

# 國立中央大學

## 資訊管理學系

### 111 (一) 系統分析與設計

#### 系統軟體設計規格書

#### 第九組

資管三 A 109403019 鄒翔宇

資管三 A 109403022 劉宗翰

企管四 A 107401511 何羽軒

資管三 B 109403530 陳俐吟

資管三 B 109403035 張羽慈

資管三 B 109403529 陳侑宣

經濟三 108409532 林喬楚

指導教授：葉羅堯 教授

中華民國 1 1 1 年 1 月 6 日

# 目錄

目錄.....	i
表目錄.....	iii
圖目錄.....	iv
版本修訂.....	vi
第 1 章 簡介.....	1
1.1 文件目的 .....	1
1.2 系統範圍 .....	1
1.3 參考文件 .....	1
1.4 文件架構 .....	2
第 2 章 資料庫設計.....	3
第 3 章 類別圖.....	5
第 4 章 系統循序圖.....	7
4.1 使用案例圖 .....	7
4.2 Use Case實作之循序圖 .....	9
4.2.1 商業流程編號4.1：課程結帳.....	9
4.2.2 商業流程編號10.1：挑戰課程.....	10
第 5 章 系統開發環境.....	11
5.1 環境需求 .....	11
5.1.1 伺服器硬體.....	11
5.1.2 伺服器軟體.....	11
5.1.3 前端套件（可於./client/package.json找到，僅列較重要者）. 12	
5.1.4 後端套件（可於./server/package.json找到） .....	12

5.1.5 客戶端使用環境.....	13
5.2 專案架構.....	13
5.2.1 系統架構圖.....	13
5.2.2 MVC 架構.....	14
5.2.2.1 顯示層 (Presentation Layer): MVC-View .....	14
5.2.2.2 商業邏輯層 (Business Logic Layer) .....	14
5.2.2.3 資料層 (Data Layer) .....	16
5.3 部署 .....	18
第 6 章 專案撰寫風格.....	20
6.1 程式命名風格 .....	20
6.2 回傳訊息規範 .....	21
6.3 API 規範 .....	22
6.4 專案資料夾架構 .....	23
6.5 Route 列表 .....	27
6.6 程式碼版本控制 .....	29
第 7 章 專案程式設計.....	30
7.1 JSON .....	31
7.1.1 JSON 格式介紹.....	31
7.1.2 前端發送 AJAX Request 說明.....	31
7.1.3 前端表格 Render 與欄位回填.....	34
7.2 MongoDB 與 Mongoose .....	35

## 表目錄

表 1：會員資料表 (Member) 之資料結構.....	4
表 2：課程資料表 (Course) 之資料結構.....	4
表 3：CarryU 專案前端套件表.....	12
表 4：CarryU 專案後端套件表.....	12
表 5：CarryU Route 列表.....	27

## 圖目錄

圖 1：設計階段之實體關係圖.....	3
圖 2：後端類別圖.....	5
圖 3：後端類別圖（工具生成）.....	6
圖 4：前端類別圖（工具生成）.....	6
圖 5：使用者案例圖.....	8
圖 6：商業流程編號 4.1 課程結帳循序圖.....	9
圖 7：商業流程編號 10.1 挑戰課程循序圖.....	10
圖 8：CarryU專案系統架構圖.....	13
圖 9：ExpressJS 之 Controller 範例模板1（以 challenge-controller.js 為例）.....	15
圖 10：ExpressJS 之 Controller 範例模板2（以 challenge-controller.js 為例）.....	16
圖 11：使用mongoose的功能進行連線與其設定.....	17
圖 12：將其引入./server/index.js之中使用connectDB().....	17
圖 13：資料庫參數.....	18
圖 14：於.env中儲存資料庫敏感資訊之參數.....	18
圖 15：專案部署圖.....	19
圖 16：course-controller 之 GET 取得回傳之資料格式範例.....	22
圖 17：CarryU專案簡易資料夾結構.....	24
圖 18：CarryU專案前端資料夾結構.....	25
圖 19：CarryU專案後端資料夾結構.....	26
圖 20：後端import範例圖.....	30
圖 21：前端import範例圖.....	30

圖 22：JSON格式範例圖.....	31
圖 23：後端使用Joi製作registerValidation.....	32
圖 24：於user-controller.js 中導入並使用registerValidation驗證資料.	32
圖 25：前端呼叫API後，出現錯誤時react-toastify會顯示錯誤通知.....	32
圖 26：前端使用axios呼叫API.....	33
圖 27：更新會員資料以json格式回傳.....	34
圖 28：更新會員資料後取得更新後的資料.....	34
圖 29：取得更新後的資料進行刷新.....	35
圖 30：連接資料庫之檔案（節錄）.....	36
圖 31：新增點數之updateOne method（節錄）.....	36
圖 32：userSchema（節錄）.....	36

## 版本修訂

版本	修訂者	修訂簡述	日期
v0.0.1	何羽軒	Draft	2022/12/26
v0.1.1	何羽軒	5、6、7章更新 更新各章圖表	2023/1/6

# 第 1 章 簡介

軟體設計規格書 (software design description, SDD) 係依據軟體產品之主要使用者之需求規格文件 (software requirements specification, SR S)、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課程平台，其中主要包含管理員管理、課程資訊、訂購課程、購買紀錄、退貨管理、課程管理、課程推薦、挑戰課程、學習等模組，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課程平台所需的管理流程。

## 1.3 參考文件

1. [系統分析與設計—需求 \(Requirement\)](#)
2. [系統分析與設計—分析 \(Analysis\)](#)
3. [CarryU README](#)



## 1.4 文件架構

1. 第二章撰寫設計階段之資料庫架構 ER 圖，包含資料表之元素與特殊要求。
2. 第三章描述設計階段之類別圖，包含細部之屬性與方法。
3. 第四章講述每個use case之細部循序圖，以供實作階段使用。
4. 第五章闡述專案之開發環境與系統架構和部署方法。
5. 第六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之用。
6. 第七章說明本專案之程式設計特殊與獨特之處，並說明其緣由。

## 第 2 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 (Entity-Relation Diagram，進行確認並依據其規劃資料庫之資料表，共計包含 5 個實體 (Entity)、5 個關係 (Relationship)、0 個複合性實體 (Compound Entity)，下圖 (圖 1) 為設計階段之 ER 圖，亦可使用資料庫綱要圖 (Schema Diagram)：

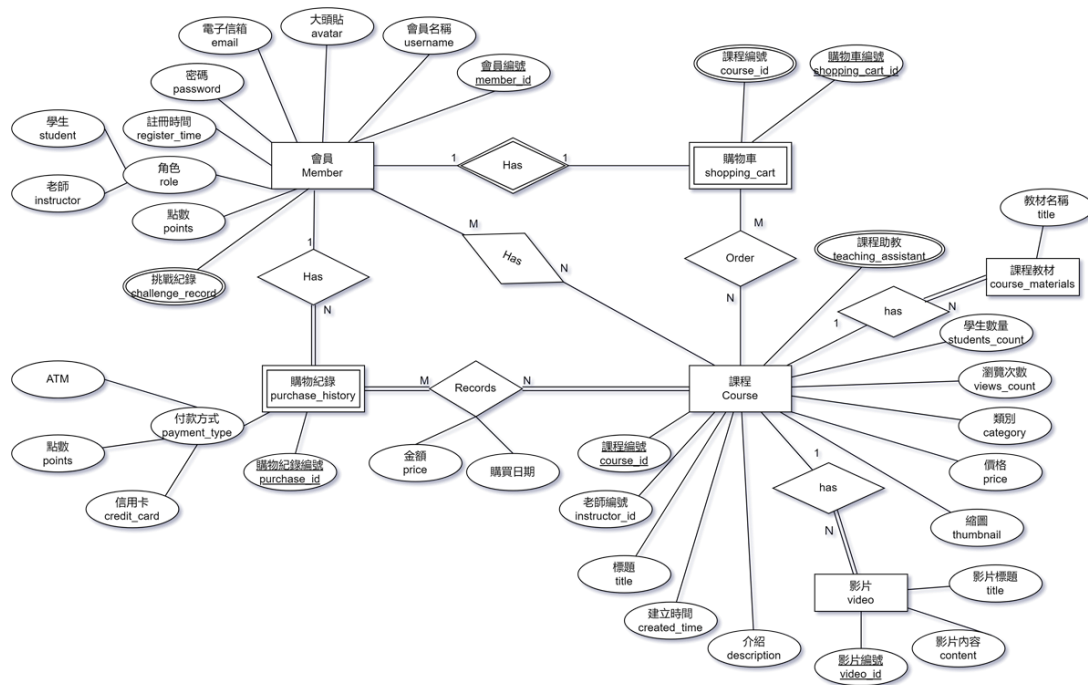


圖 1：設計階段之實體關係圖

根據上圖進行資料表之設計，以下將逐一說明資料庫每張資料表之欄位，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，因此資料表如下表 (表 1、表 2) 所示：

