,
        RideTime,
        People
    FROM RideSharingInformation
    WHERE Member_ID = %s
      AND Status = 'matched'
      AND CONCAT(RideDate, ' ', RideTime) < NOW()
    ORDER BY RideDate DESC, RideTime DESC
""", (member_id,))
```

圖二十七(付費共乘歷史訂單顯示)

```
cursor.execute("""
    SELECT
        s.FreeRideSchedule_ID,
        s.Start_Location,
        s.End_Location,
        s.Departure_Time,
        s.Arrival_Time,
        s.CarName,
        s.LicensePlate,
        s.MaxCapacity,
        s.Status,
        (SELECT COUNT(*) FROM FreeRideMember fm
         WHERE fm.FreeRideSchedule_ID = s.FreeRideSchedule_ID) AS PassengerCount
    FROM FreeRideScheduleList s
    WHERE s.FreeRideSchedule_ID = %s
""", (schedule_id,))
```

圖二十八(付費共乘歷史訂單詳情)

7. 通知功能

在「新增通知」部分，系統會於觸發特定事件（如配對成功、任務變更或乘客退出）時，自動呼叫 create_notification() 函式，將通知內容、建立時間及對應會員編號寫入 Notification 資料表中，並將初始狀態設為未讀 (IsRead = 0)。

```
def create_notification(data: NotificationCreate):
    db = get_db()
    cursor = db.cursor()
    cursor.execute("""
        INSERT INTO Notification (Member_ID, Message, CreatedTime, IsRead)
        VALUES (%s, %s, NOW(), 0)
