ProjectMgmt 專案管理系統 - 專題報告

1. 專題題目、組名

   題目: ProjectMgmt 任務 / 專案管理系統
   - 建立一個多人協作的專案管理系統，用戶可以在系統內建立多個專案，並加入專案協作者。每個專案的建立者會成為管理員，只有管理員有權限移除專案協作者、刪除專案。專案的協作者可以在專案內新增多個任務列表，每個任務列表下可以新增多個任務事件卡，可以在任務事件卡內新增任務協作者、待辦事項、留言等，並可以管理專案、任務狀態。

   組名: TheBestDbEver

2. 隊長及隊員姓名學號與系級

   隊長:
   - 程至榮 111753151 資科碩一

   隊員:
   - 吳邁龍 111753150 資科碩一
   - 蘇柏鈞 111753133 資科碩一
   - 吳家瑩 111753221 資科碩一
   - 黃滋宥 111753224 資科碩一

3. 每位成員負責之任務分工
   - 程至榮 111753151 資科碩一
     - 需求分析、系統功能、ER Model、Relational Schema、系統架構
     - 前端
     - API 、專案架構
     - 撰寫文件

   - 吳邁龍 111753150 資科碩一
     - 需求分析、系統功能、ER Model、Relational Schema、系統架構
     - 後端 (Task_Card, Task_List)
     - 期末報告心得

   - 蘇柏鈞 111753133 資科碩一
     - 需求分析、系統功能、ER Model、Relational Schema、系統架構
     - 後端 (WorksOn,Todo)

   - 吳家瑩 111753221 資科碩一
     - 需求分析、系統功能、ER Model、Relational Schema、系統架構
     - 後端 (Project)
     - sql 建置

- 黃滋宥 111753224 資科碩一
    - 需求分析、系統功能、ER Model、Relational Schema、系統架構
    - 後端 (User, Comment)

4. 需求分析 (至少 5 個 Entity Types, 5個 Relationship Types)
    - 不同資料表之間需要儲存建立時的 Unix timestamp, 作為主鍵的 ID、以及便於後續利用 ID (創立時間) 查找資料
    - 用戶資料需要包含用戶名字、用戶信箱、用戶頭像 URL
    - 專案資料需要包含專案名字、專案色碼 (讓不同的專案能在前端呈現不同的顏色)
    - 任務列表資料需要包含列表標題、列表狀態
    - 任務卡資料需要包含標題、描述、開始時間、截止時間、任務狀態
    - 待辦事項資料需要包含待辦描述、待辦狀態
    - 留言區資料需要包含留言人名字、留言內容
    - 列表狀態、任務狀態、待辦狀態為: 進行中、已完成兩種
    - 用戶與專案存在 "管理" 的關係, 用戶不一定要擔任管理員, 但每個專案都需要有管理員
    - 用戶與專案存在 "參與" 的關係, 用戶不一定會參與專案, 但專案若存在則一定有用戶參與
    - 用戶與任務卡存在 "參與" 的關係, 用戶不一定會參與任務卡, 但任務卡若存在則一定有用戶參與
    - 專案與任務列表存在 "擁有" 的關係, 專案不一定會有任務列表, 但任務列表若存在則一定存在於專案中
    - 任務列表與任務卡存在 "擁有" 的關係, 任務列表不一定會有任務卡, 但任務卡若存在則一定存在於任務列表中
    - 任務卡與待辦事項存在 "擁有" 的關係, 任務卡不一定有待辦事項, 但是待辦事項若存在則一定存在於任務卡中
    - 任務卡與留言區存在 "擁有" 的關係, 任務卡不一定有留言, 但是留言若存在則一定存在於任務卡中

5. 系統功能 (系統功能必須具備 CRUD)
    a. 建立專案的用戶會成員預設會成員專案管理員, 同時會是專案的協作者
    b. 專案協作者可以加入新的協作者
    c. CRUD
        - Create
            - 建立用戶資料
            - 建立專案
            - 建立任務列表
            - 建立任務卡
            - 建立留言
            - 建立待辦事項
        - Read
            - 進入首頁會讀取與該 User 相關的專案
            - 進入專案會顯示任務列表、任務卡等資訊
            - 點開任務卡可以看到任務標題、描述、開始時間、結束時間、留言區、待辦任務