# 1. 會員資料表 (Member)：

表 1：會員資料表 (Member) 之資料結構

<b>Member(會員資料表)</b>					
Key	名稱	類型	預設值	空值	自動生成
Pk	member_id	String	N	N	Y
	username	String	N	Y	N
	avatar	BLOB	N	N	N
	email	String	N	Y	N
	password	String	N	Y	N
	register_time	Date	Y	Y	Y
	role	String	Y	N	N
	shopping_cart	Array	N	N	N
	course_id	String	N	N	Y
	purchase_history	Array			
	purchase_record				
	course_id	String	N	N	Y
	course_name	String	N	Y	N
	price	int	N	N	N
	payment_type	String	N	Y	N
	purchase_date	Date	N	Y	N
	points	int	Y	N	N
	challenge_history	Array			
	challenge_record				
	course_id	String	N	N	Y
	challenge_date	Date	N	Y	N
	status	String	N	Y	N

# 2. 課程資料表 (Course)：

表 2：課程資料表 (Course) 之資料結構

<b>Course(課程資料表)</b>					
Key	名稱	類型	預設值	空值	自動生成
PK	course_id	String	N	N	Y
	instructor	String	N	N	Y
	title	String	N	N	N
	description	String	N	Y	N
	thumbnail	BLOB	N	Y	N
	price	int	N	N	N
	category	String	N	Y	N
	views_count	int	Y	Y	N
	students_count	int	Y	Y	N
	students	Array	N	Y	N
	comments	Array	N	Y	N
	videos	Array	N	Y	N
	video				
	content	BLOB	N	Y	N
	title	String	N	Y	N
	teaching_assistant	Array	N	N	N
	member_id	String	N	N	Y
	created_time	Date	Y	N	N

### 第 3 章 類別圖

下圖（圖 2、圖 3、圖 4）係依據線上課程訂購系統的分析模型、建立的互動圖、以及實體關係圖（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：CarryU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，亦包含相關之模型（model）及控制物件（controller），由於前後端使用類別過於繁雜，使用繪製類別方法及工具生成兩種方式繪製。