    """, (data.member_id, data.message))
```

圖二十九(新增通知)

在「通知顯示」功能中，使用者可透過 API 取得所有通知且僅篩選未讀通知。系統會依照建立時間與通知編號進行排序，確保最新通知優先顯示。

```
if unread:
    # 只抓未讀
    cursor.execute("""
        SELECT * FROM Notification
        WHERE Member_ID = %s AND IsRead = 0
        ORDER BY CreatedTime DESC, Notification_ID DESC
    """, (member_id,))
else:
    # 全部通知，單純依時間排序
    cursor.execute("""
        SELECT * FROM Notification
        WHERE Member_ID = %s
        ORDER BY CreatedTime DESC, Notification_ID DESC
    """, (member_id,))
```

圖三十(通知顯示)

最後，「設為已讀」功能提供使用者在閱讀通知後更新狀態的能力。系統透過更新指令將 IsRead 欄位設為 1，並檢查是否成功更新資料，以確保操作正確性與資料一致性。

```
cursor.execute("""
    UPDATE Notification SET IsRead = 1 WHERE Notification_ID = %s
""", (notification_id,))
```

圖三十一(已讀通知)

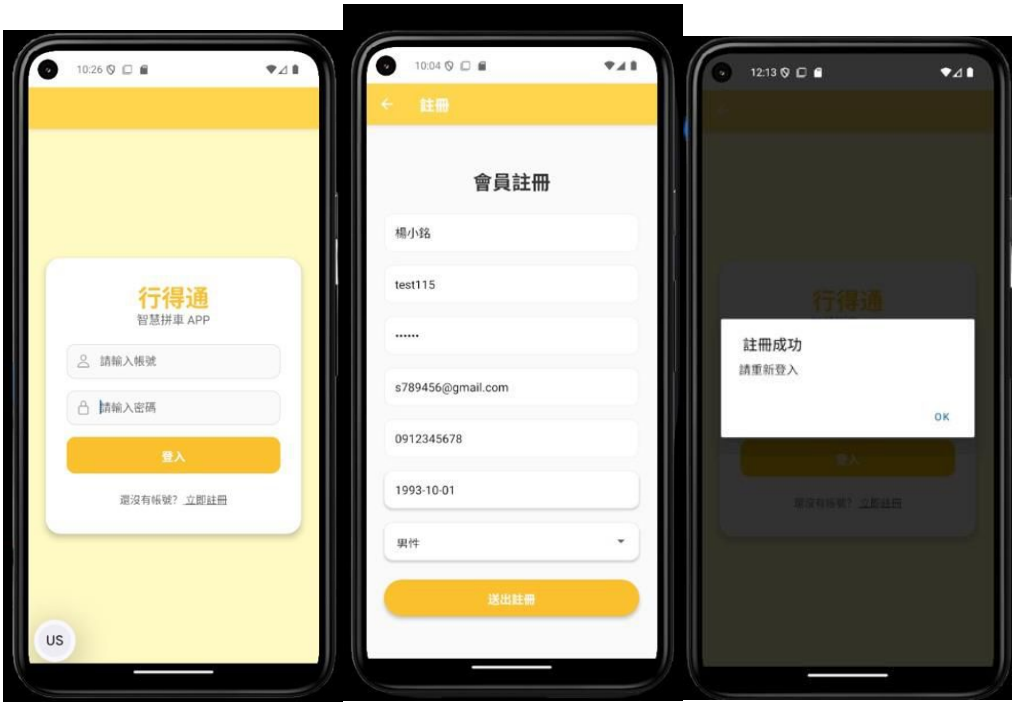
Chapter 4 實驗結果與系統展示

4.1 系統展示

1.登入及註冊

本系統的首頁為登入頁面，主要提供使用者登入與註冊的入口。

畫面中央設有登入區塊，包含兩個輸入欄位：「帳號」與「密碼」。使用者需輸入註冊時設定的帳號與密碼後，點擊下方的「登入」按鈕即可進入主畫面。對於第一次使用本系統的使用者，畫面下方亦提供「立即註冊」的提示，點擊即可跳轉至註冊頁面進行會員註冊。



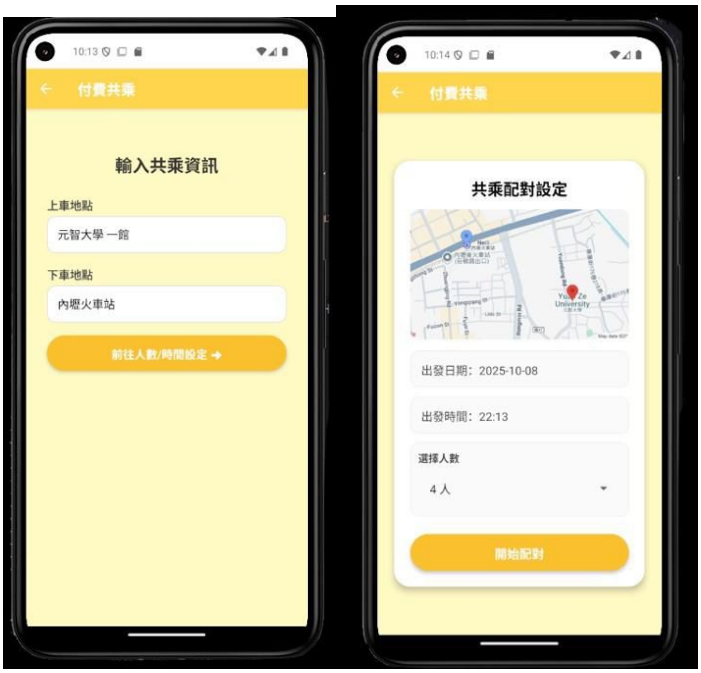
2.功能畫面

登入成功後，使用者可於主畫面中操作以下功能：查看目前訂單、查看歷史訂單、付費共乘、免費共乘、通知與配對狀態等。使用者可透過「查看目前訂單」了解正在進行中的訂單狀況；「查看歷史訂單」則可查詢過去已完成的共乘紀錄。「付費共乘」功能讓使用者輸入出發地點、目的地、出發時間與共乘人數後發起配對任務；「免費共