- Update
    - 更新專案名稱、名字、色碼
    - 更新列表標題、狀態
    - 更新任務卡標題、描述、開始時間、截止時間、任務狀態
    - 更新待辦完成狀態
    - 加入專案、任務協作者
- Delete
    - 刪除專案
    - 刪除專案、任務協作者
    - 刪除任務列表
    - 刪除任務卡
    - 刪除待辦事項
    - 刪除留言

6. ER Model

【注意】, 我們後來的程式實作上在 Comment 新增了 **Comment_ID**, 由 (Commenter_ID, Comment_ID) 共同構成 Comment 的主鍵, 以解決留言者無法重複留言的問題。

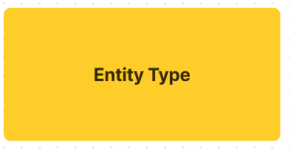
a. Entity Types 及 Relationship Types 設計
    - Entity Types
        - User:
            - User_ID (主鍵)
            - User_Name
            - User_Mail
            - User_Avatar
            - User_Password
        - Project:
            - Project_ID (主鍵)
            - Project_Name
            - Project_Color
        - Task_List:
            - Task_List_ID (主鍵)
            - Task_List_Name
            - Task_List_Status
        - Task_Card
            - Task_Card_ID (主鍵)
            - Task_Card_Name
            - Task_Card_Text
            - Task_Card_StartTime
            - Task_Card_EndTime
            - Task_Card_Status
        - Todo
            - Todo_ID (主鍵)
            - Todo_Text
            - Todo_Status

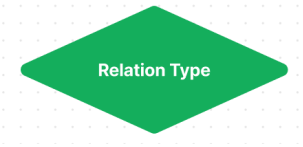
        - Comment
            - Commenter_ID (主鍵)
            - Comment_ID (主鍵)
            - Commenter_Name
            - Comment_Text

- Relationship Types
    - User 和 Project 有 Manage 的關係，Project 為完全參與
    - User 和 Project 有 Works_on 的關係，Project 為完全參與
    - User 和 Task_Card 有 Works_on 的關係，Task_Card 為完全參與
    - Project 和 Task_List 有 Has 的關係，Task_List 為完全參與
    - Task_List 和 Task_Card 有 Has 的關係，Task_Card 為完全參與
    - Task_Card 和 Comment 有 Has 的關係，Comment 為完全參與
    - Task_Card 和 Todo有 Has 的關係，Todo 為完全參與

