DBMS Final Project Report

112550205 張程翔 112550069 謝侑哲 112550158 徐柏安 112550103 游翔宇 112550087 廖漢軒

1. Main Idea

我們希望製作一個現代化、高互動性的動畫資料整合網站，具有分類、檢索、登入、收藏清單、評價、留言等功能，讓使用者能夠更方便找到感興趣的動畫。

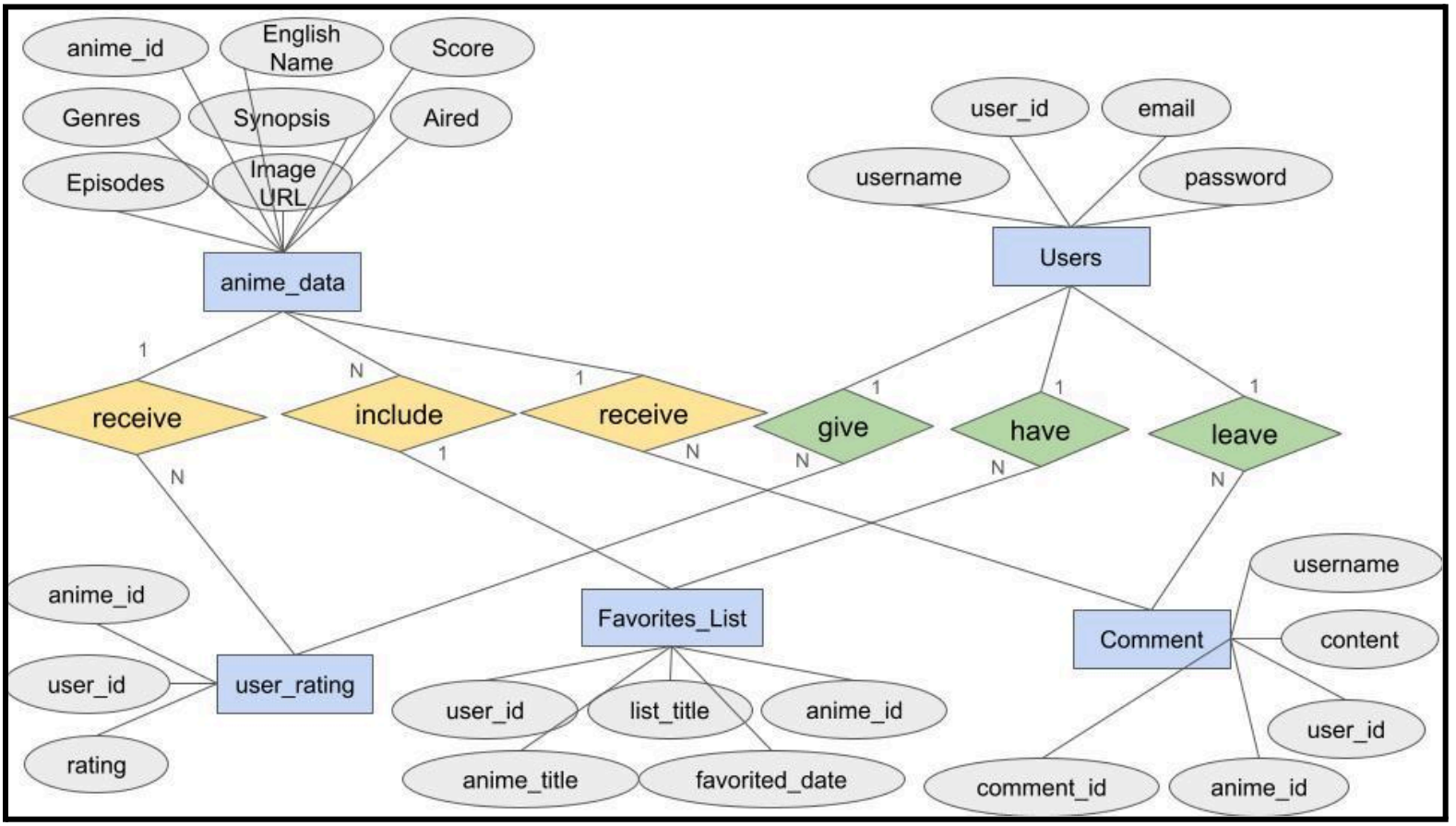
分類及檢索系統方便讓用戶找到自己想看的動畫。評分系統可以讓用戶給出自己的評分，並且參考其他用戶的評分，了解有甚麼好看的動畫。另外搭配留言功能，可以對於自己喜歡或不喜歡的動畫發表評論。收藏清單則是當用戶看完動畫的資料跟他人的評價後，可以把動畫收藏起來以後看的功能