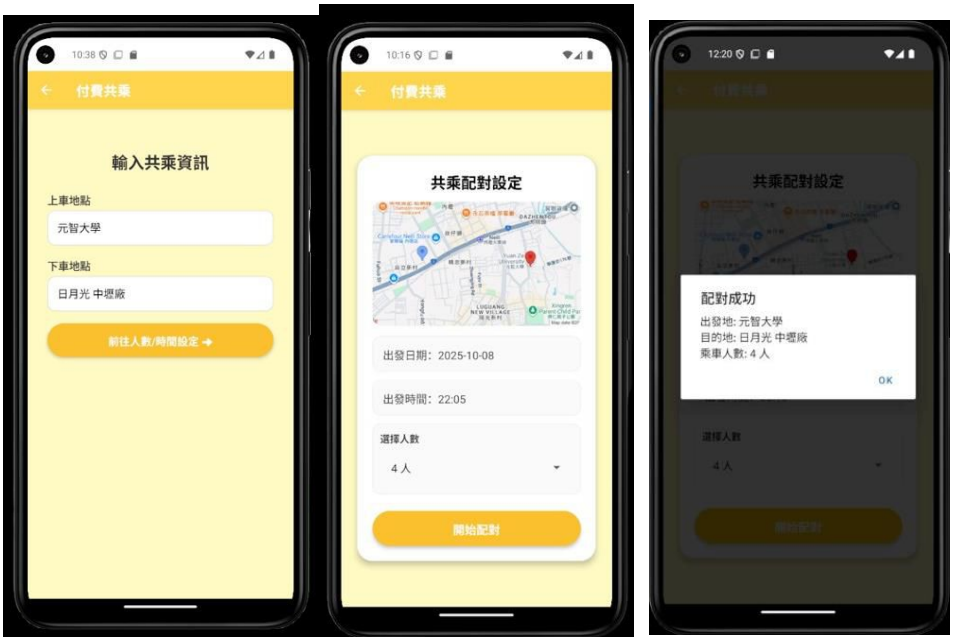
乘」則提供搜尋活動接駁車的服務。「通知」功能會在配對成功、訂單成立或取消時即時推播提醒；而「配對狀態」則可隨時查詢所發起任務的配對進度，確保使用者能即時掌握共乘資訊。

4.付費共乘功能

為了模擬實際的配對情況，我們使用了其他三個帳號預輸入了上下車地點以及出發時間以及乘車人數。



使用者選可以擇乘坐人數為 4 或 7 人，當輸入的上下車地點與其他使用者的路徑距離為 1 公里內以及出發時間相差 15 分鐘內時，系統就會自動配對至一起。



當使用者發起共乘任務後，系統會開始進行自動配對。若此時尚

未成功與其他使用者配對，則可於「配對狀態」頁面中查看目前任務的配對進度。

5.通知功能

當有新的狀態更動時，就會顯示提醒使用者，下圖的通知是上述的配對已成功，因此新增通知到通知列

6.目前訂單

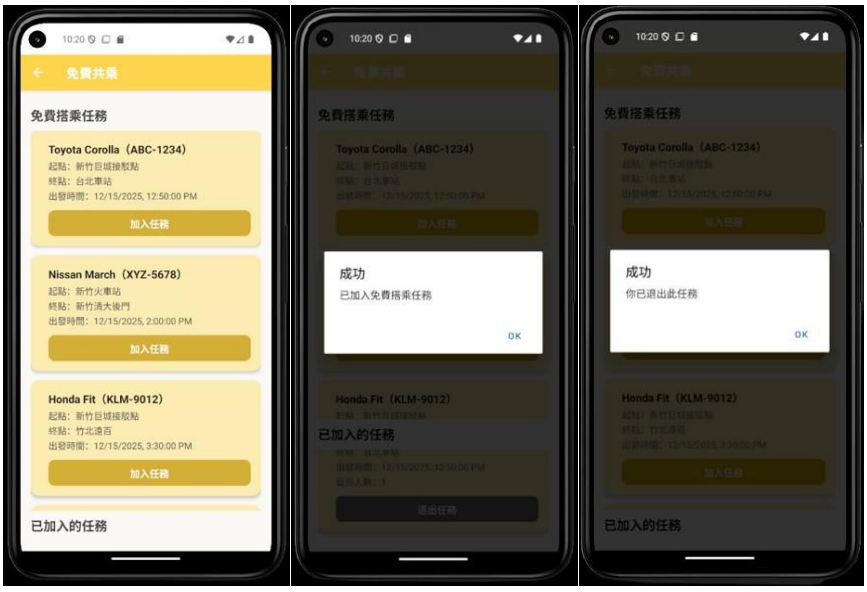
當共乘配對成功後，系統會自動將該筆任務建立為「訂單」，並顯示於「目前訂單」頁面中。此頁面可讓使用者清楚掌握當前尚未結束的共乘任務。

7.付費共乘歷史訂單

當使用者的付費共乘任務完成後，系統會自動將該筆訂單從「目前訂單」移至「歷史訂單」中保存，方便使用者後續查詢過往的搭乘紀錄

8.免費共乘

使用者可以自行加入想加入的任務清單，加入後可以看到該任務的共乘人數，也可以在加入後隨時退出，當到達出發時間時，30 分鐘後會將該任務從清單中轉移至免費共乘歷史訂單



9.免費共乘歷史訂單

免費共乘歷史訂單頁面提供使用者查詢過往的免費共乘紀錄。當使用者參與或發起的免費共乘任務結束後，系統會自動將該筆訂單移入歷史訂單中保存，方便日後查詢。

Chapter 5 結論

5.1 系統回顧

本系統以 FastAPI 為後端框架，React Native 為前端開發工具，整體架構為一套智慧共乘應用程式。在系統開發過程中，主要完成了資料庫的建立、後端 API 的設計，以及前端介面的開發，並實現前後端的連接。系統整合後，使用者能透過行動裝置輸入上車與下車地點，獲得共乘配對結果，達到便利的乘車體驗。

在實作系統之前，先進行了系統流程圖與 ERD 的設計規劃。透過這個步驟，能在開發前明確掌握系統資料流與整體結構，使後續的

實作更具方向性，提升整體開發效率。