b. ER Model

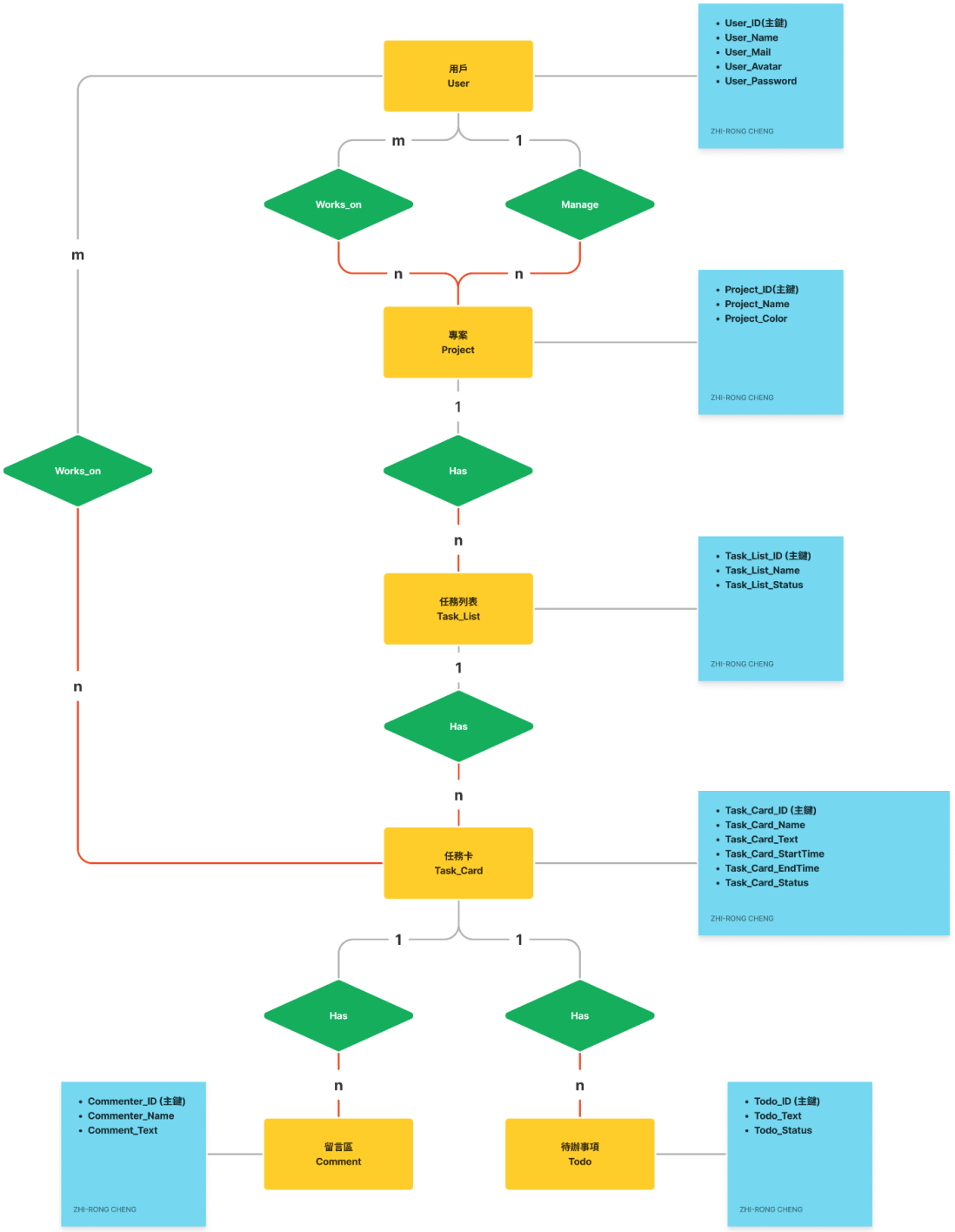
紅色線表示 "完全參與"，而以下的符號各自代表為:

a. 

b. 

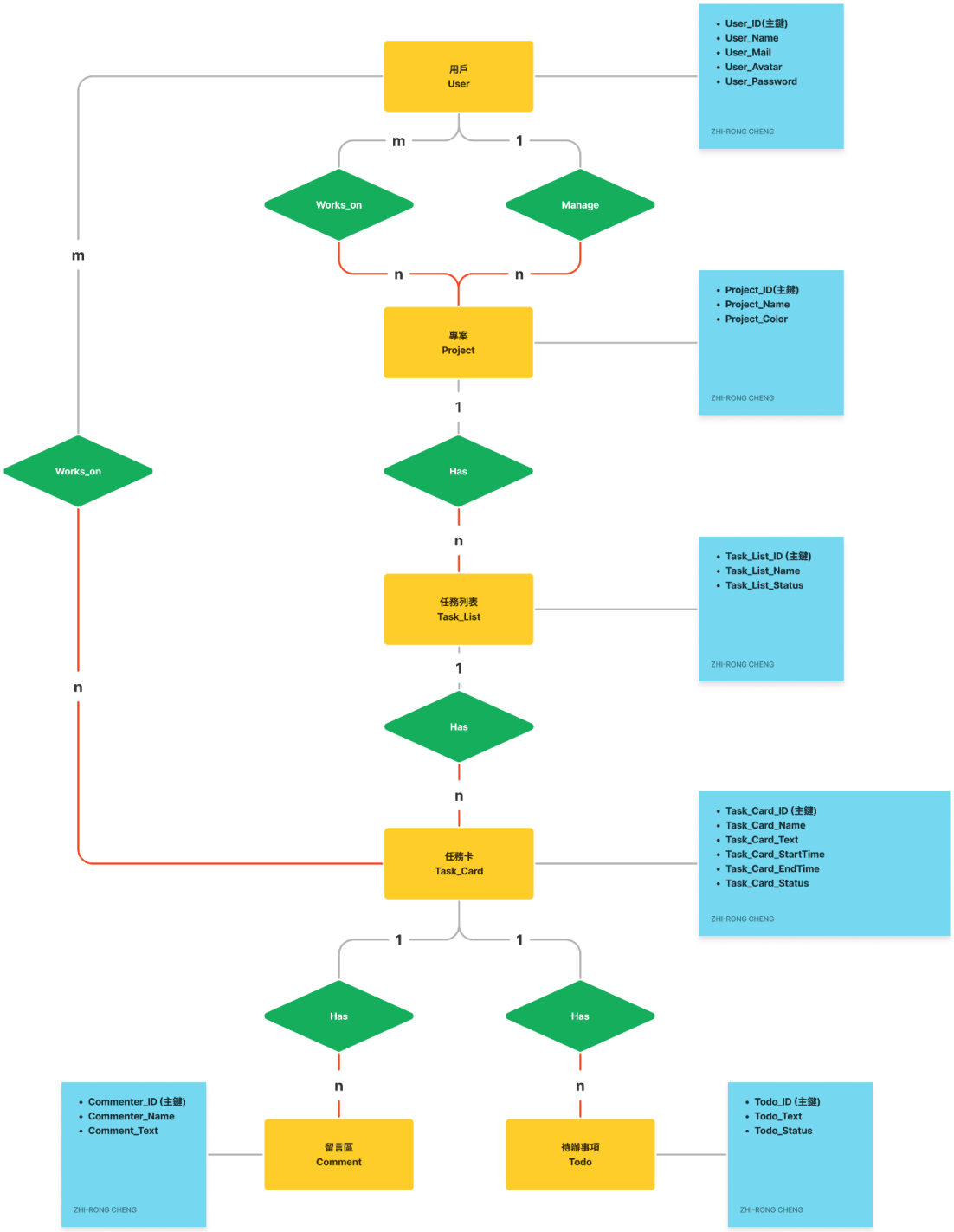
c. 

