國立中央大學資訊管理學系

軟體設計規格書

指導教授：許智誠 教授

第七組

113423023 鄒秉叡 113423032 林子恩

113423037 陳俐吟 113423040 賴昀廷

113423056 王詠平 113423068 鄭博修

目錄

第 1 章 簡介

軟體設計規格書係依據軟體產品之主要使用者之需求規格文件、分析規格文件進行規範，主要用於描述實際設計之軟體架構與系統範圍之文件。藉由本文件得以了解軟體系統架構之目的，並作為軟體實作之依據。

1.1 文件目的

本文件之目的用於提供軟體系統開發人員設計之規範與藍圖，透過軟體設計規格書，開發人員可以明確了解軟體系統之目標與內容，並得以此為據遵照共同訂定之規格開發軟體系統，以達到多人分工與一致性。

1.2 系統範圍

本系統範圍用於線上募資平台，其中主要包含管理員管理、會員管理、提案資訊列表、提案管理、提案結帳、購買紀錄、chat bot 聊天、 評論管理等模組，並且能進行相關新增、查閱與維護工作。藉由此系統支持完成線上募資平台所需的管理流程。

1.3 文件架構

1. 第二章撰寫本系統設計之架構，含環境需求、部署說明及系統架構圖。
2. 第三章描述本系統軟體之 MVC 設計架構，包含顯示層、商業邏輯層以及資料層
3. 第四章表達本專案之撰寫風格與規範，以達到多人協同便於維護用。
4. 第五章撰寫設計階段之資料庫架構 ERD 圖，含資料表之元素與特殊要求。
5. 第六章描述設計階段之類別圖，包含細部之屬性與方法。
6. 第七章講述每個 use case 之細部循序圖，以供實作階段使用。

第 2 章 系統架構

該章節說明本專案系統所開發之預計部署之設備與環境需求，同時說明本專案開發時所使用之第三方軟體之版本與套件，此外亦說明專案所使用之架構與未來部署之方法。

2.1 環境需求

2.1.1 伺服器硬體

本專案預計部署之設備如下：

1. OS：GCP App Engine
2. CPU：e2-medium （2 個 vCPU，4 GB 記憶體）
3. RAM：10 GB

2.1.2 伺服器軟體

為讓本專案能順利在不同時期進行部署仍能正常運作，以下為本專案所運行之軟體與其版本

- Node.js Version： Node.js v18.12.1
- Web Application Server： Express.js 4.18.1
- Database： MongoDB 6.0
- Cloud Database Service： MongoDB Atlas
- IDE or Text Editor： Microsoft Visual Studio Code 1.74.1
- 專案類型： MERN Full-Stack
- 程式語言：JavaScript
- ChatBot ： IBM Watson Assistant

2.1.3 客戶端使用環境

本專案預計客戶端（Client）端使用多屏（行動裝置、桌上型電腦、平板腦等）之瀏覽器即可立即使用，因此客戶端之裝置應裝載下列所述之軟體其一即可正常瀏覽：

- Google Chrome 76 以上
- Mozilla Firefox 69 以上
- Microsoft Edge 44 以上

- Microsoft Internet Explorer 11 以上

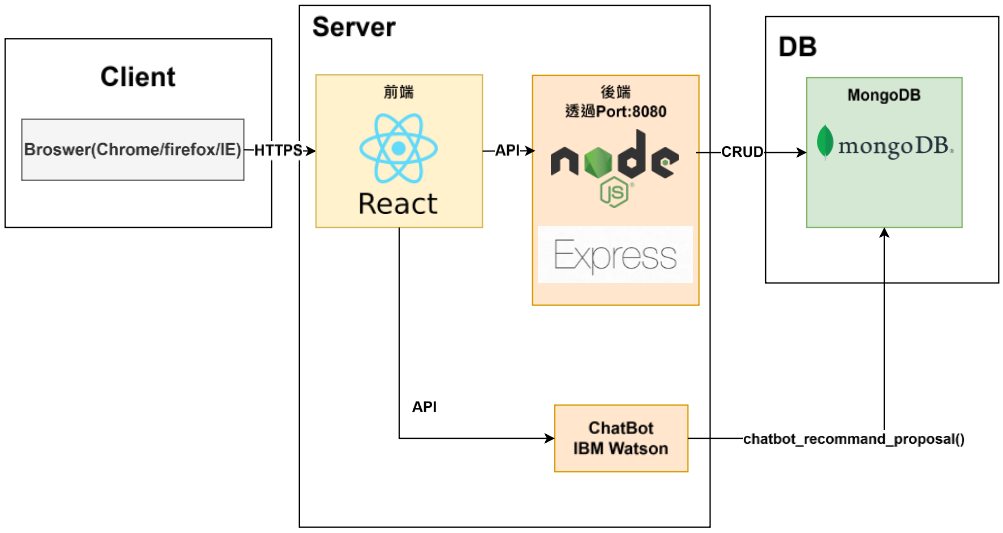
除以上之瀏覽器通過本專案測試外，其餘之瀏覽器不保證其能完整執行本專案之所有功能。