2. Data

我們總共使用了5個tables，詳細內容如下：

Tables	Columns	Primary Key
anime_data (動漫詳細資料)	anime_id, Name (羅馬拼音名稱), English Name, Other name (日文名稱), Score, Genres, Synopsis (簡介), Type (TV/Movie…), Episodes, Aired (上映日期), Premiered (上映年份和季節), Status (現正上映/已完結...), Producers, Licensors, Studios, Source (原作小說/漫畫...), Duration, Rating(年齡限制), Rank, Popularity, Favorites, Scored By(評分人數), Members (加到清單中的人數), Image URL (圖片網址)	anime_id
user_rating (使用者的評價)	user_id, anime_id, rating (使用者評分)	user_id, anime_id
Favorites_List (收藏清單)	user_id, list_title, anime_id, anime_title, favorited_date(收藏的時間，用來排序清單)	user_id, list_title, anime_id
Users (新使用者登入資訊)	user_id, username, password, email	user_id
Comment (留言)	comment_id, anime_id, user_id, username, content (留言內容)	comment_id

ER Diagram:



3. Database

在Final Project, 我們選用PostgresSQL作為資料庫, 雖然MySQL在使用上更簡單, 但是PostgreSQL在功能、Concurrency Control上, 都有更進階的設計, 也更多我們可以操作的空間。因此我們選擇使用PostgreSQL。

在後端語言的使用上, 我們使用基於node.js的express.js框架, 因為他有很好的router, middleware的設計機制, 也有asynchronous的設計可以讓程式的運行更快, 更是比較普遍被使用的後端框架。雖然學習成本相較於成員們本來就會的python更高, 但是我們認為是值得投入去學習的, 因此選用express.js。另外, 我們也搭配python腳本來做資料的清洗。

而對於怎麼維護資料庫及與前端應用作互動, 我們將針對不同功能去做解釋。

a. 資料前處理 **&** 資料庫建置

我們的資料取自kaggle的資料集, 不過因為該資料集的資料品質不是很好, 有很多格式不統一、資料缺漏、特殊字元的問題。因此我們先寫了一個python腳本csv_preprocess.py來處理這些case。

接著，我們寫了一個/importCSV的API來把CSV資料引入，使用importCsvToDb的function來處理，當Table不存在時，會自動創建基於parameter的Table。因為處理的資料量巨大，有10G左右，所以使用PostgreSQL獨有的COPY SQL，來做到快速、批量的數據處理，把資料從CSV匯入到Database中。

在exception handling上，因為匯入的資料量大、時間久，中途比較有可能會出現一些意外，因此我們使用SQL的Begin、Commit來確保所有的資料都會被正常匯入，否則就Rollback所有的SQL，避免資料庫出錯。

b. 動畫排序以及**ID**查詢

這部分有兩個主要功能：(1)使用ID查詢動畫(2)依據上映年或是分數來排序動畫。這兩個Query皆無須更改任何資料，只有單純的查詢，所以不會有因為更新資訊而導致資料庫出錯(inconsistence)的情況。

此API是使用Express.js來定義路徑(/all 和 /:id)，當使用者傳了GET Request時，會依照內容決定該執行的路徑以及Query。像是/all?sort={sort_type}，是利用GET Request的sort來決定排序方式(上映年或是分數)，再利用Switch Case來定義該使用什麼Query達到使用者需求。而/:id也是使用GET Request，查詢特定的動畫ID並回傳特定資料，像是上映時間、完結時間、評分、簡介等等。