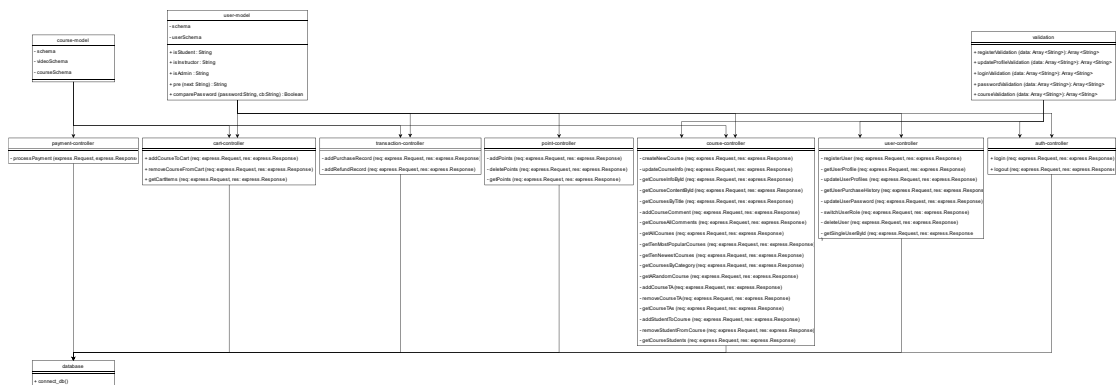


圖 2：後端類別圖

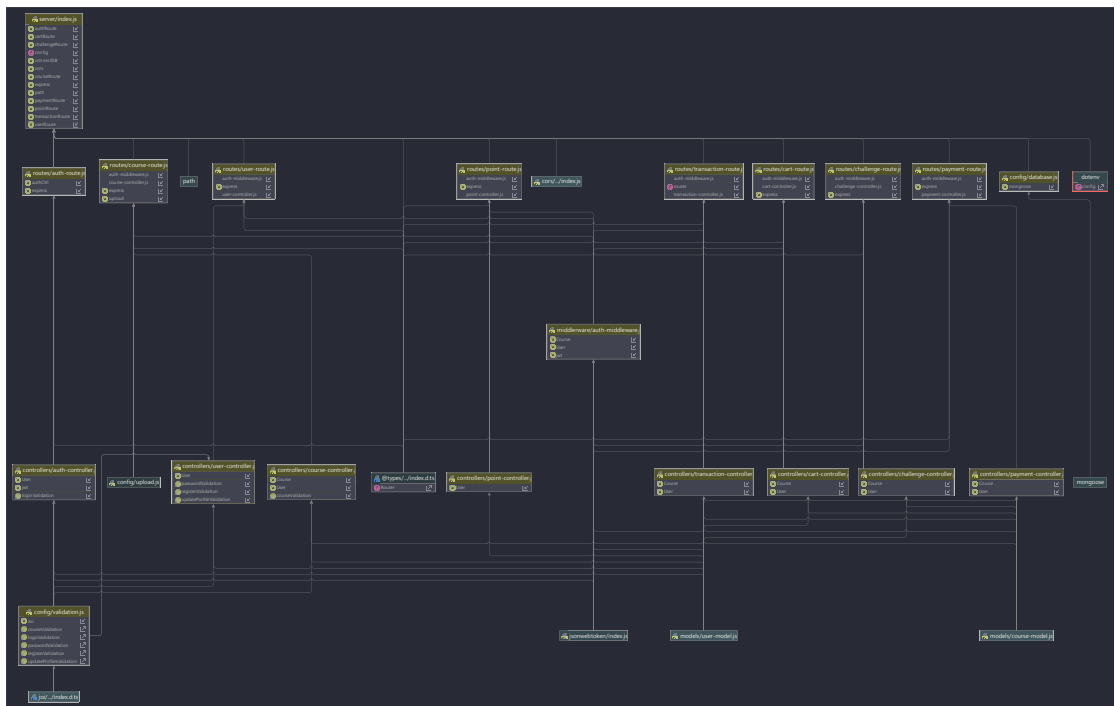


圖 3：後端類別圖（工具生成）

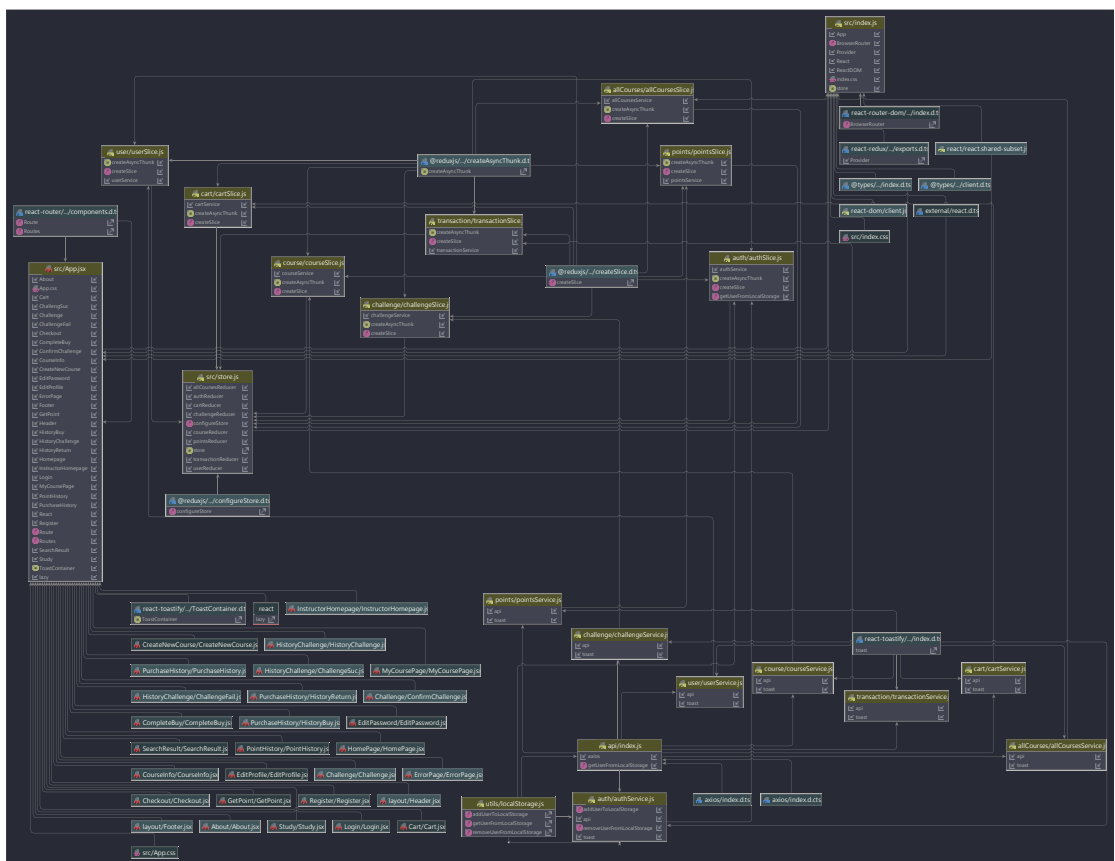


圖 4：前端類別圖（工具生成）

## 第 4 章 系統循序圖

本章節主要依照第一份文件需求所產生之使用案例圖 (use case) 與第二份文件分析之邏輯階段活動圖與強韌圖為基礎，進行設計階段之循序圖設計，將每個使用案例進行闡述。於此階段，需要有明確之類別 (class) 名稱與呼叫之方法 (method) 與傳入之變數名稱與型態等細部設計之內容。

### 4.1 使用案例圖

依據第一份文件針對專案之需求進行確定 (Software Requirement Specification)，本線上課程平台系統預計共有 6 位動作者與 49 個使用案例，並依照不同之模組區分成不同子系統，共計十一個子系統，依序為①會員子系統、②課程資訊子系統、③購物車子系統、④課程結帳子系統、⑤購買紀錄子系統、⑥退貨管理子系統、⑦課程管理者子系統、⑧管理員管理子系統、⑨課程推薦子系統、⑩挑戰課程子系統、⑪學習子系統，如下圖 (圖 5) 所示。

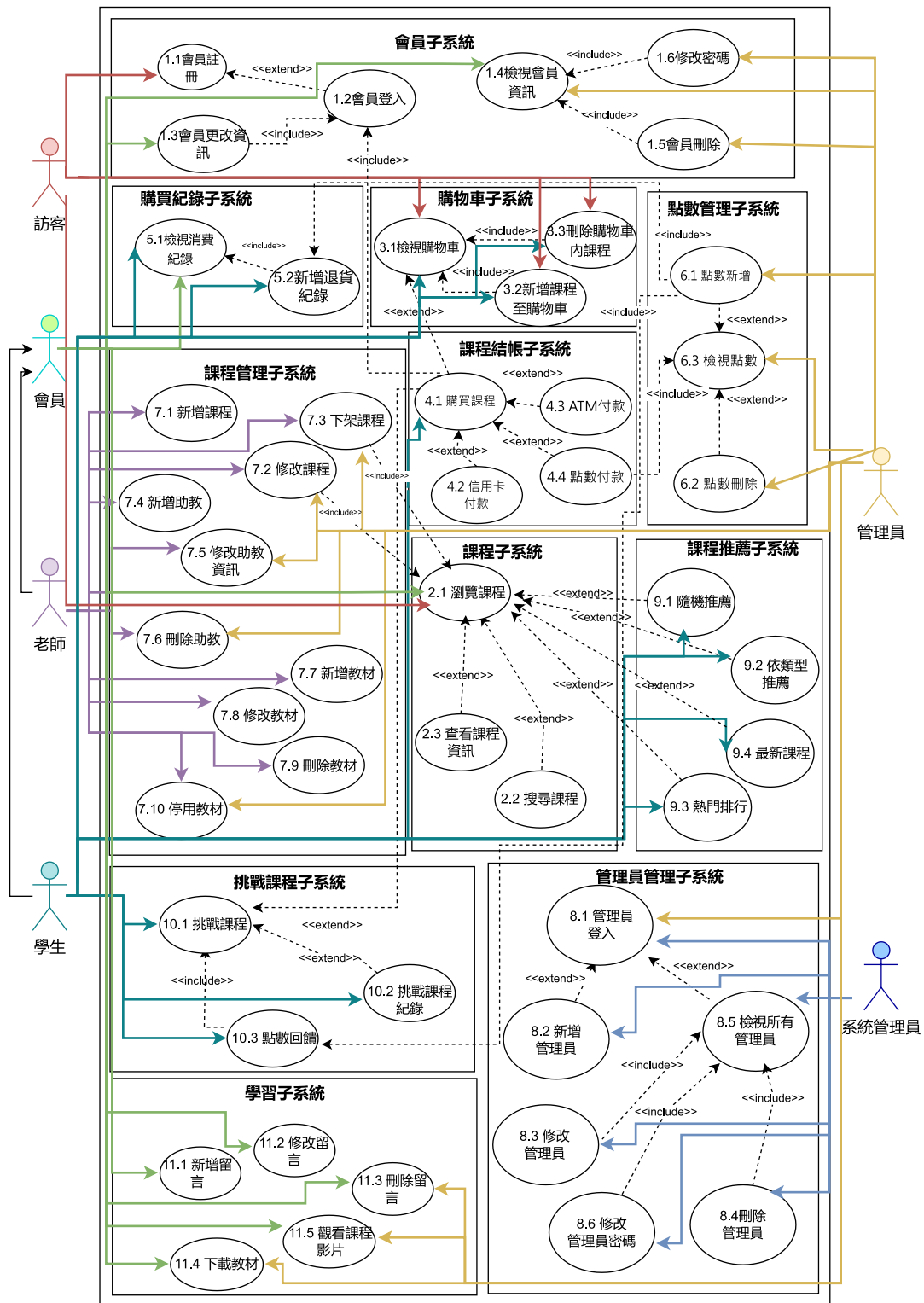


圖 5：使用者案例圖

## 4.2 Use Case實作之循序圖

### 4.2.1 商業流程編號4.1：課程結帳

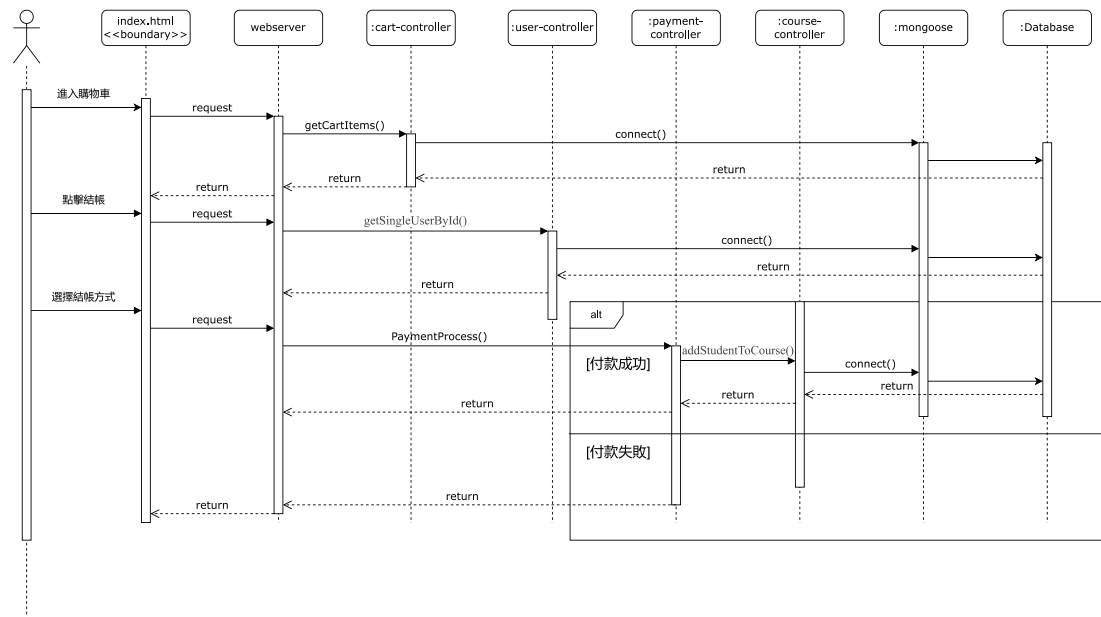


圖 6：商業流程編號 4.1 課程結帳循序圖

1. 使用者送出檢視購物車請求。
2. 後端以cart-controller之getCartItems()進行處理，進入資料庫中取出購物車中的物件資訊，回傳給前端供使用者檢視。
3. 使用者送出結帳請求，後端以payment-controller之PaymentProcess()處理，若付款成功則使用course-controller的addStudentToCourse()更新資料表，將使用者加入課程的student中。
4. 若付款失敗則直接回傳付款失敗之訊息。
5. 回傳結帳成功、加入課程之訊息。