2.2 部署說明

實際觀點（physical view）是從系統工程師的觀點呈現的系統，即真實世界的系統拓樸架構，可以描述最後部署的實際系統架構和軟體元件，也稱為部署觀點（deployment view）。本專案線上贊助平台，使用 NodeJS 平台技術建構 Web 應用程式。其實際觀點模型的部署方式，如下所述:

1. 本專案之部署方式，其硬體與軟體規格如前揭（2.1 環境需求）所述。
2. 本專案之網頁伺服器軟體與資料庫同屬相同之伺服器且所有流量皆導向相同之伺服器（Server）。
3. 整個專案之檔案將分成前端與後端分別進行部署，資料庫將使用 MongoDB Atlas，並設定完成所需之帳號、密碼。
4. 本專案之網頁伺服器埠號採用預設之 8080。
5. 客戶端僅需使用裝置上瀏覽器，藉 https 即可連上本專案網站。

2.3 系統架構圖



圖一 系統架構圖

第 3 章 軟體設計架構

3.1 MVC 架構

本專案採用三層式(Three-Tier)架構，包含顯示層(Presentation Layer)、商業邏輯層(Business Logic Layer) 與資料層(Data Access Layer)。

此外，本專案使用 React、TailwinCSS 等框架進行網站前端之開發，並提供權限之管理。而後端則採用 Node.js、MongoDB Atlas。前後端將使用 API 進行溝通，且資料格式定義為 JSON 格式，以便於小組共同作業、維護與並行開發，縮短專案開發時程，並增加未來更動之彈性。以下分別論述本系統之三層式架構各層級:

3.1.1 顯示層(Presentation Layer):MVC-View

1. 顯示層主要為 JSX 檔案，放置於「carryu/client/src/components/*」，其中靜態檔案則放置於「carryu/client/src/assets/*」資料夾當中。
2. 網頁皆使用 HTML 搭配 CSS 與 JavaScript 等網頁物件作為模板。
3. JavaScript 部分採用 JSX 之方式撰寫。
4. API 之溝通採用 AJAX 方式進行，並透過 JavaScript 重新更新與渲染(Render)置網頁各元素。

3.1.2 商業邏輯層(Business Logic Layer)

1. 商業邏輯層於本專案中主要以 JavaScript 程式語言進行編寫，其中可以分為 「Business Model」和「View Model」兩部分。
2. Business Model: MVC-Model
   - 主要放置於「coco/server/models/*」資料夾當中，主要以 JavaScript 撰寫，有兩個 Model 於其中:UserModel.js 、ProposalModel.js。
   - 主要用於處理邏輯判斷資料查找與溝通，並與資料層(DB)藉由 MongoDB 進行存取，例如:SELECT、UPDATE、INSERT、DELETE。

- 其他功用與雜項之項目則統一會放置到「coco/server/config /*」資料夾當中，其中包含 database.js、validation.js 等

3. View Controls:MVC-Controller
   - 主要放置於「coco/server/controller/*」資料夾當中。
   - Controller 主要為 View 和 Model 之溝通橋梁，當前端 View 發送 AJAX Request 透過 ExpressJS 提供之 Router 指定處理的 Controller，並由後端之 NodeJS 進行承接。
   - 主要用於控制各頁面路徑(Route)與功能流程，規劃之 usecase 於此處實作，主要將 model 所查找之數據進行組合，最終仍以 JSON 格式回傳給使用者。
   - 命名會以*Controller 進行命名

3.2.3 資料層(Data Layer)

1. 作為資料庫取得資料的基本方法之用，藉由第三方模組 Mongoose 進行實現。
2. 透過編寫 database.js 之檔案進行客製化本專案所需之資料庫連線內容。
3. 透過 import 此 database 的 function 便能套用到不同的功能當中，而本專案只會引入至./server/index.js 當中，只要建立連線後便能對資料庫進行操作。

第 4 章 專案撰寫風格

4.1 程式命名風格

程式命名風格(coding convention)為系統實作成功，維持產出的品質以及往後之維護，需先進行定義實作上之規範。以下說明本專案系統之變數命名基本規則，以增加程式碼可讀性，同時也讓相同專案之成員能快速理解該變數所代表之意義，以達共同協作之目的。

1. 通用規則
    1.1. 縮行兩個空白，可使用 tab(須於 VSCode 中設定 Tab Size)。
    1.2. code 一行超過 100 個字元即折行。
2. 單字組成方式
    2.1. 動作:get/do/delete/check 等。
    2.2. 附加欄位:主要與該欄位之涵蓋範圍有關。如:duplicate/all 等。
    2.3. 主要關聯之資料庫資料表。
3. 前端檔案名稱
    3.1. React Component (.jsx)採用「Upper Camel Case 命名法」單一單詞字首皆大寫，不包含底線或連接號。例如: HomePage.jsx。
    3.2. 其他 Common JavaScript 的檔案接採用「小駝峰式命名法 (lower camel case)」，首字皆為小寫，第二單字開始首字為大寫。例如:authService.js、localStorage.js 等。
4. 後端檔案名稱(controller, route, middleware, model...)
    4.1. 採用帕斯卡命名法（Pascal Case）。在帕斯卡命名法中，單詞的首字母都大寫，並且不使用空格或底線連接，包含 controller、route、middleware 等等。
        - controllers/: 命名以*Controller 為主(例如: UserController.js)。
        - routes/: 命名以*Route 為主，為後端之函式。
        - models/: 命名以*Model 為主，為資料之模型與 Schema。