在實際開發過程中，仍遇到部分與原始設計不同的情況。例如，在前後端串接及資料表欄位設定時，發現某些需求需要調整。這時，透過先前盡力好的系統架構，我們能依實際狀況做彈性的修改，而不用從頭開始，進而縮短實作的時間。

在開發初期，為了能快速進入系統實作階段，將後端伺服器架設於本地端電腦。此做法在開發與測試階段相當方便，能即時進行除錯與資料驗證。然而，當系統由虛擬機轉換至實體手機進行操作時，發現前後端必須連接於同一個 Wi-Fi 網路下，才能透過相同的區域 IP 進行使用。此設計雖有助於開發測試，但未來若是要考慮實際上架，應將後端伺服器部署於雲端環境。

整體而言，本次系統開發讓我們更加熟悉前後端整合流程，並深刻體會到事前規劃在專案開發中的重要性。透過本次實作，我學習到如何在架構設計與實際開發間取得平衡，為未來系統開發與優化作打算。

5.2 未來發展

1.配對演算法優化

本系統目前的配對方法是直接以新資料跟其他所有舊資料全部比對，在實際的應用上是相當不實際的，因此若是能優化整體配對的效率，例如說使用 BinerySearch、Hashing 等資料結構，使整體的程式碼複雜度下降，或是在配對前先排除差異性較大的資料，對於較為相近的資料再用比較詳細的比對演算法去坐進一步的配對。

2.UI/UX 的流暢度

現階段的系統已具備登入、共乘發起、配對查詢及通知等核心功能，但在使用流程與互動設計上，仍有進一步精進的空間。

首先，在介面設計上，我們將導入更符合行動裝置操作習慣的版面配置，優化按鈕大小、字體可讀性與顏色對比，讓使用者能在不同螢幕尺寸下都有一致且清晰的操作體驗。

在使用者體驗方面，未來將導入動態反饋與互動指引，例如在配對成功、訂單成立時提供即時動畫，增加系統的回饋感。此外，我們也規劃在使用者使用系統功能後蒐集他們的行為數據與回饋問卷，透過數據分析找出操作瓶頸，並以使用者為中心持續改版。

3.Google Map API 實作優化

目前系統在進行共乘配對時，會透過 Google Maps API 計算起

訖點的實際駕車距離，以確保配對的準確性。然而，此設計也帶來了部分效能上的限制，例如每日請求次數上限、API 回應延遲，以及大量同時請求時可能觸發限速問題。因此，未來系統將導入快取層，以儲存重複查詢的地點與路徑，提升 API 的呼叫效率。

此外，為改善多筆請求同時發生時的效能，後端可改採非同步架構，使配對任務能並行處理而不阻塞主線程。這樣的設計能顯著縮短等待時間，提升整體系統的反應速度。另一方面，未來也將增加 錯誤處理與回退機制，在 Google Maps API 回傳失敗或網路延遲時，能自動切換至離線距離估算模式或使用先前的快取資料，以確保服務不中斷。

附錄 A.專題工作內容