Exception Handling的部分的話，以ID查詢動畫時若所查詢的ID不存在，則會回傳錯誤訊息告訴前端是無效查詢。而依據上映年排序動畫則有點複雜，因為我們是使用anime_data這個table來查詢動畫，而裡面的欄位”Aired”的格式並不統一，有些為NULL或是”Not available”，有些則是只有上映年份，又有些是包含上映年月日以及完結年月日。所以在處理這些問題時要先將NULL以及”Not available”的欄位去除掉，並在Query中利用CAST()和substring()來擷取上映年份，像是：CAST(substring("Aired" FROM '\\d{4}') AS INT)會擷取第一個連續4個數字(上映年份)並轉換成integer型別，再排序。若所要求的排序方式有誤的話，會回傳錯誤訊息給前端是無效排序方式。

c. 動畫分類系統

此部分也分為兩個功能：(1)回傳現有的所有分類(2)依據分類回傳動畫。這兩個功能一樣沒有更改任何資料，所以不會有因更新資訊而導致資料庫出錯(inconsistence)的問題。

此API也是使用Express.js來定義路徑(/ 以及 /:category)。並使用GET Request，依照內容決定該執行的路徑以及Query。/ 為單純

回傳所有分類，/:category?sort={sortType}則是會根據參數(哪個category以及哪個排序方式)，回傳anime_id。

回傳所有分類的Query的部分，因為在anime_data這個table內的”Genres”的格式是將不同的分類用”, “隔開，所以SELECT的部分時使用SELECT DISTINCT UNNEST(string_to_array("Genres", ', ')) AS genre 來取得所有不同的分類。

依據分類以及排序方式回傳動畫則是跟 b. 的方式非常類似，一樣是利用/:category?sort={sortType}的參數決定排序方式，也是利用Switch Case來決定該使用哪個Query。而最重要的判斷分類的部分是使用WHERE “Genres” ILIKE來達成。

Exception Handling的部分，如果收到的排序方式無效或是收到一個不存在的分類的話，會傳錯誤訊息給前端。

d. 動畫搜尋系統

在搜尋上，我們主要完成的功能有二: (1)搜尋, (2)搜尋結果排序，都是基於GET request的功能，以下將針對這兩個功能敘述。

在搜尋的參數接收上，我們嘗試使用URL參數，來實現，透過在URL的後面加上?keyword={...}&sort={...}的方式，讓前端能將參數傳入。

得到搜尋的參數後，使用SELECT搭配ILIKE的方式，做到soft search，只要動畫的名稱中帶有關鍵字，就可以被搜尋出來。另外再利用ORDER BY來做排序。

關於排序的部分，是最困難的，我們的搜尋排序有4種，按照播出時間、分數高低做排序。其中，因為資料集中有相當多不規則的資料，因此我們使用了不同的SQL來做exception handling。

在分數的排序上，因為常常會出現分數為null的狀況，因此使用COALESCE("Score", 0)來解決分數為null的狀況。

在時間的排序上，使用的方式比較複雜，因為時間的格式非常混亂，有”2015”, “Jul, 2015”, “Jul 2, 2015”...等等，一堆的格式，因此我們使用CASE WHEN的方式來處理，具體的處理SQL如下:

```
sortBy = `
    CASE
        WHEN "Aired" ~ ' to ' THEN TO_DATE(SUBSTRING("Aired" FROM '^(.+) to'), 'Mon DD, YYYY')
        WHEN "Aired" ~ ', ' THEN TO_DATE("Aired", 'Mon DD, YYYY')
        ELSE TO_DATE('Jan 1, ' || "Aired", 'Mon DD, YYYY')
    END
`;
```

基於PostgreSQL的特性，可以使用正規表達式來判斷日期的格式，並且使用SUBSTRING、STRING OR的方式進行日期的處理，最後再使用TO_DATE依據處理完的格式，轉成DATE的data type來進行正確的排序。另外，針對完全例外的格式，也透過WHERE配

合正規表達式的方式，限制雖然格式可以有變化，但必須有某些條件才能被搜尋。

e. 登入登出系統

此部分主要有四個功能：(1)登入(2)註冊帳號(3)更改密碼(4)回傳使用者資訊。這些功能基於GET POST以及PATCH。

我們的使用者資訊存在Users的Table中，密碼的部分則會使用bcrypt來加密再存到資料庫內。登入時也會先以相同的加密方式再跟資料庫內的密碼對照。

登入時使用POST，需要Request Body中的username和password，並先使用Query來尋找有沒有此username，再來將輸入的password加密並和Query找到的password對比。至於如何解決SQL Injection，我們使用參數式查詢(Parameterized Queries)，如下圖。這樣能保證[username]這個變數會被視為單純的字串，不會被解讀成指令。像是WHERE username = 'admin\' OR \'1\'=\'1'，而非WHERE username = 'admin' OR '1'='1'。

```
const query = 'SELECT * FROM users WHERE username = $1';
const { rows } = await pool.query(query, [username]);
```