5. 函式(Method)

5.1. 主要採用「snake case 命名方式」，首字皆小寫以底線區隔。 例如: user_register()、proposal_edit()等。

6. 變數(Variable)

6.1. 主要採用「snake case 命名方式」，首字皆小寫以底線區隔。 例如:purchase_history 等。

4.2 回傳訊息規範

1. 透過 JsonReader 類別之 response()的 method 進行回傳，主要需要傳入要回之物件與將 Express 之 HttpServletResponse 物件。

1.1. 該 method 使用 Overload(多載)之方式，允許傳入 JSON 格式之字串或 JSONObject。

1.2. 欲回傳資料給予使用者皆應在 Controller 之 method 最後呼叫該方法。

2. Controller 無論回傳正確或錯誤執行判斷後之訊息皆使用 JSON 格式。

2.1. API 回傳資料之組成包含三個 KEY 部分，以下分別進行說明:

- status

錯誤代碼採用 HTTP 狀態碼(HTTP Status Code)之規範如下所示:

o 200: 正確回傳。

o 400: Bad Request Error，可能有需求值未傳入。

o 403: 權限不足。

o 404: 找不到該網頁路徑。

o 500: 伺服器錯誤，可能是伺服器沒有成功處理錯誤例外或是伺服器非正常運行。

- message 及 err_msg

o 主要英文回傳所執行之動作結果。

o 可用於後續渲染(Render)至前端畫面。

- response

儲存另一 JSON 格式物件，可隨所需資料擴充裡面的值。

4.3 API 規範

1. 路徑皆採用「api/*」。

2. API 採用 AJAX 傳送 JSON 物件。

3. 透過實作 Express Router 之方法，其對應如下:

    3.1. GET:用於取得資料庫查詢後之資料。

    3.2 POST:用於新增資料與登入。

    3.3 DELETE:用於刪除資料。

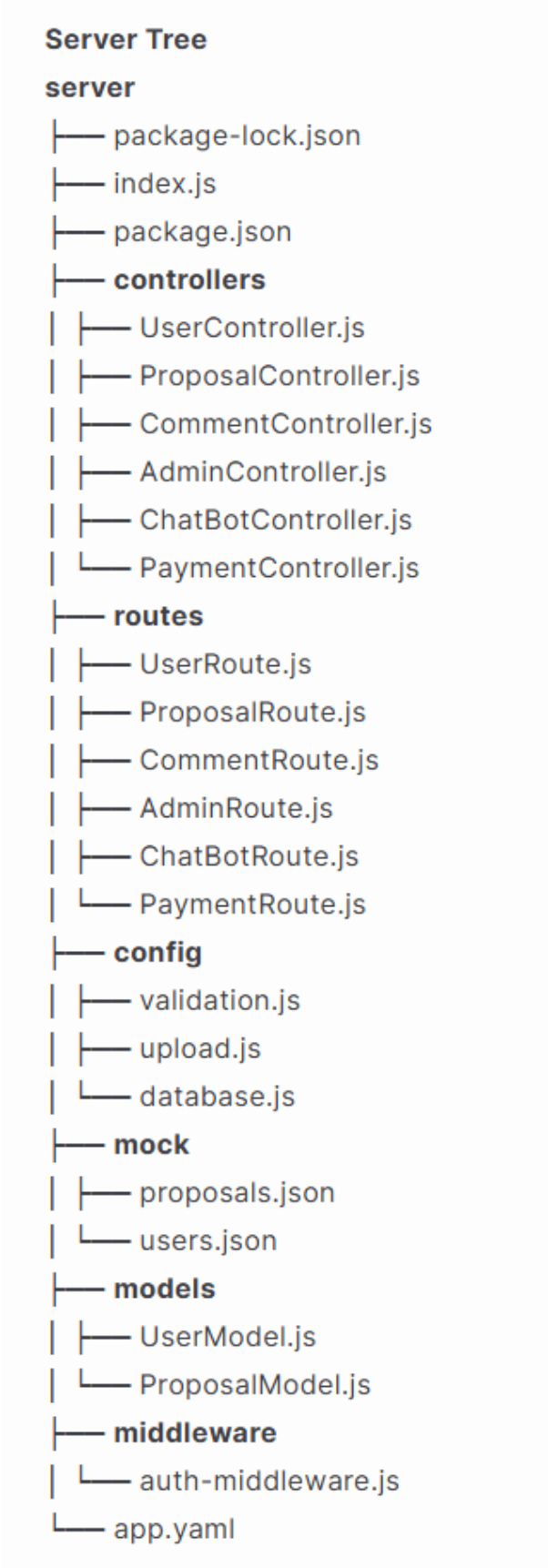
    3.4 PUT:用於將全部資料進行更新作業。

    3.5 PATCH:用於將部分資料進行更新作業。

4. 傳入之資料需要使用 JSON.stringify() 將物件序列化成 JSON 字串。

4.4 專案資料夾架構

圖二 前端資料夾架構


Server Tree
server
package-lock.json
index.js
package.json
controllers
UserController.js
ProposalController.js
CommentController.js
AdminController.js
ChatBotController.js
PaymentController.js
routes
UserRoute.js
ProposalRoute.js
CommentRoute.js
AdminRoute.js
ChatBotRoute.js
PaymentRoute.js
config
validation.js
upload.js
database.js
mock
proposals.json
users.json
models
UserModel.js
ProposalModel.js
middleware
auth-middleware.js
app.yaml

圖三 後端資料夾架構

4.5 Route 列表

以下表格為所有頁面之 Route 列表並依照 Index 順序逐項進行功能說明，由於本專案之範例僅實作後台管理者之會員模組，此 Route 之規劃可依照所實作之功能複雜度與結果進行描述。列表如下:

Index	Route	Action	網址參數	功能描述、作法
1	api/admin/proposal/review	POST		管理員審核提案
2	api/admin/proposal/delete/	DELETE		管理員刪除提案
3	api/admin/users	GET		管理員檢視所有會員資訊
4	api/admin/user/delete/	DELETE		管理員刪除會員
5	api/admin/user/password-reset/	PATCH		管理員修改會員密碼
6	api/auth/register	POST		會員註冊
7	api/auth/login	POST		會員登入
8	api/user/profile	GET		檢視會員資訊
9	api/user/profile/update	PATCH		會員修改資訊
10	api/user/password/reset	PATCH		修改密碼
11	api/proposals/all	GET		瀏覽提案
12	api/proposals/category/:category	GET	category	以分類搜尋提案
13	api/proposals/search/:title	GET	title	搜尋提案

14	api/proposal	POST		新增提案
15	api/proposal/	PATCH		修改提案
16	api/proposal/	DELETE		刪除提案
17	api/proposal/info	GET		檢視提案
18	api/checkout/payment	POST		購買提案
19	api/checkout/payment/credit	POST		信用卡付款
20	api/checkout/payment/atm	POST		ATM 付款
21	api/purchase/history	GET		檢視購買紀錄
22	api/purchase/refund	POST		新增退貨提案
23	api/chatbot/search/:id	POST	id	使用 ChatBot 尋找提案
24	api/chatbot/help	GET		操作說明
25	api/chatbot/recommendations	GET		使用推薦系統
26	api/chatbot/proposal-info	GET		獲取提案資訊
27	api/comment	POST		新增評論
28	api/comment/	PATCH		修改評論
29	api/comment/	DELETE		刪除評論
30	api/comment/all	GET		檢視所有評論