ER Diagram 如下圖:



7. Relational Schema

a. 根據以下 ER Diagram 進行 ER-Relational Mapping



- Step 1: for each regular entity type E
  - Create a relation R
  - Include all simple attributes of E
  - Include only the simple component attributes of a
  - composite attribute

- Choose one of key attributes of E as primary key for R

在此我們做了以下轉換:

User					
User_ID	User_Name	User_Mail	User_Avatar	User_Password	
PK					
Project					
Project_ID	Project_Name	Project_Color			
PK					
Task_List					
Task_List_ID	Task_List_Name	Task_List_Status			
PK					
Task_Card					
Task_Card_ID	Task_Card_Name	Task_Card_Text	Task_Card_StartTime	Task_Card_EndTime	Task_Card_Status
PK					
Todo					
Todo_ID	Todo_Text	Todo_Status			
PK					
Comment					
Commenter_ID	Commenter_Name	Comment_Text			
PK					

- Step 2: for each weak entity type W
  - Create relation R
  - Include all simple attributes of W as attributes of R
  - Include primary key of relations that correspond to owner entity type as foreign key of R
  - Primary key of R = (primary key of owner, partial key of W)

  我們沒有 weak entity type，故跳過這個 Step

- Step 3: for each binary 1:1 relationship type R
  - Identity the relations S & T participating in R
  - Choose one relation S (better to choose entity type with total participation) & include the primary key of T as foreign key in S
  - Include all simple attributes of R as attributes of S

  我們沒有 binary 1:1 relationship type，故跳過這個 Step

- Step 4: for each binary 1:N relationship type R
  - Identify relation S at the N-side of relationship
  - Include primary key of relation T (the other side) as foreign key in S

- Include any simple attributes of R as attributes of S

在此我們做了以下轉換:

User						
User_ID	User_Name	User_Mail	User_Avatar	User_Password		
PK						
Project						
Project_ID	Project_Name	Project_Color	Mgr_ID			
PK			FK			
Task_List						
Task_List_ID	Task_List_Name	Task_List_Status	Project_ID			
PK			FK			
Task_Card						
Task_Card_ID	Task_Card_Name	Task_Card_Text	Task_Card_StartTime	Task_Card_EndTime	Task_Card_Status	Task_List_ID
PK						FK
Todo						
Todo_ID	Todo_Text	Todo_Status	Task_Card_ID			
PK			FK			
Comment						
Commenter_ID	Commenter_Name	Comment_Text	Task_Card_ID			
PK			FK			