組員:陳佳佑

1. 系統非付費共乘設計與構想
2. API 功能導入
3. 後端環境架設
4. APP 功能實作

組員:邱育宏

1. 系統付費功能設計與構想
2. 前端介面設計
3. 資料庫設計
4. APP 功能實作

附錄 B.專題心得與建議

組員:陳佳佑

經過兩個學期的校內專題製作，讓我深刻體會到學習不僅侷限於課堂上的內容，也能根據自己的興趣與方向設定目標，進而主動探索並學習額外的知識。由於本次專題主題與 App 開發相關，使我在學習新技術的過程中，不僅學習了新的技術，也培養了自學與解決問題的能力。

我十分感謝系上提供這樣的學習機會，並特別感謝指導教授楊錦生教授，在整個過程中給予我們許多啟發與建議。教授並不會直接告訴我們應該如何執行，而是引導我們自行思考、找出方向，並在構想階段提供意見回饋，讓我們能夠在後續實作時更有目標與信心。

最後，我要感謝我的組員。專題製作並非一個人的努力，而是團隊合作的成果。起初我並不擅長表達自己的想法，在小組討論時常不知該如何提出意見，但在組員的鼓勵與協助下，我學會了更勇於溝通與分享，這讓我在專題過程中成長許多，也體會到團隊合作的重要性。

組員:邱育宏

當初在構想題目時，我就以自身生活環境為核心，去思考如果我的生活中加了什麼樣的系統，能夠增加便利性，剛好在一次返家的途中，高額的計程車費用以及尖峰時段叫不到車的痛苦，讓我萌生做共乘平台的想

法。

在開發初期，我負責系統整體架構的設計與前端介面規劃。從最一開始的畫面流程圖、ERD 設計，到最後以 React Native 完成實際的 App 介面，每一個步驟都需要與組員不斷討論並進行實機測試。由於系統包含「付費共乘」與「非付費共乘」兩大功能，因此前端畫面必須同時兼顧兩種使用情境。在實作過程中，我學會如何以模組化的方式進行開發，使不同功能能夠在同一套框架下順暢運作。這段經驗讓我學習到如何在初期的系統規劃與架構設計中，影響後續開發的困難度。

除了技術面的挑戰之外，也讓我在踏入職場之前感受團隊合作的重要性。與組員分工時，我們明確劃分了「前端、後端與資料庫」三個主要部分，並透過 Git 進行協作。雖然在初期常因想法不同或是邏輯不一致而花費許多時間協調，但透過磨合彼此實作的個性，逐步地找到了平衡點，也學會了如何透過會議討論、程式註解，保持專案的可讀性。這讓我深刻體會到溝通與紀錄的重要性，也培養了我在團隊中扮演協調與整合角色的能力。

最後，我要感謝我的指導教授在過程中的耐心指導。在每次討論中都讓我們找到我們的想法中還有哪些不足之處。除此之外，也感謝我的組員在開發過程中互相扶持，讓這次的專題能順利完成。