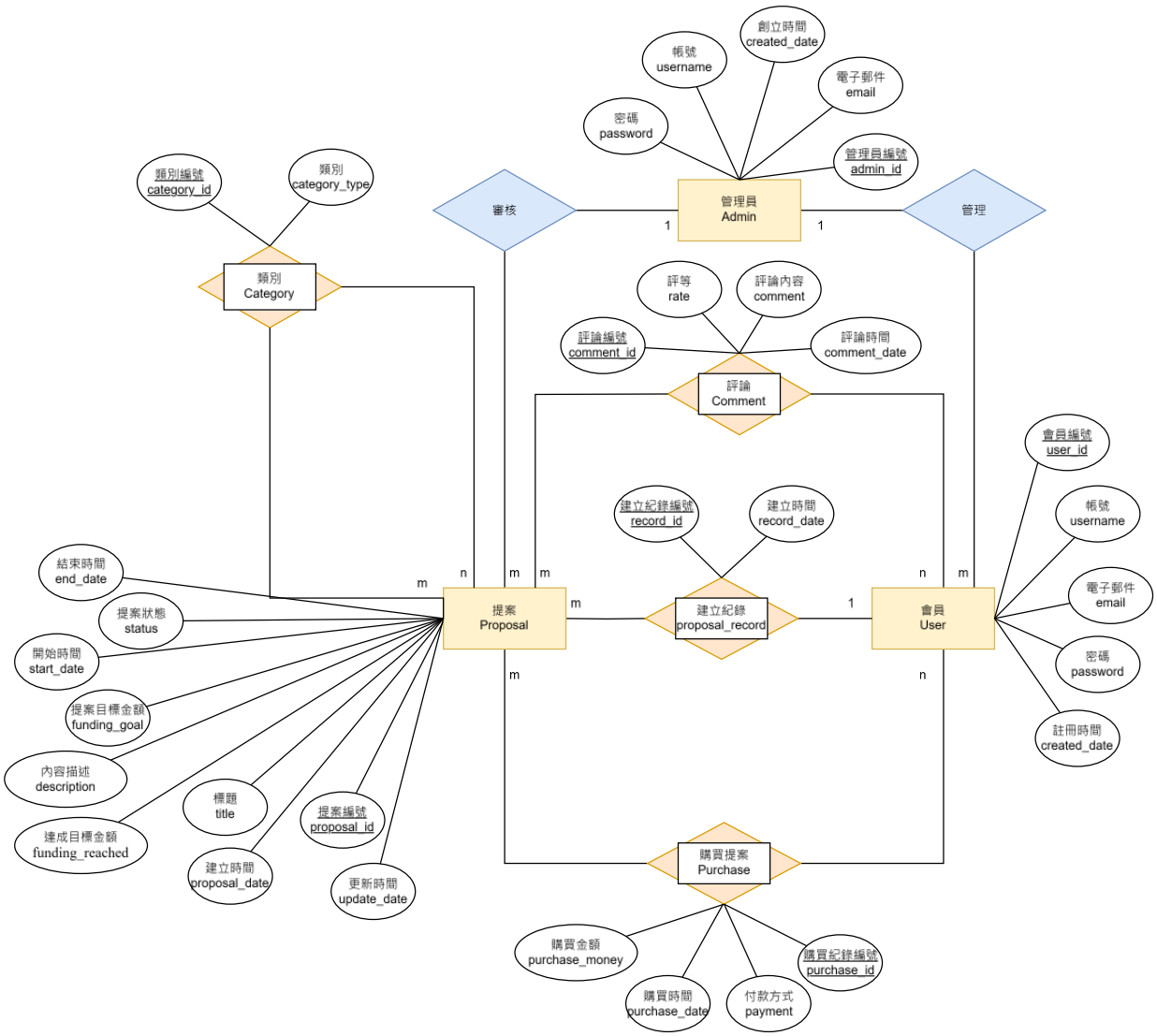
4.6 程式碼版本控制

本專案為達到追蹤程式碼開發之過程，並確保不同人所編輯之同一程式檔案能得到同步，因此採用分散式版本控制(distributed revision control)之軟體 Git，同時為避免維護程式碼檔案之問題，因此採用程式碼託管平台(GitHub) 作為本專案之使用。

本專案於 GitHub 上採用公開程式碼倉庫進行軟體開發，GitHub 允許註冊與非註冊用戶進行瀏覽，並可隨時隨地將專案進行 fork 或是直接 clone 專案進行維護作業，同時本專案僅限執行人員可以進行發送 push 之請求，最後說明文件 readme 檔案採用 markdown 格式進行編輯，本專案之 GitHub 網址為: https://github.com/CoCo-CrowdFunding。

第 5 章 資料庫設計

設計階段之資料庫，根據分析文件之實體關係圖，進行確認並依據其規劃資料庫之資料表，共計包含 3 個實體 (Entity)、2 個關係(Relationship)、4 個依賴（Depandance），下圖(圖四)為設計階段之 ERD 圖：



圖四 ERD 圖

由於此次專案所使用資料庫為「非關聯式資料庫」，自動新增欄位也更改成自動生成。以下將逐一說明資料庫每張資料表之欄位，如下表所示：

1. 管理員資料表（Admin）

key	Attribute	Type	自動生成
PK	admin_id	int	Y
	username	string	
	email	string (email format)	
	password	string	
	created_at	date	Y

2. 會員資料表（User）

key	Attribute	Type	自動生成	備註
PK	user_id	int	Y	
	username	string		
	email	string (email format)		
	password	string		
	created_at	date	Y	
	purchases_record	array of objects		購買提案紀錄
	proposal_id	int	Y	
	proposal_title	string		
	purchase_date	date	Y	
	purchase_money	int		
	present_record	array of objects		自己提案紀錄
	proposal_id	int	Y	
	proposal_title	string		
	purchase_date	date	Y	

3. 提案資料表（Proposal）

key	Attribute	Type	自動生成	備註
PK	proposal_id	int	Y	
	establish_user_id	int	Y	
	title	string		
	description	string		
	funding_goal	int		
	current_total_amount	int		
	start_date	date		
	end_date	date		
	status	int (enum: 0, 1, 2)		0 // 審核通過上架中 1 // 審核中 2 // 審核未通過
	funding_reached	int (enum: 0, 1)		0 // 尚未達到目標資金 1 // 已達到目標資金