登入之後會使用jsonwebtoken來產生時常為1小時的token，裡面包含了user_id的資訊，登入成功後會回傳token給前端並記錄下來，以供未來使用。

註冊也是使用POST，需要Request Body中的username，email以及password。首先會先檢查Users Table中有沒有重複的username或是email，有的話會回傳已經註冊的錯誤訊息給前端。再來會將輸入的password使用bcrypt加密之後再存入資料庫。Query的部分同樣使用參數式查詢(Parameterized Queries)，確保SQL Injection不會發生。

更改密碼則是使用PATCH，需要使用者輸入原密碼以及新密碼。若原密碼不符合，則會傳錯誤訊息告訴前端密碼不符合。如果有少填的欄位則會告訴使用者原密碼以及新密碼都需要輸入。舊密碼如果是正確的話，會使用bcrypt將新密碼加密，並用UPDATE Users SET password來更新資料庫裡面此使用者的密碼。

查詢使用者是使用GET，路徑為/search/:username。藉由URL後面username的參數來決定查詢哪位使用者，Query一樣使用參數式查詢(Parameterized Queries)來預防SQL Injection。若無法查到此使用者名稱，則會傳錯誤訊息告訴前端查無此使用者。

f. 收藏清單系統

在收藏清單的部分，有幾個主要的功能要完成(1)清單建立(2)搜尋使用者擁有的清單(3)插入動畫進清單(4)從清單刪除動畫(5)刪除整個清單(6)取得清單內有哪些動畫，這些功能都是基於POST request的，對於這些功能，將先綜述，再針對每項功能敘述。

我們的收藏清單的資料庫儲存邏輯，是將每一筆被收藏的動畫都放入同一張Table - “Favorites_List”中，其中包含了user_id, list_title, anime_id, anime_title, favorited_date，因為所有的收藏紀錄都存在同個Table中，因此要區分每一筆資料就基於primary key: (user_id, list_title, anime_id)，也就是每個使用者可以有相同的清單命名、動畫收藏，但是自己不能有兩個同樣的清單，並且同個清單中不能插入相同的動畫。透過這些column，可以有效的在一個Table內分出不同使用者擁有的清單，而不需要大量的創建Table。接下來是分述部分。

在清單建立的部分，因為實現邏輯是紀錄每一筆被收藏的動畫，而創建清單時，是不會收藏動畫的，因此這裡使用假動畫資訊來插入，將插入的anime_id, anime_title都設為-1，以此來實現清單的創現。而exception handling則是當參數不足或是清單被判定已存在時，直接回傳400 Bad Request回前端。另外，Table不存在時，也會在create地同時自動創建。

在搜尋使用者擁有的清單時，則是直接對於某個user_id做SELECT DISTINCT list_title並回傳擁有的list有哪些。

關於插入動畫進清單的功能，就是最基本的功能，輸入user_id, anime_id, list_title等資訊，並在資料庫中去尋找anime_id對應的動畫名稱等等的資訊，再將資料插入進Table中，使用邏輯就跟前面描述的核心邏輯相同。在exception handling的部分則是取得anime_id時，會先在database裡面做搜尋，避免插入不存在的幽靈動畫，也會對於重複的動畫id報錯，避免重複內容被插入。

關於刪除動畫跟清單，就是簡單使用primary key或是(user_id, list_title)，配合SQL的DELETE來做row的單個或批量的刪除。exception handling部分，則是會對於不存在的動畫或是清單報錯。

取得清單的動畫，跟刪除的做法差不多，同樣是透過(user_id, list_title)來取得清單內的所有動畫，同樣對於不存在的動畫或是清單報錯。

g. 留言系統

在留言的部分，有幾個主要的功能(1)加入留言(2)取得動畫底下的所有留言(3)刪除留言，皆是基於POST request的功能。

在Table的設計上，因為一個使用者可以在同一個動畫底下有多個留言，因此獨立出一個comment_id來做為primary_key。核心的Table邏輯是紀錄anime_id, user_id, content及其他附加的資料，變成在某個動畫底下，某個用戶留言了某些內容。跟收藏清單的運作概念類似，一樣是利用一個Table儲存所有的留言內容