## 4.2.2 商業流程編號10.1：挑戰課程

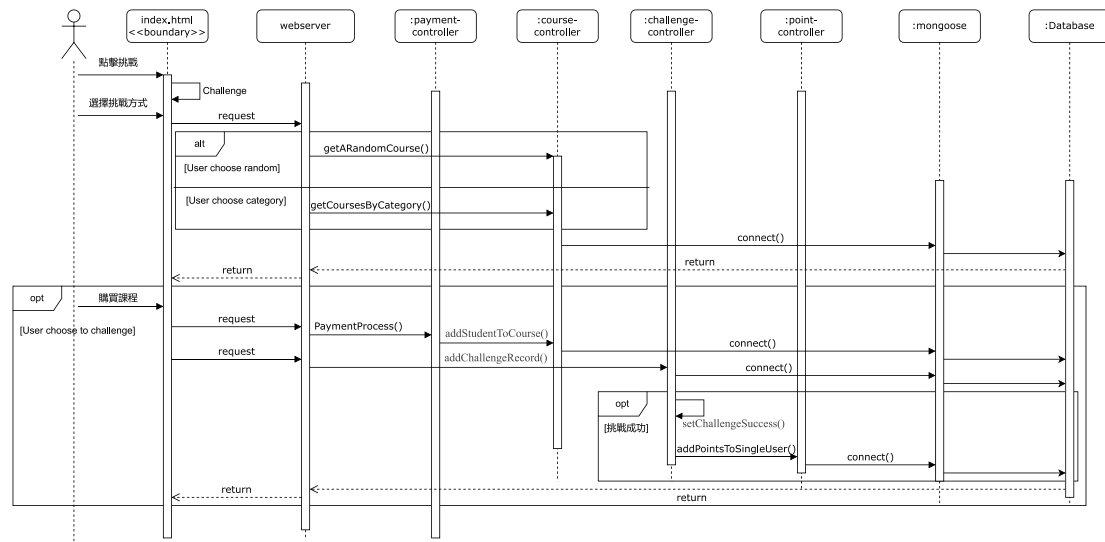


圖 7：商業流程編號 10.1 挑戰課程循序圖

1. 使用者送出挑戰課程請求。
2. 前端以index.html之challenge.jsx供使用者選擇要隨機挑戰或是依類別挑戰。
3. 若使用者選擇隨機挑戰則以後端course-controller的getARandomCourse()隨機提供使用者課程，若選擇依類別則使用course-controller的getCoursesByCategory()提供使用者依類別選擇要挑戰的課程。
4. 後端以course-controller的getCourseContentById()從資料庫中取出課程資料，並且前端將使用者導向該課程結帳介面供使用者檢視課程資料及決定是否挑戰。
5. 若使用者選擇挑戰則使用payment-controller進行結帳，並且由後端challenge-controller的addChallengeRecord()新增挑戰紀錄。
6. 若挑戰成功則使用challenge-controller的addChallengeSuccess()新增挑戰成功紀錄，並透過point-controller之addPointsToSingleUser()為使用者新增點數。

## 第 5 章 系統開發環境

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。

### 5.1 環境需求

#### 5.1.1 伺服器硬體

本專案預計部署之設備如下：

1. OS：Ubuntu 20.04 LTS
2. CPU：e2-medium（2 個 vCPU，4 GB 記憶體）
3. RAM：4 GB

最終部署使用GCP中的App Engine，以較高的費用省去管理伺服器資源的時間。

#### 5.1.2 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

### 5.1.3 前端套件（可於./client/package.json找到，僅列較重要者）

表 3：CarryU專案前端套件表

套件名稱	版本	簡述
react	5.1.0	打造使用者頁面的JavaScript Library
react-router-dom	6.4.5	負責設定前端路由的套件
react-icons	4.7.1	提供許多icon並且為向量圖示
react-redux	8.0.5	管理狀態（state）的套件
@reduxjs/toolkit	1.9.1	redux工具包，簡化使用redux的工具
tailwindcss	2.8.5	Utility-first的CSS框架
axios	16.0.3	Promise based發送HTTP請求的工具
react-toastify	9.1.1	客製化添加應用程式中的通知

### 5.1.4 後端套件（可於./server/package.json找到）

表 4：CarryU專案後端套件表

套件名稱	版本	簡述
bcrypt	5.1.0	用於加密使用者密碼
cors	2.8.5	跨來源資源共用。讓單一往預的Web Application 可以與其它網域中的資源互動
dotenv	16.0.3	使程式能夠讀取所有環境變數
express	4.18.2	為nodejs打造的輕量web framework
joi	17.7.0	用於檢查驗證POST的資料
jsonwebtoken	8.5.1	簽發與驗證使用者token的工具
mongoose	6.7.2	MongoDB物件模型工具

### 5.1.5 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。

## 5.2 專案架構

### 5.2.1 系統架構圖

本專案系統整體架構如下圖（圖 8）所示，主要使用JavaScript 語言撰寫，詳細說明請參閱（5.2.2 MVC架構），以port 8080與用戶端（Client）相連。由於本專案之範例以後台管理者會員管理為範例，因此就現有之檔案進行繪製，實際圖檔仍須依照實作將所有物件繪製進行說明。

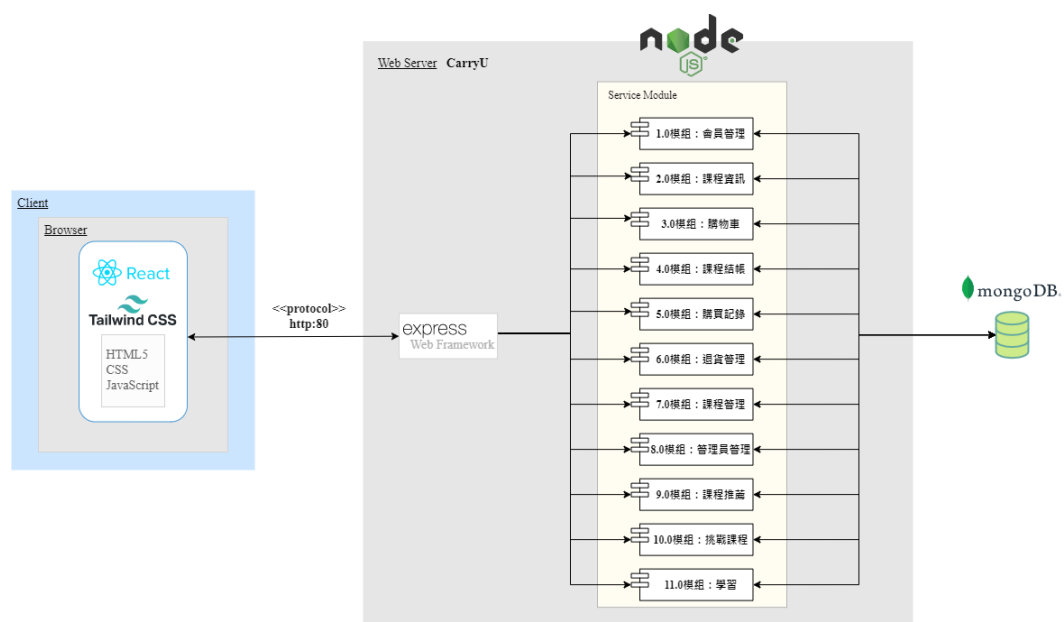


圖 8：CarryU專案系統架構圖

## 5.2.2 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 Node.js 之框架用於編寫動態互動式網站。

採用此架構可完全將前端與後端進行分離，其中中間溝通過程使用呼叫 API 方式進行，資料之格式定義為 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：