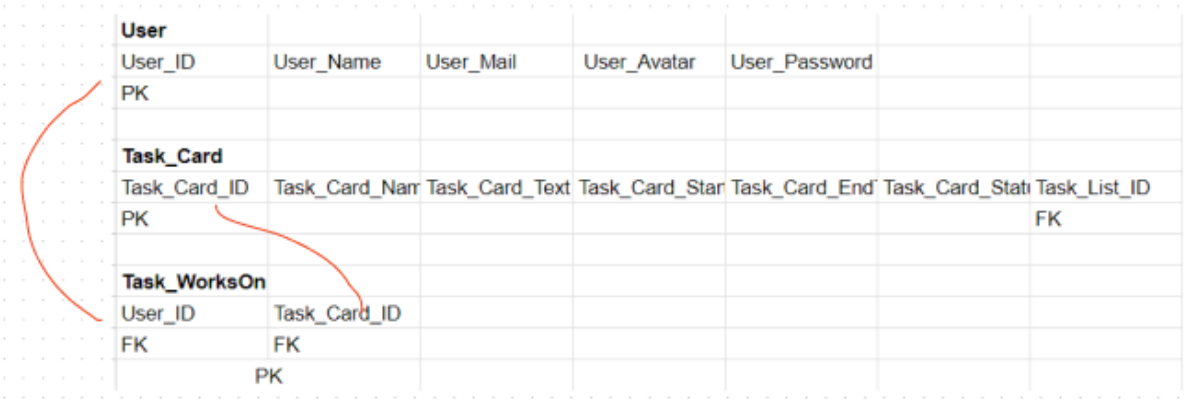
- Step 5: for each binary M:N relationship type R
  - Create a new relation S to represent R
  - Include primary keys of participating relations as foreign key of S
  - Their combination form primary key of S
  - Include any simple attributes of R as attributes of S

在此我們做了以下轉換:

a. Project_WorksOn

User				
User_ID	User_Name	User_Mail	User_Avatar	User_Password
PK				
Project				
Project_ID	Project_Name	Project_Color	Mgr_ID	
PK			FK	
Project_WorksOn				
User_ID	Project_ID			
FK	FK			
PK				

b. Task_WorksOn



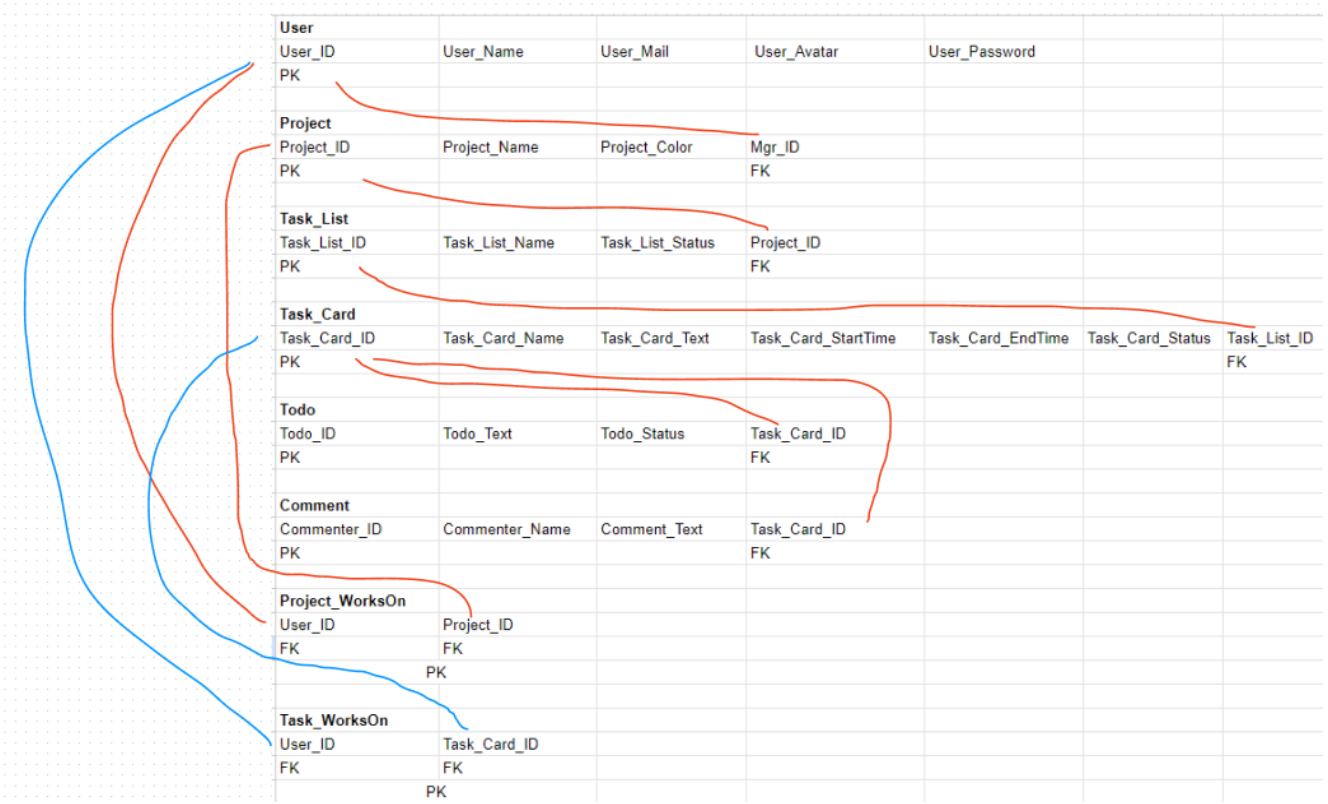
- Step 6: for each multivalued attribute A
  - Create a new relation R
  - Include an attribute corresponding to A
  - Include primary key K (as a foreign key of R) of relation S as an attribute
  - Primary key of R = (A, K) – If A is composite, include its simple components