關於加入留言的功能，比較簡單，就是將輸入的參數直接插入Table中。

在刪除留言的部分，雖然也是直接對於留言做刪除，但是有在前端做額外的exception handling，當登入的使用者，他刪除的留言得user_id與該使用者不同時，那就會在前端就擋掉使用者的刪除操作。另外，刪除留言有分兩種方式，一種是基於comment_id的刪除，也就是對於特定留言的刪除，算是絕對的刪除單個留言的手段，另外一種則是基於anime_id, user_id, content來做刪除，這是比較軟性的刪除手段，但是如果有人在動畫底下大量刷同樣留言時，就可以透過這個API直接把刷頻的留言全部刪掉。

在取得留言的部分，則是針對某個動畫，利用anime_id去SELECT所有內容，並回傳給前端。對於不存在的anime_id的exception handling則是處理成直接回傳空值。

h. 評分系統

此系統主要有三個功能:(1)新增評分(2)移除評分(3)取得評分。這三個功能都是使用POST。

這部分會更新到兩個table，第一個是user_rating，第二個是anime_data。使用者對動畫做的評分的資訊會被存入user_rating，同時anime_data內的”Score”(評分平均)以及”Scored_By”(總評分人數)會被更新。

首先是新增評分，跟新anime_data的”Score”時，得先判斷這個動畫的”Score”是否為NULL，使用COALESCE("Score", 0)可以將”Score”為NULL的值跟改為0。計算新的”Score”時，要先將原本的”Score” * “Scored_By”。接下來加上目前使用者的評分，再除以”Scored_By” + 1，最後再將”Scored+By”的值加一。

```
await pool.query(`
    UPDATE anime_data
    SET "Score" = (
        ((COALESCE("Score", 0) * COALESCE("Scored_By", 0)) + $1) / (COALESCE("Scored_By", 0) + 1)
    ),
    "Scored_By" = COALESCE("Scored_By", 0) + 1
    WHERE "anime_id" = $2
`,[score, anime_id]);
```

下一步是新增評分到user_rating中，紀錄使用者的評分。這部分比較簡單，只需要INSERT正確的anime_id、user_id以及rating即可。

移除評分的流程基本上和新增評分一樣，只是這次是將data從user_rating移除，並修改anime_data的資料而已。刪除user_rating的資料使用DELETE FROM user_rating WHERE的方式即可。

```
await pool.query(`
    UPDATE anime_data
    SET "Score" = CASE
            WHEN "Scored_By" - 1 = 0 THEN NULL
            ELSE ((COALESCE("Score", 0) * COALESCE("Scored_By", 0)) - $1) / (COALESCE("Scored_By", 0) - 1)
        END,
        "Scored_By" = GREATEST(COALESCE("Scored_By", 0) - 1, 0)
    WHERE "anime_id" = $2
`, [score, anime_id]);
```

取得評論的方式為利用user_rating的資料，利用user_id以及anime_id來查詢此user對某個動畫的評分為幾分。如果此user_id或是anime_id不存在，又或是此使用者沒有對這個動畫評分，會傳錯誤訊息告訴前端不存在此評分。

後端API document連結: https://reurl.cc/1XK0jW

4. Application

我們的Anime Helper網頁主要有9個功能。

a. 名稱搜尋 (Search by Sort and Name)
我們使用`/api/search/Soft/?keyword={keyword}&sort={sort_type}` (get)，用react的useState hook去保存sort下拉式選單的value，以及輸入欄裡面的value，在送出按鈕被按下後，執行api的呼叫。當使用者在輸入欄裡面填寫內容時，不能同時使用category分類，使用者無法點選種類按鈕，避免意外情況的發生。這個部分對應到資料庫的R

b. 種類搜尋 (Search by Sort and Category)
我們使用`/api/category/{category}?sort={sort_type}`(get)，用react的useState hook去保存sort下拉式選單的value，以及所選中的Category，在送出按鈕被按下後，執行api的呼叫。當使用者選擇用種類來搜尋時，不能同時使用輸入欄作為搜尋的依據。因此，當某一個種類被選中時，會禁用輸入欄的使用，避免意外情況的發生。這個部分對應到資料庫的R。

c. 排序動漫 (Sort Anime)
我們使用`/api/anime/all?sort={sort_type}`(get)，用react的useState hook去保存sort下拉式選單的value。當輸入欄的value為空字串，且沒有