### 5.2.2.1 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 等網頁常用物件作為模板。
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。

### 5.2.2.2 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-Mode
  - 主要放置於「carryu/server/models/\*」資料夾當中，以 JavaScript 搭配 Mongoose 撰寫，有兩個 Model 於其中，course-model.js、user-model.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「carryu/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「carryu/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。
- 實做 Controller 應依照以下範例進行撰寫。
- 命名會以 \*-controller 進行命名，下圖 (圖 9) 顯示 Controller 之範本，每一個 function 上方都會註解此 function 要做的事情以及其 api 路徑。

```

1  const User = require("../models/user-model");
2
3  // @desc    Add challenge record
4  // @route    PUT /api/challenge/add
5  // @access   Private
6  exports.addChallengeRecord = async (req, res) => {
7      const { course_id } = req.body;
8      const status = "inprogress";
9
10     User.findById(req.user._id)
11       .then(async (user) => {
12         if (!user) {
13           return res.status(404).json({ err_msg: "User Not found" });
14         }
15
16         // Check if the course is already in the challenge history.
17         const courseInHistory = user.challenge_history.find(
18           (record) => record.course_id.toString() === course_id.toString()
19         );
20
21         if (courseInHistory) {
22           return res.status(400).json({
23             err_msg:
24               "You have already challenged this course! Please try another course.",
25           });
26         }
27
28         // Check if there's any course in the challenge history with status "inprogress".
29         const inProgressCourse = await user.challenge_history.find(
30           (record) => record.status === "inprogress"
31         );

```

圖 9：ExpressJS 之 Controller 範例模板1 (以 challenge-controller.js 為例)

- 呼叫 function 或是物件之路徑可直接於該其他 JavaScript 檔案當中指定 (例如：const {addChallengeRecord} = require("../controllers/ challenge-controller")), 並將其放置於該檔案程式碼最上方，以下圖 (圖 10) 為例為第1行，也可指定多路徑到此檔案中。

```

65 // @route PUT /api/challenge/set-to-success
66 // @access Private
67 exports.setChallengeToSuccess = async (req, res) => {
68   const { course_id } = req.body;
69
70   User.findById(req.user._id)
71     .then((user) => {
72       if (!user) {
73         return res.status(404).json({ err_msg: "User Not found" });
74       }
75
76       const courseInHistory = user.challenge_history.find(
77         (record) => record.course_id.toString() === course_id.toString()
78       );
79
80       if (!courseInHistory) {
81         return res.status(404).json({ err_msg: "Challenge record not found" });
82       }
83
84       if (courseInHistory.status === "success") {
85         return res.status(400).json({ err_msg: "Challenge already succeeded" });
86       }
87
88       courseInHistory.status = "success";
89       courseInHistory.completed_at = Date.now();

```

圖 10：ExpressJS 之 Controller 範例模板2（以 challenge-controller.js 為例）

ExpressJS提供許多http method之實作，詳請可查閱官方文件。

(<https://expressjs.com/en/api.html#app>)

### 5.2.2.3 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。如下圖（圖 11、圖12）所示：

```

server > config > js database.js > ...
You, 3 weeks ago | 1 author (You)
1  const mongoose = require("mongoose");
2
3  const dbName = "carryu";
4
5  const connectDB = () => {
6    console.log("Connecting to database...");
7
8    mongoose
9      .connect(process.env.MONGO_URI, {
10        useNewUrlParser: true,
11        useUnifiedTopology: true,
12        dbName,
13      })
14      .then(() =>
15        console.log(`MongoDB connected! Connected to database ${dbName}`)
16      )
17      .catch((err) => console.log(err));
18  };
19
20  module.exports = connectDB;

```

圖 11：使用mongoose的功能進行連線與其設定

```

server > js index.js > ...
xxrjun, 7 days ago | 3 authors (You and others)
1  const express = require("express");
2  require("dotenv").config();
3  const connectDB = require("../config/database.js");
4  const cors = require("cors");
5  const app = express();
6
7  app.use(cors());
8  const PORT = process.env.PORT || 3000;
9
10 // Import routes
11 const authRoute = require("../routes/auth-route.js");
12 const userRoute = require("../routes/user-route.js");
13 const courseRoute = require("../routes/course-route.js");
14 const cartRoute = require("../routes/cart-route.js");
15 const pointRoute = require("../routes/point-route.js");
16 const paymentRoute = require("../routes/payment-route.js");
17 const transactionRoute = require("../routes/transaction-route.js");
18 const challengeRoute = require("../routes/challenge-route.js");
19
20 // Connect to database
21 connectDB();

```

圖 12：將其引入./server/index.js之中使用connectDB()



本專案所使之資料庫參數可以直接於function中填入，敏感資訊如帳號密碼則是於.env檔案中進行設定，降低資料庫帳號密碼外洩的風險，如下圖（圖 14）所示：

- 其中必須填入資料庫連線的URI。
- 將useNewUrlParser設為true，允許使用者發現新解析氣中的錯誤時退回到舊解析器。
- 將useUnifiedTopology設為true，選擇使用MongoDB驅動程序的新連接管理器，以保持穩定的連接。
- 可以指定Database的名稱。

```
8      mongoose
9      .connect(process.env.MONGO_URI, {
10        useNewUrlParser: true,
11        useUnifiedTopology: true,
12        dbName,
13      })
```

圖 13：資料庫參數

```
server > .env
1  PORT=8080
2  MONGO_URI="mongodb+srv://admin:carryu-admin@cluster0.f6orekg.mongodb.net/?
  retryWrites=true&w=majority"
3  JWT_SECRET="CARRYU_SECRET"
```

圖 14：於.env中儲存資料庫敏感資訊之參數

## 5.3 部署

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撲架構，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）。本專案線上課程影音平台，使用 NodeJS 平台技術建構 Web 應用程式，其實際觀點模型的部署圖，如下圖（圖 15）所示：

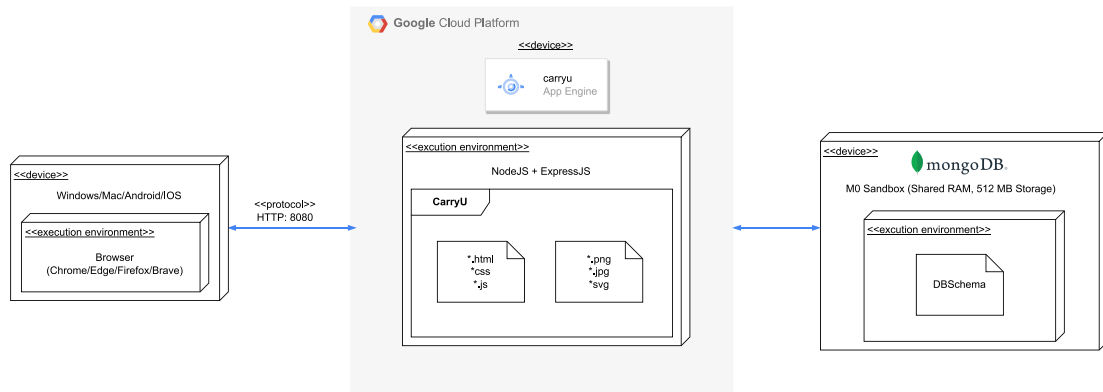


圖 15：專案部署圖

1. 本專案之部署方式，其硬體與軟體規格如前揭（5.1 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. 最終整個專案之檔案將分成前端與後端分別進行部署，前端匯出至「./client/build」並撰寫app.yaml進行部署；後端則撰寫完app.yaml直接部署。
4. 本專案之資料庫將使用MongoDB Atlas，並設定完成所需之帳號、密碼。
5. 本專案之網頁伺服器埠號採用預設之8080。
6. 客戶端（Client）僅需使用裝置上瀏覽器，藉 https 即可連上本專案網站。

## 第 6 章 專案撰寫風格

### 6.1 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。

搭配使用VSCode Extension中的[Prettier](#)進行程式碼自動排版與美化。

#### 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。例如：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「Kebab Case命名法」，以“-”來分隔單字，包含controller, route, middleware等等。