4. 分類資料表（Category）

key	Attribute	Type	自動生成
PK	category_id	int	Y
	name	string	

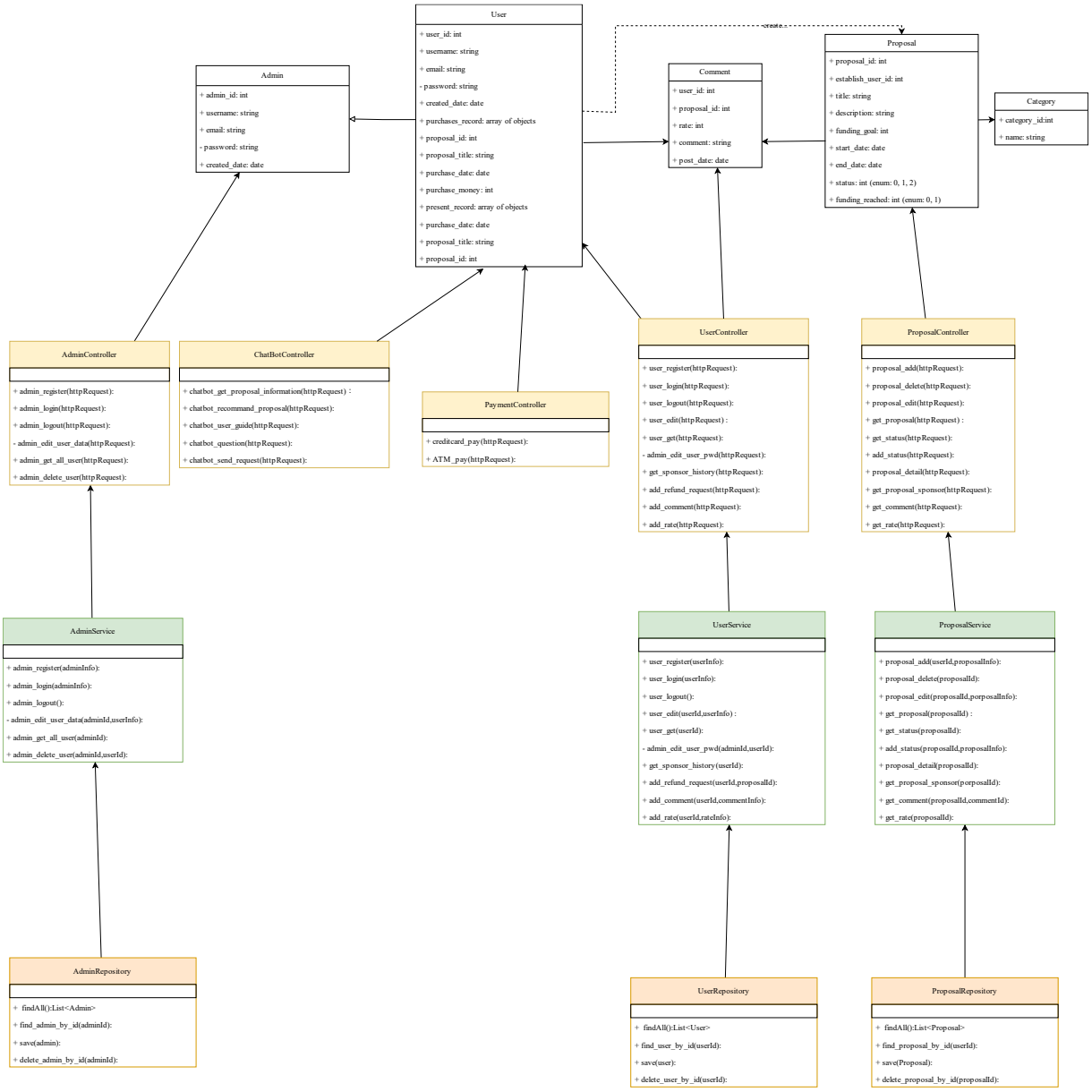
5. 評論資料表（Comment）

key	Attribute	Type	自動生成
PK	comment_id	int	Y
	user_id	int	Y
	proposal_id	int	Y
	rate	int	
	comment	string	
	post_date	date	Y

第 6 章 類別圖

下圖(圖五)係依據募資平台系統的實體關係圖(Entity-Relation Diagram)所繪製之設計階段之類別圖(Class Diagram)，用於描述系統的類別集合，包含其中之屬性，與類別之間的關係。

本階段之類別圖屬於細部(detail)之設計圖，與上一份文件分析階段之類別圖需要有詳細之變數型態、所擁有之方法，依據這些設計原則，本類別圖之說明如下所列：本專案使用之資料庫為非關聯式資料庫，類別圖除包含與資料庫相對應之物件外，亦包含相關控制物件(controller)。