  我們沒有 multivalued attribute，故跳過這個 Step

- Step 7: for each n-ary relationship type, n>2
  - Create a new relation S to represent R
  - Include primary key of participating relations as foreign key
  - Include any simple attributes of R as attributes of S
  - Primary key of S = all foreign keys that references participating R

  我們沒有 n-ary relationship type，故跳過這個 Step

b. 最終 Relational Schema 設計



- Relational Schema
  - User:
    - User_ID: String NOT NULL
    - User_Name: String
    - User_Mail: String
    - User_Avatar: String
    - User_Password: String

    PK -> User_ID

  - Project:
    - Project_ID: String NOT NULL
    - Project_Name: String
    - Project_Color: String

    PK -> Project_ID
    FK -> Mgr_ID References User(User_ID)
    ON DELETE CASCADE

  - Task_List:
    - Task_List_ID: String NOT NULL
    - Task_List_Name: String
    - Task_List_Status: Boolean

    PK -> Task_List_ID
    FK -> Project_ID References Project(Project_ID)
    ON DELETE CASCADE

- Task_Card
  - Task_Card_ID: String NOT NULL
  - Task_Card_Name: String
  - Task_Card_Text: String
  - Task_Card_StartTime: String
  - Task_Card_EndTime: String
  - Task_Card_Status: Boolean

  PK -> Task_Card_ID
  FK -> Task_List_ID References Task_List(Task_List_ID)
  ON DELETE CASCADE

- Todo
  - Todo_ID: String NOT NULL
  - Todo_Text: String
  - Todo_Status: Boolean

  PK -> Todo_ID
  FK -> Task_Card_ID References Task_Card (Task_Card_ID)

- Comment
  - Commenter_ID: String NOT NULL
  - **Comment_ID**: String NOT NULL
  - Commenter_Name: String
  - Comment_Text: String

  PK -> Todo_ID
  FK -> Task_Card_ID References Task_Card (Task_Card_ID)