- controllers/：命名以\*-controller為主 (例如：user-controller.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)

4.1. 主要採用「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：getPassword()、updateMember()、getAllMembers()等。

## 6. 變數 (Variable)

5.1. 採用「snake case命名方式」，首字皆小寫以底線區隔。例如：purchase\_history 等。

## 6.2 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，相關之範例可參閱下圖 (圖 16) 所示。

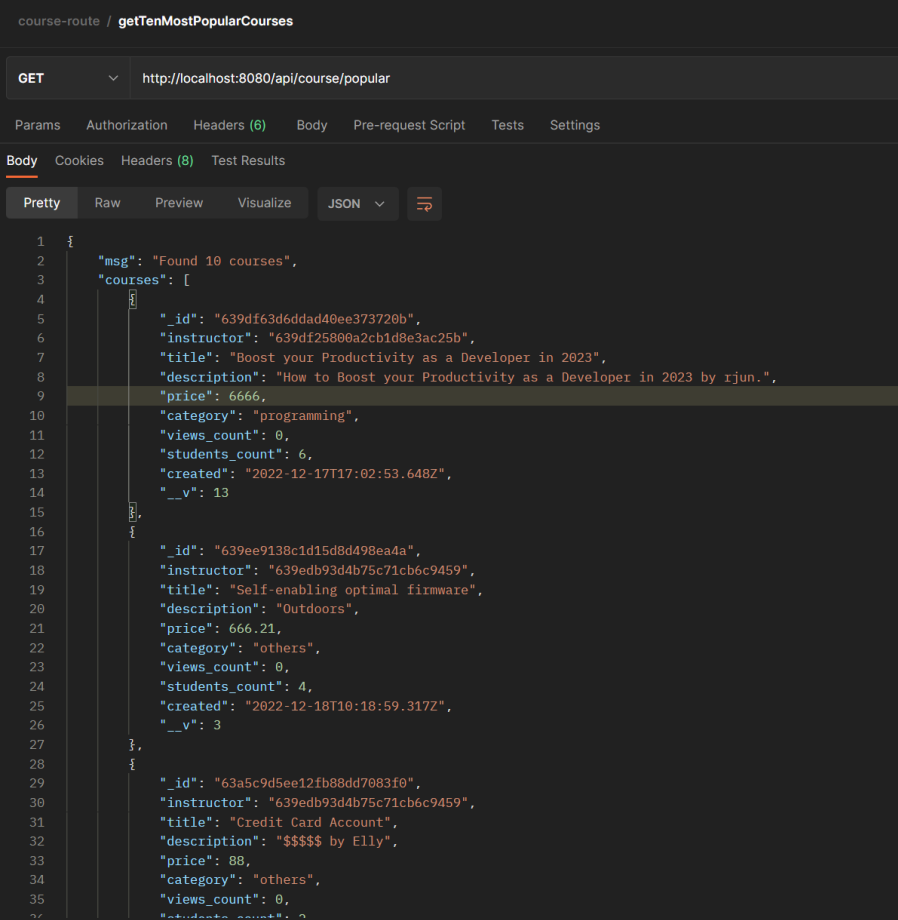
2.1. API 回傳資料之組成包含三個 KEY 部分，以下分別進行說明：

- status

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

- 200：正確回傳。
- 400：Bad Request Error，可能有需求值未傳入。
- 403：權限不足。
- 404：找不到該網頁路徑。
- 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。

- msg || message || err\_msg
  - 主要英文回傳所執行之動作結果。
  - 可用於後續渲染 (Render) 至前端畫面。
- res
  - 儲存另一個JSON格式物件，可跟隨所需資料擴充裡面的值。



```

course-route / getTenMostPopularCourses
GET http://localhost:8080/api/course/popular
Body Cookies Headers (8) Test Results
Pretty Raw Preview Visualize JSON
1 {
2   "msg": "Found 10 courses",
3   "courses": [
4     {
5       "_id": "639df63d6ddad48ee373720b",
6       "instructor": "639df25800a2cb1d8e3ac25b",
7       "title": "Boost your Productivity as a Developer in 2023",
8       "description": "How to Boost your Productivity as a Developer in 2023 by rjun.",
9       "price": 6666,
10      "category": "programming",
11      "views_count": 0,
12      "students_count": 6,
13      "created": "2022-12-17T17:02:53.648Z",
14      "__v": 13
15    },
16    {
17      "_id": "639ee9138c1d15d8d498ea4a",
18      "instructor": "639edb93d4b75c71cb6c9459",
19      "title": "Self-enabling optimal firmware",
20      "description": "Outdoors",
21      "price": 666.21,
22      "category": "others",
23      "views_count": 0,
24      "students_count": 4,
25      "created": "2022-12-18T10:18:59.317Z",
26      "__v": 3
27    },
28    {
29      "_id": "63a5c9d5ee12fb88dd7083f0",
30      "instructor": "639edb93d4b75c71cb6c9459",
31      "title": "Credit Card Account",
32      "description": "$$$$ by Elly",
33      "price": 88,
34      "category": "others",
35      "views_count": 0,
36      "students_count": 0
37    }
38  ]
39 }

```

圖 16：course-controller 之 GET 取得回傳之資料格式範例

## 6.3 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()` 將物件序列化成為 JSON 字串。

## 6.4 專案資料夾架構

下圖（圖 17）為本系統之專案資料夾整體架構：

- 1. 根目錄主要存放git 相關檔案和前端（client）之相關文件（包含JSX、HTML等），網站之靜態文件（image、icon等）則存放於assets資料夾當中。
- 2. server則存放controllers、models、config等後端檔案：
  - A. web.xml：存放網站之路徑（route）資料。
  - B. 網站上線時，須將所有前端檔案建置至./client/build，因為瀏覽器讀不懂JSX，因此所有JSX檔案最終都會變成JavaScript的檔案。故build中只會有靜態檔案、HTML、CSS、JavaScript檔案。
  - C. 只需要於前端執行「npm run build」即可執行建置動作。
- 3. docs 資料夾則是存放本專案所有相關文件，包含需求文件、分析文件、設計文件、GitHub基礎教學，archive放置舊的檔案。