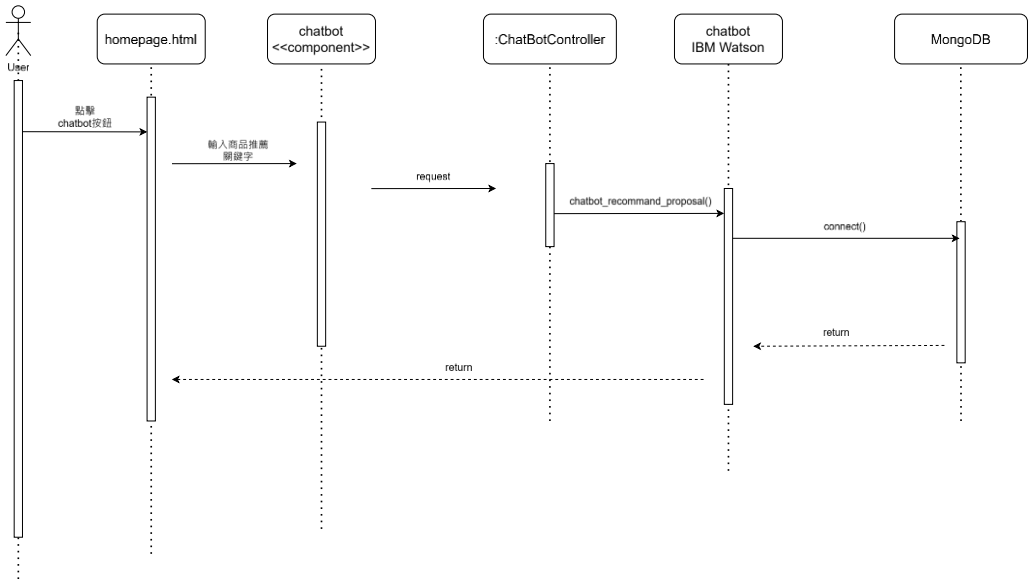

User
+ user_id: int
+ username: string
+ email: string
- password: string
+ created_date: date
+ purchases_record: array of objects
+ proposal_id: int
+ proposal_title: string
+ purchase_date: date
+ purchase_money: int
+ present_record: array of objects
+ purchase_date: date
+ proposal_title: string
+ proposal_id: int
create
Admin
+ admin_id: int
+ username: string
+ email: string
- password: string
+ created_date: date
Comment
+ user_id: int
+ proposal_id: int
+ rate: int
+ comment: string
+ post_date: date
Proposal
+ proposal_id: int
+ establish_user_id: int
+ title: string
+ description: string
+ funding_goal: int
+ start_date: date
+ end_date: date
+ status: int (enum: 0, 1, 2)
+ funding_reached: int (enum: 0, 1)
Category
+ category_id:int
+ name: string
AdminController
+ admin_register(httpRequest):
+ admin_login(httpRequest):
+ admin_logout(httpRequest):
- admin_edit_user_data(httpRequest):
+ admin_get_all_user(httpRequest):
+ admin_delete_user(httpRequest):
ChatBotController
+ chatbot_get_proposal_information(httpRequest) :
+ chatbot_recommand_proposal(httpRequest):
+ chatbot_user_guide(httpRequest):
+ chatbot_question(httpRequest):
+ chatbot_send_request(httpRequest):
PaymentController
+ creditcard_pay(httpRequest):
+ ATM_pay(httpRequest):
UserController
+ user_register(httpRequest):
+ user_login(httpRequest):
+ user_logout(httpRequest):
+ user_edit(httpRequest) :
+ user_get(httpRequest):
- admin_edit_user_pwd(httpRequest):
+ get_sponsor_history(httpRequest):
+ add_refund_request(httpRequest):
+ add_comment(httpRequest):
+ add_rate(httpRequest):
ProposalController
+ proposal_add(httpRequest):
+ proposal_delete(httpRequest):
+ proposal_edit(httpRequest):
+ get_proposal(httpRequest) :
+ get_status(httpRequest):
+ add_status(httpRequest):
+ proposal_detail(httpRequest):
+ get_proposal_sponsor(httpRequest):
+ get_comment(httpRequest):
+ get_rate(httpRequest):
AdminService
+ admin_register(adminInfo):
+ admin_login(adminInfo):
+ admin_logout():
- admin_edit_user_data(adminId,userInfo):
+ admin_get_all_user(adminId):
+ admin_delete_user(adminId,userId):
UserService
+ user_register(userInfo):
+ user_login(userInfo):
+ user_logout():
+ user_edit(userId,userInfo) :
+ user_get(userId):
- admin_edit_user_pwd(adminId,userId):
+ get_sponsor_history(userId):
+ add_refund_request(userId,proposalId):
+ add_comment(userId,commentInfo):
+ add_rate(userId,rateInfo):
ProposalService
+ proposal_add(userId,proposalInfo):
+ proposal_delete(proposalId):
+ proposal_edit(proposalId,porposalInfo):
+ get_proposal(proposalId) :
+ get_status(proposalId):
+ add_status(proposalId,proposalInfo):
+ proposal_detail(proposalId):
+ get_proposal_sponsor(porposalId):
+ get_comment(proposalId,commentId):
+ get_rate(proposalId):
AdminRepository
+ findAll():List<Admin>
+ find_admin_by_id(adminId):
+ save(admin):
+ delete_admin_by_id(adminId):
UserRepository
+ findAll():List<User>
+ find_user_by_id(userId):
+ save(user):
+ delete_user_by_id(userId):
ProposalRepository
+ findAll():List<Proposal>
+ find_proposal_by_id(userId):
+ save(Proposal):
+ delete_proposal_by_id(proposalId):

圖五 類別圖

第 7 章 系統循序圖

7.1 Use Case 實作之循序圖

商業流程編號 7.3 使用推薦系統



1. **使用者啟動聊天機器人**：

   使用者點擊聊天機器人按鈕，以啟動聊天機器人功能。

2. **使用者輸入關鍵字**：

   在 chatbot 視窗中，使用者可以輸入欲查詢提案的關鍵字。

3. **請求發送至 ChatBotController**：

   chatbot 組件收到使用者的輸入後，將此請求傳至後端 ChatBotController 請求推薦相關的提案。

4. **呼叫 IBM Watson 聊天機器人服務**：

   ChatBotController 收到請求，呼叫 chatbot_recommand_proposal() 方法，並將使用者輸入的關鍵字傳送至 IBM Watson 聊天機器人服務，以便取得推薦的提案。