選中任何Category時，我們會執行這個api的呼叫。這個部分對應到資料庫的R。

d. 註冊帳號
我們使用`/api/auth/register` (post)，用react的useState hook去保存表單的內容，並使用preventDefault來阻止表單提交後重新刷新頁面的行為。按下送出按鈕後，我們會檢查confirmed password是否和password一致，防止使用者誤輸密碼，提升穩固性。我們新增了提示文字的功能，可以跳出錯誤、成功訊息和使用者互動。完成註冊後，會跳轉到登入的介面。這個部分對應到資料庫的C。

e. 登入
我們使用`/api/auth/login` (post)，用react的useState hook去保存表單的內容，並使用preventDefault來阻止表單提交後重新刷新頁面的行為。按下送出按鈕後，如果成功登入，則會跳回主頁面，且右上方的sign in按鈕會即時替換成使用者的icon，從這裡可以看到使用者的profile。這個部分對應到資料庫的R。

f. 修改密碼
我們使用`/api/auth/update-password` (PATCH)，用react的useState hook去保存表單的內容，並使用preventDefault來阻止表單提交後重新刷新頁面的行為。另外，前端這邊會先檢查confirmed password是否和password一致，防止使用者誤輸密碼，提升穩固性。若密碼沒問題，會將user_id, old_password, new_password等放到request body一同送出。這個部分對應到資料庫的U。

g. 收藏動漫
這個功能需要使用者登入後才能操作，因此進到這個頁面時，我們會檢查當前使用者的登入狀態，若尚未登入則跳轉到login頁面。當使用者想要查看其收藏的動漫清單時，我們會使用`/api/favorite/getList` (POST)來取得。創建新的收藏清單時，我們使用`/api/favorite/create` (POST)。當使用者想要使用快速收藏（預設），或是將動漫加入新的收藏清單時，我們使用`/api/favorite/insert` (POST)。當使用者想要進行刪除時，我們使用`/api/favorite/deleteList` (POST)、`/api/favorite/deleteAnime` (POST)。以上都能藉由簡單的按鈕、useState hook、提示訊息，讓使用者簡單操作，並清楚的看到操作是否成功。這個部分對應到資料庫的CRUD部分，我們可以輕鬆創建、讀取、更新、刪除最愛清單。

h. 留言
使用者可以新增留言，用到`/api/comment/add` (POST)。另外，所有人（即便未登入）都可以看到所有留言，因此我們會用`/api/comment/get`

(POST)去拿到所有評論。由於使用者可以管理他自己的留言，因此我們會使用`/api/comment/get` (POST)來查看使用者是否是這個留言的發布者，若結果為是，則前端這邊會顯示一個刪除按鈕，讓使用者可以刪除他自己的留言，用到`/api/comment/deleteByID` (POST)。反之，如果使用者尚未登入，或是不是這個留言的發布者，則我們不會顯示留言的刪除按鈕。這個部分對應到了資料庫的CRD，使用者可以新增、查看、刪除留言，並且妥善管理權限。

i. 評分
我們使用`/api/score/add` (POST)，`/api/score/remove` (POST)，`/api/score/getScore` (POST)來進行新增、移除、查看分數的功能。使用者的評分只有在登入後才看得到。這個部分對應到資料庫的CRD

search anime / sort anime / sort by category / login / register / modify pw email / favorite list / comment list / score anime

5. Others

- Github repository:
  - https://github.com/NYCU-DBMS/DBMS-Final-Porject
- Youtube link:
  - https://www.youtube.com/watch?v=fT7bDZtwl6s
- Progress Status of Project:

前端規劃	規劃日期	實際日期
學習React.js	11/9	11/12
首頁	11/15	11/17
動畫頁面	12/22	12/23
分類功能	12/22	12/23
搜尋功能	12/22	12/23
登入功能	12/23	12/25
收藏清單功能	12/24	12/24
評分功能	12/25	12/26
留言功能	12/26	12/27