# CarryU Simple Tree

```
carryu
├── README.md
├── client
├── server
├── images
└── docs
```

4 directories, 1 file

圖 17：CarryU專案簡易資料夾結構

## Client Tree

```

client
├── package-lock.json
├── README.md
├── package.json
├── tailwind.config.js
├── webpack.config.js
├── src
│   ├── App.jsx
│   ├── index.css
│   ├── App.css
│   ├── store.js
│   └── components
│       ├── layout
│       │   ├── Nav.jsx
│       │   ├── Dropdown.jsx
│       │   ├── Card.jsx
│       │   ├── Footer.jsx
│       │   ├── Search.jsx
│       │   ├── Input.jsx
│       │   ├── Header.jsx
│       │   ├── Button.jsx
│       │   ├── Loading.jsx
│       │   └── Title.jsx
│       ├── HistoryChallenge
│       │   ├── ChallengeNav.jsx
│       │   ├── ChallengeCard.jsx
│       │   ├── ChallengeFail.jsx
│       │   ├── ChallengeSuc.jsx
│       │   ├── ChallengeRecord.jsx
│       │   └── HistoryChallenge.jsx
│       ├── Study
│       │   ├── Comment.jsx
│       │   ├── List.jsx
│       │   ├── CourseContent.jsx
│       │   ├── Study.jsx
│       │   ├── Overview.jsx
│       │   └── Video.jsx
│       ├── Checkout
│       │   ├── CheckoutPrice.jsx
│       │   ├── Checkout.jsx
│       │   ├── Payment.jsx
│       │   ├── CheckoutItem.jsx
│       │   └── CheckoutButton.jsx
│       ├── PurchaseHistory
│       │   ├── HistoryNav.jsx
│       │   ├── HistoryCard.jsx
│       │   ├── HistoryReturn.jsx
│       │   ├── HistoryBuy.jsx
│       │   └── PurchaseHistory.jsx
│       ├── Challenge
│       │   ├── ConfirmChallenge.jsx
│       │   ├── Challenge.jsx
│       │   ├── Roll.jsx
│       │   └── Category.jsx
│       ├── CreateNewCourse
│       │   ├── Category.jsx
│       │   ├── CreateNewCourse.jsx
│       │   ├── UploadFile.jsx
│       │   └── Description.jsx
│       ├── Cart
│       │   ├── Cart.jsx
│       │   ├── CartItem.jsx
│       │   └── TotalPrice.jsx
│       ├── CompleteBuy
│       │   ├── SellerInfo.jsx
│       │   ├── CompleteBuy.jsx
│       │   └── CourseCheckButton.jsx
│       ├── About
│       │   ├── About.jsx
│       │   └── Member.jsx
│       ├── HomePage
│       │   ├── HomePage.jsx
│       │   └── HomePageCourseList.jsx
│       ├── Login
│       │   ├── Login.jsx
│       │   └── Checkbox.jsx
│       ├── MyCoursePage
│       │   ├── MyCoursePage.jsx
│       │   └── MyCourseCard.jsx
│       ├── PointHistory
│       │   ├── PointList.jsx
│       │   └── PointHistory.jsx
│       ├── SearchResult
│       │   ├── SearchResultCard.jsx
│       │   └── SearchResult.jsx
│       ├── CourseInfo
│       │   ├── CourseInfo.jsx
│       │   ├── EditPassword.jsx
│       │   ├── EditProfile.jsx
│       │   ├── ErrorPage.jsx
│       │   ├── GetPoint.jsx
│       │   ├── InstructorHomepage.jsx
│       │   ├── Register.jsx
│       │   └── Register.jsx
│       ├── index.js
│       └── features
│           ├── allCourses
│           │   ├── allCoursesService.js
│           │   └── allCoursesSlice.js
│           ├── auth
│           │   ├── authService.js
│           │   └── authSlice.js
│           ├── cart
│           │   ├── cartService.js
│           │   └── cartSlice.js
│           ├── challenge
│           │   ├── challengeSlice.js
│           │   └── challengeService.js
│           ├── course
│           │   ├── courseService.js
│           │   └── courseSlice.js
│           ├── points
│           │   ├── pointsService.js
│           │   └── pointsSlice.js
│           ├── transaction
│           │   ├── transactionSlice.js
│           │   └── transactionService.js
│           ├── user
│           │   ├── userService.js
│           │   └── userSlice.js
│           └── api
│               └── index.js
│
├── assets
│   ├── icons
│   └── images
├── utils
│   ├── localStorage.js
│   └── authHeader.js
├── styles
│   └── style.scss
├── app.yaml
├── public
│   ├── index.html
│   └── manifest.json
└── postcss.config.js
    
```

圖 18：CarryU專案前端資料夾結構



# Server Tree

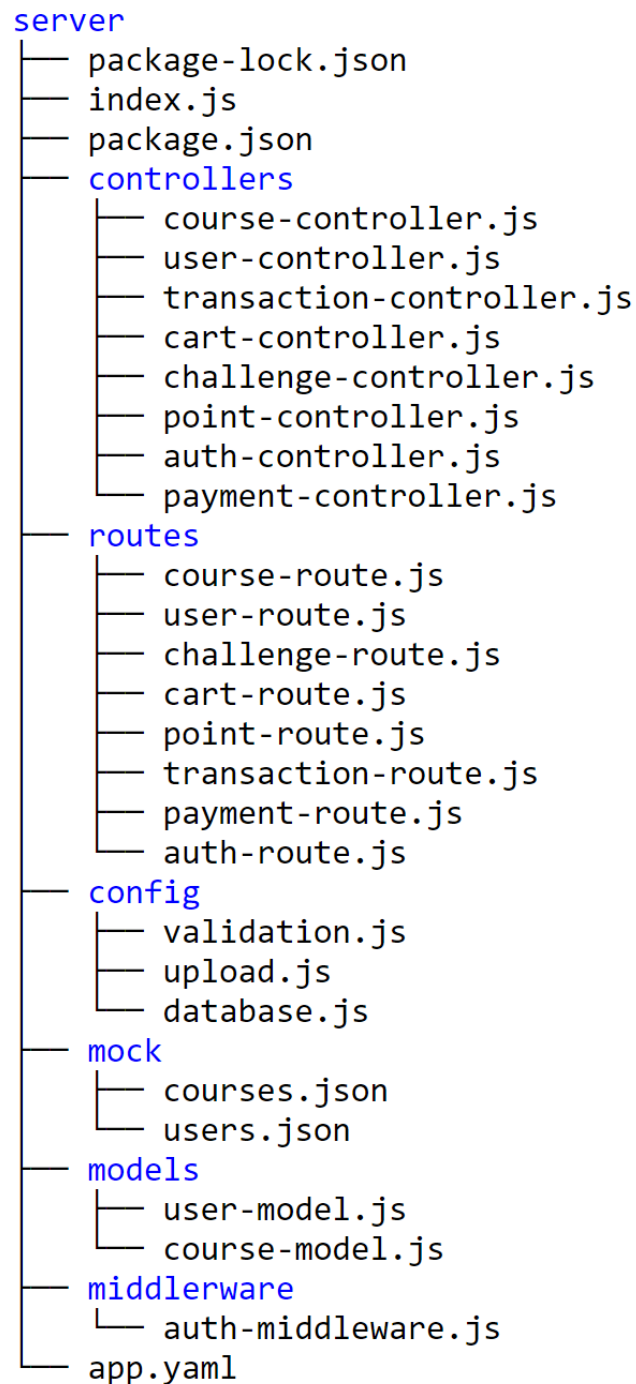


圖 19：CarryU專案後端資料夾結構

## 6.5 Route 列表

以下表格（表 5）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，因開發時間有限，下表標註\*號為api已串接好但未使用，共42支api，實際已運用共39支，列表如下：

表 5：CarryU Route 列表

Index	Route	Action	網址參數	功能描述/作法
1	api/auth/login	POST		授權登入
2	api/auth/logout	GET		授權登出
3	api/cart	POST		增加課程至購物車
4	api/cart	DELETE		刪除購物車課程
5	api/cart	GET		取得購物車課程資料
6	api/course/create	POST		上傳新課程
7	api/course/:_id/info	POST	_id	更新課程資訊
8	api/course/:_id/info	GET	_id	以id搜尋課程資訊
9	api/course/:_id/content	GET	_id	以id搜尋課程內容
10	api/course/search/:title	GET	title	以title搜尋課程
11	api/course/:_id/comment	POST	_id	新增課程留言
12	api/course/:_id/comment	GET	_id	取得所有課程留言
13	api/course/all	GET		取得所有課程
14	api/course/popular	GET		取得前十熱門課程

15	api/course/newest	GET		取得前十最新課程
16	api/course/category/: category	GET		以類別搜尋課程
17	api/course/random	GET		隨機取得課程
18	api/course/: _id/teaching-assistant	PUT	_id	新增課程助教
19*	api/course/: _id/teaching-assistant	DELETE	_id	刪除課程助教
20	api/course/: _id/teaching-assistant	GET	_id	取得課程助教
21	api/course/: _id/student	POST	_id	新增學生至課程
22*	api/course/: _id/student	DELETE	_id	從課程移除學生
23	api/course/: _id/student	GET	_id	取得課程中的學生
24	api/course/: _id/video	POST	_id	上傳課程影片
25	api/points/add	PUT		新增點數
26	api/points/delete	PUT		刪減點數
27	api/points/get	GET		取得點數資料
28	api/transaction/purchase	POST		新增購買紀錄
29	api/transaction/refund	POST		新增退貨紀錄
30	api/user/register	POST		會員註冊
31	api/user/profile	GET		取得會員資料
32	api/user/profile	PATCH		更新會員資料
33	api/user/purchase-history	GET		取得會員購買紀錄
34	api/user/password	PATCH		更新會員密碼

35	api/user/switch-role	PATCH		會員切換角色
36*	api/user/delete-user/ _:id	DELETE	_id	刪除會員
37	api/user/:_id	GET		以id搜尋會員
38	api/user/get-all-user	GET		取得所有會員
39	api/challenge/add	PUT		新增挑戰紀錄
40	api/challenge/set-to-success	PUT		挑戰成功
41	api/challenge/set-to-failed	PUT		挑戰失敗
42	api/challenge	GET		取得挑戰紀錄

## 6.6 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 (public repositories) 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/CARRYUU/carryu>。