- Project_WorksOn
  - User_ID: String NOT NULL
  - Project_ID: String NOT NULL

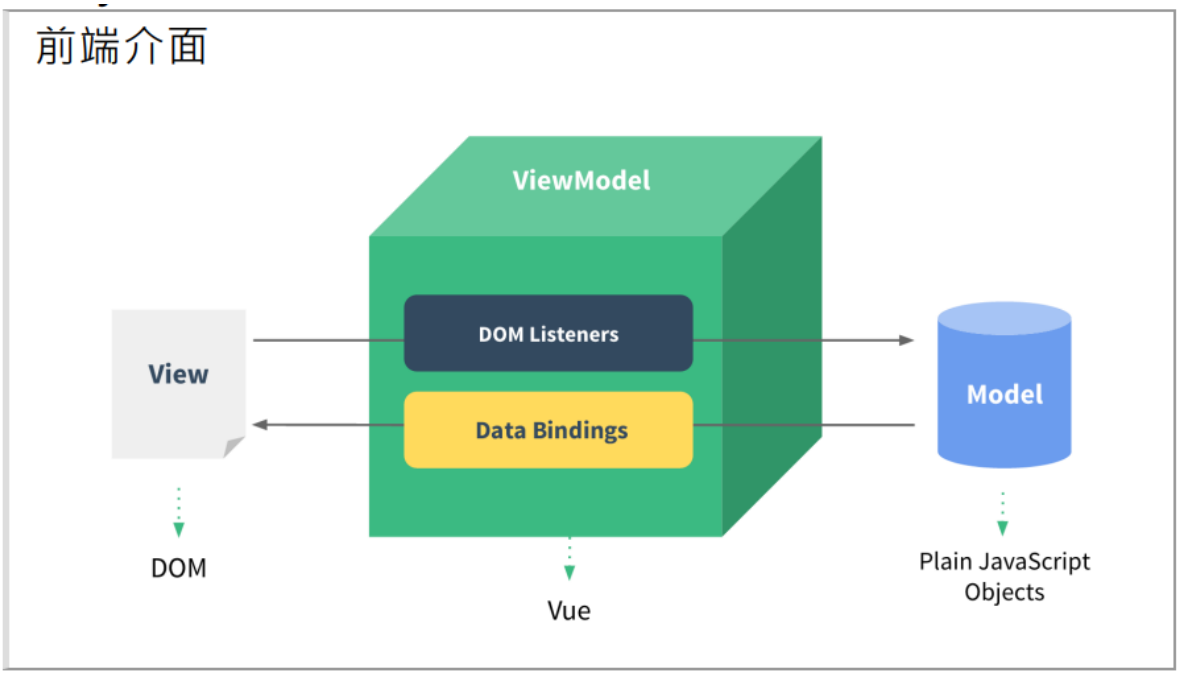
  PK -> (User_ID, Project_ID)
  FK -> User_ID, Project_ID References User(User_ID), Project(Project_ID)

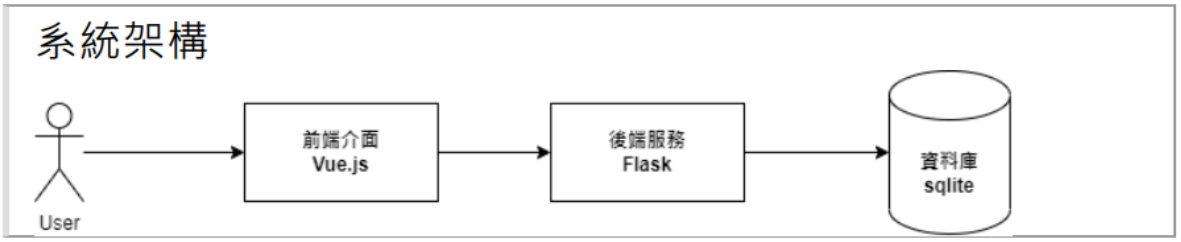
- Task_WorksOn
  - User_ID: String NOT NULL
  - Task_Card_ID: String NOT NULL

  PK -> (User_ID, Task_Card_ID)
  FK -> User_ID, Task_Card_ID References User(User_ID), Task_Card(Task_Card_ID)

8. 系統架構

專案採取前後端分離的 WEB 架構，前端採用 Vue.js 的 MVVM 網頁渲染方法，後端採取 RESTful 架構撰寫 API，彼 此以 json 格式溝通資料。





系統開發的程式語言、DBMS 與工具、系統模組為:

- 前端: Vue.js + Vite + Pinia
- 後端: Flask
- DBMS: sqlite

9. 心得、收穫與建議

組員們大多沒有網頁撰寫的底子，後端組的組員都是第一次使用 python + flask + sqlite3 進行後端開發，不過有些人有使用其他語言進行開發的經驗。最大的問題是溝通成本，雖然前後端分離的架構看似能降低大家工作的耦合度，但實際上後端組的成員對於 API 接口的理解還是有差異，協同合作上遇上了蠻大的問題，尤其是組員都沒做過前後端分離的專案讓這個缺口更加明顯。後來我們透過優先定義好 API 以及專案架構以解決溝通不良的問題，這次的專案我們獲得了設計 API，以及實際撰寫 SQL 應用的經驗，收穫頗豐!