後端規劃	規劃日期	實際日期
學習 Express.js	11/9	11/8
建置Database	11/15	11/22
動畫資訊API	12/21	12/22
分類API	12/21	12/22
搜尋API	12/21	12/22
登入API	12/22	12/24
收藏清單API	12/23	12/24
評分API	12/24	12/25
留言API	12/25	12/25

Problems met in Project:

前端:

在前端的部分，我們遇到了下列問題:

a. Anime資料集提供的URL問題
   在處理資料集時，我們發現部分anime的URL存在問題，會出現連結失效的情況。為了解決這個問題，我們在前端渲染出動漫之前，額外新增了一個機制，以檢查URL的可用性，確保只有有效的連結才會被放到搜尋結果上。
b. 畸形、噁心畫風的過濾
   在展示內容時，我們也注意到有些動漫的畫風有異常情況，甚至會讓用戶感到不適。針對這個問題，我們主動整理了相關的id，並建立了一套過濾系統，在展示內容之前進行檢查，避免這些不適宜的內容影響整體體驗。

後端:

主要的問題都在於我們所選的資料集裡面，有很多一個欄位的內容格式不統一、資料缺陷又或是出現特殊字元，導致從最一開

始處理資料、匯入database、實際寫API並使用Query查詢，都花了不少時間和心力。

在一開始的資料處理中，因為原本的資料集很髒，有很多不合規範的資料，因此我們一開始想要處理資料的方式是，一筆一筆的邊處理邊匯入資料，但是因為處理的CSV檔案高達10GB，這樣的處理方式會導致記憶體無法負荷。一開始我們不斷嘗試去shrink記憶體消耗的方法，但不管怎麼shrink都沒辦法讓記憶體負荷這麼大的資料。到了最後，才發現根本的方法就錯了，不應該邊處理邊匯入，才改成後來的先進行資料愈處理，再做匯入的功能。

這個問題讓我們發現，遇到一個問題時，很多時候應該先從根源去解，而不是問題出在哪裡，就直接對於問題去處理，這樣處理下來可能用了各種trick都沒辦法成功解掉問題。觀察問題的本質才能有效做出處理。

在處理動畫上映時間、完結時間以及依照年份排序時，我們後端組其實花了不少時間處理，這也是因為記錄這些資料的欄位裡面的資料格式沒有統一，裡面總共有6種表示方式：(1) DD Mon, YYYY to DD Mon, YYYY (2) DD Mon, YYYY (3) Mon YYYY (4) YYYY (5) Not available (6) NULL，非常的雜亂。最後我們使用CAST()、CASE、SUBSTRING()等等方法並結合起來，再搭配正規表示式(Regular Expression)，將此欄位分成上映時間以及完結時間，來將動畫以正確的方式排序。

此外還有因為我們都是第一次使用Express.js寫API，所以需要從頭開始學習語法，而且我們是使用PostgreSQL，雖然也有部分和課程中所學的MySQL重疊的部分，但也有許多不同之處。有時候也會為了一些比較特殊的需求，像是怎麼使用SQL裡面的正規表示式的寫法、CASE的語法、UNNEST()等等function，都是需要在網路上學習的。不過也還好現在網路上的工具非常方便，官網的教學手冊也寫得很清楚，Stack Overflow上也有很多高手們的解答，再加上現在的AI工具越來越厲害，只要知道如何正確並有效地使用網路上的資源，其實很快就能上手了。

全端:

在前後端串接的過程中，因為前後端分離的設計，兩邊不能完全知道另外一邊的狀況，因此常常出現前端的在使用API實現功能時，出現了後端沒預期到的問題，因此需要即時的on call做功能的增修跟debug，也常常會有溝通斷層的問題。

總結來說。在串接的過程中，遇到了滿多資訊差異、溝通錯位的問題。對於這類的問題，很大一部分來源於訊息傳遞的問題，我們大多數的訊息都是直接在Discord群做溝通，而沒有系統性整理訊息、專案管理的方式。因此比較好的做法，會是在未來的project中，引入專案管理的方法及工具，幫助所有人能認清目前專案的進度以及自己該解的issue等等，也讓大家能夠更平行化的去工作，而非一個等一個的讓整個專案的進程被某個瓶頸卡住。

Contribution:
前端開發: 廖漢軒、游翔宇、徐柏安
後端開發: 張程翔、謝侑哲
Presentation: 謝侑哲
Demo: 廖漢軒、張程翔
影片後製: 張程翔