5. **聊天機器人服務存取資料庫**：

   IBM Watson 聊天機器人會連接到 MongoDB 資料庫，以根據使用者的輸入來查詢相關的提案資料。

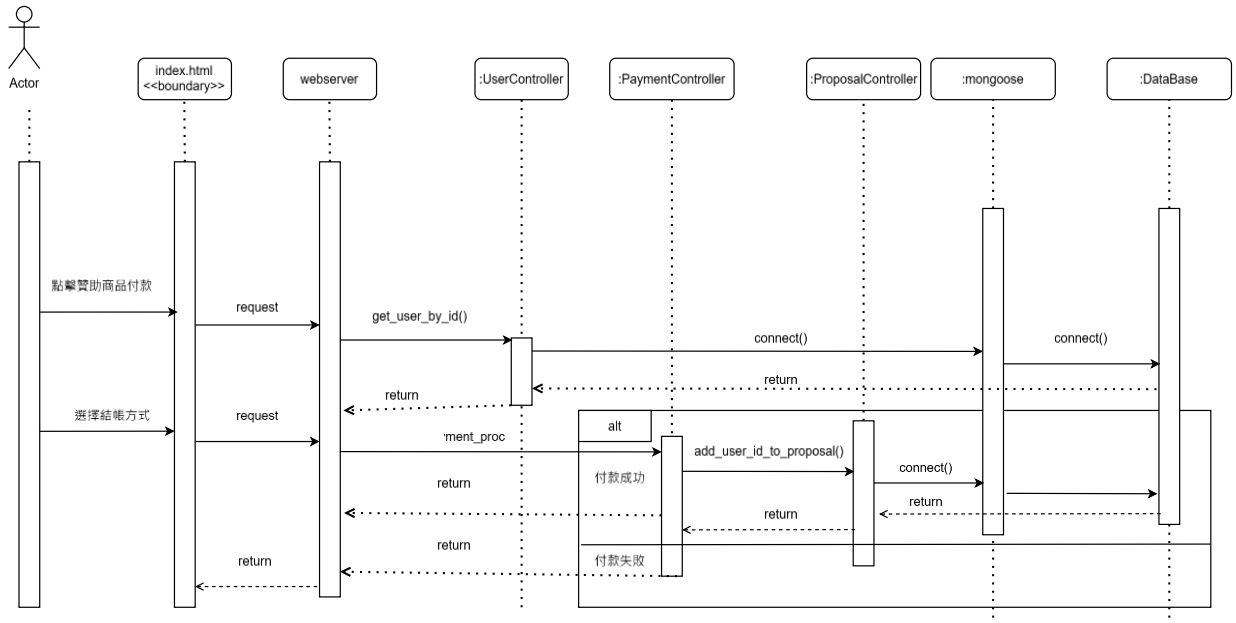
6. **返回推薦結果**：

MongoDB 回傳查詢結果至 IBM Watson 聊天機器人服務。IBM Watson 服務將推薦的提案結果傳回給 ChatBotController。

7. **將推薦結果顯示給使用者：**

ChatBotController 將推薦結果回傳至 chatbot 組件，並在 homepage.html 上顯示給使用者，讓使用者可以查看推薦的提案內容。

商業流程編號 5.1 贊助付款



1. **使用者送出付款請求：**

使用者點擊「贊助商品付款」按鈕發起付款請求，而請求會被傳送到伺服器。

2.**前端以 index.html 的 payment 畫面供使用者選擇付款方式：**

使用者進入付款選擇畫面，在這個畫面中可以選擇不同的付款方式，如信用卡或 ATM。

3. **伺服器向 UserController 發送 get_user_by_id() 請求：**

   webserver 會向 UserController 發送 get_user_by_id() 請求，藉此確認並獲取用戶的基本資料。UserController 會從資料庫中取得使用者資料，並回傳給 webserver。

4. **伺服器進行支付流程：**

   webserver 透過 PaymentController 中的 payment_process() 方法來處理支付，並將付款資料傳遞至後端進行支付驗證和處理。

5. **若支付成功，將用戶與贊助提案建立關聯：**

   若付款成功，PaymentController 會呼叫 ProposalController 的 add_user_id_to_proposal() 方法，將用戶的 ID 連結到相應的提案，表示該使用者成功贊助該提案。ProposalController 透過 mongoose 連接到資料庫，並將用戶 ID 加入到提案中，表示贊助紀錄。

6. **支付失敗處理：**

   若支付過程中出現問題，流程會進入「支付失敗」分支，webserver 會接收到失敗信息，並將失敗信息返回給前端，通知用戶支付未完成。

7. **返回結果到前端：**

   返回支付結果，供使用者檢視支付成功或失敗。