## 第 7 章 專案程式設計

以下說明本專案之特殊設計與設計原理緣由，同時說明與其他專案可能不同之處，並針對本專案之設計理念與重點進行闡述。

本專案後端所需要import之項目採用require()語法進行import，以 ./server/index.js 為例，如下圖（圖 20）所示，若需要額外 import 不同的物件請如同第1行方式進行import。

```
1 + const express = require("express");
2   require("dotenv").config();
3   const connectDB = require("./config/database.js");
4   const cors = require("cors");
5   const app = express();
6
7   app.use(cors());
8   const PORT = process.env.PORT || 3000;
9
10  // Import routes
11  const authRoute = require("./routes/auth-route.js");
12  const userRoute = require("./routes/user-route.js");
13  const courseRoute = require("./routes/course-route.js");
14  const cartRoute = require("./routes/cart-route.js");
15  const pointRoute = require("./routes/point-route.js");
16  const paymentRoute = require("./routes/payment-route.js");
17  const transactionRoute = require("./routes/transaction-route.js");
```

圖 20：後端import範例圖

前端則以 ./client/index.js 為例，如下圖（圖 21）所示，若需要額外import不同的 package 物件請如同第1行方式進行 import。

```
1 + import React from "react";
2   import ReactDOM from "react-dom/client";
3   import { BrowserRouter } from "react-router-dom";
4   import { Provider } from "react-redux";
5   import { store } from "./store";
6   import "./index.css";
7   import App from "./App";
```

圖 21：前端import範例圖

## 7.1 JSON

### 7.1.1 JSON 格式介紹

JSON (JavaScript Object Notation) 為一種輕量級之資料交換語言，其以 KEY-VALUE 為基礎，其資料型態允許數值、字串、有序陣列 (array) 和無序物件 (object)，官方 MIME 類型為「application/json」，副檔名是「.json」。

```
1 {  
2   "username": "test001",  
3   "email": "test001@mail.com",  
4   "role": "student",  
5   "address": {  
6     "street": "123 Main St",  
7     "city": "New York",  
8     "state": "NY",  
9     "zip": "10001"  
10  },  
11  "courses": [  
12    {  
13      "name": "Math",  
14      "grade": "A"  
15    },  
16    {  
17      "name": "English",  
18      "grade": "B"  
19    }  
20  ]  
21 }
```

圖 22：JSON格式範例圖

上圖（圖18）為常見之 JSON 格式，其中無序物件會以「{ }」包覆（圖中紅色區域），而物件內之組成為 key-value，有序陣列則以「[ ]」包覆（圖中綠色區域），其中陣列內可為上述之各種資料型態。

### 7.1.2 前端發送 AJAX Request 說明

前端與後端之間建立非同步請求可透過 JavaScript 原生之 XMLHttpRequest (XHR) 或使用Axios簡化請求過程，於本專案中選擇後者作為溝通之撰寫方式。本小節敘述伺服器端與用戶端之間的資料傳輸與資料格式判斷透過 JSON 和 Axios之間的互動關係。

1. 本專案當中，API 溝通的標準格式為 JSON，並且使用 AJAX 之 Request 方式呼叫 API。

2. 用戶端 (Client 端) 之資料驗證將於伺服器端 (Server 端) 使用定義好的validation進行，驗證之套件為Joi。並以react-toastify客製化的方式通知使用者錯誤之訊息，如下圖（圖 25）為例：

```
server > config > validation.js > ...
You, 3 weeks ago | 2 authors (You and others)
1 const Joi = require("joi");
2
3 const registerValidation = (data) => {
4   const schema = Joi.object({
5     username: Joi.string().min(3).max(255).required("Name is required"),
6     email: Joi.string().min(6).max(255).required("Email is required").email(),
7     password: Joi.string().min(6).max(255).required("Password is required"),
8   });
9
10  return schema.validate(data);
11  };
```

圖 23：後端使用Joi製作registerValidation

```
server > controllers > user-controller.js > registerUser
You, 5 hours ago | 2 authors (You and others)
1 const User = require("../models/user-model.js");
2 const registerValidation =
3   require("../config/validation.js").registerValidation;
4 const updateProfileValidation =
5   require("../config/validation.js").updateProfileValidation;
6 const passwordValidation =
7   require("../config/validation.js").passwordValidation;
8
9 // @desc Register new user
10 // @route POST api/user/register
11 // @access Public
12 exports.registerUser = async (req, res) => {
13   // Destruct register data from request body.
14   let { username, email, password } = req.body;
15
16   // Check if register data is valid.
17   const { error } = registerValidation({ username, email, password });
18   if (error) return res.status(400).json({ err_msg: error.details[0].message });
```

圖 24：於user-controller.js 中導入並使用registerValidation驗證資料

```
39 const message =
40   (error.response && error.response.data && error.response.data.err_msg) ||
41   error.response.data.message ||
42   error.message ||
43   error.toString();
44
45 toast.error(message, {
46   toastId: "registerUserError",
47   theme: "colored",
48 });
```

圖 25：前端呼叫API後，出現錯誤時react-toastify會顯示錯誤通知

```

client > src > features > api > js index.js > ...
  You, 5 days ago | 2 authors (You and others)
1  import axios from "axios";
2  import { getUserFromLocalStorage } from "../../utils/localStorage";
3
4  const API = axios.create({
5    baseURL: process.env.REACT_APP_API_URL,
6    timeout: 2000,
7  });
8
9  API.interceptors.request.use(
10    (config) => {
11      const user = getUserFromLocalStorage();
12
13      if (user) {
14        config.headers.authorization = `Bearer ${user.token}`;
15      }
16
17      return config;
18    },
19    (error) => {
20      return Promise.reject(error);
21    }
22  );
23
24  // Path: baseURL/auth/
25  export const login = (data) => API.post("auth/login", data);
26  export const logout = () => API.get("/auth/logout");

```

圖 26：前端使用axios呼叫API

3. baseURL：指定之 API 路徑。
4. timeout：設定 AJAX 最多等待時間，避免檢索時間過久。
5. 圖中9~22行：設定呼叫API前必須先嘗試取得用戶端本地儲存的使用者Token，並且將其放入Request的Header的authorization中，其類別為 Bearer Token。
6. API.\*：需依照 API 指定 GET/POST/DELETE/PUT/PATCH以及API路徑。
7. data：傳送資料須以 JSON 格式傳送。



### 7.1.3 前端表格 Render 與欄位回填

由於採用 AJAX 方式進行溝通，因此需要藉由 JavaScript 將頁面上之元素，以 Response 之結果進行更新，下圖（圖 27）顯示會員更新之欄位，根據檢所之結果進行回填方式：

```
220  return res.status(200).json({
221      success: true,
222      msg: "User role switched to instructor successfully",
223      user_profile: {
224          username: updatedUser.username,
225          email: updatedUser.email,
226          role: updatedUser.role,
227      },
228  });
```

圖 27：更新會員資料以json格式回傳

若資料要以表格方式呈現，可以先使用redux套件工具取得狀態（行20），並且使用useEffect設定當頁面刷新時取得使用者資料（行40~42）。

```
20  const { user } = useSelector((state) => state.user);
21
22  const { username, email } = formData;
23
24  const dispatch = useDispatch();
25
26  const handleChange = (e) => {
27      setFormData((prevState) => ({
28          ...prevState,
29          [e.target.name]: e.target.value,
30      }));
31  };
32
33  const handleSubmit = (e) => {
34      dispatch(updateUserProfile(formData));
35      setTimeout(() => {
36          window.location.reload(true);
37      }, 1000);
38  };
39
40  useEffect(() => {
41      dispatch(getUserProfile());
42  }, []);
```

圖 28：更新會員資料後取得更新後的資料

取得更新後的資料後，填入我們做好的元素即可正確顯示。

```

44   return (
45     <div>
46       <Title pageTitle="Profile & Setting" />
47       <div className="flex max-w-md flex-col items-center justify-center px-4 py-8 mx-auto md:h-screen lg:py-0">
48         <div className="flex flex-col items-center justify-center w-full py-8 px-2 bg-white rounded-lg shadow
49           dark:bg-gray-100">
50           <Input
51             labelName="Username"
52             name="username"
53             type="text"
54             placeholder={user?.username}
55             onChange={handleChange}
56             value={username}
57           />
58           <Input
59             labelName="Email"
60             name="email"
61             type="email"
62             placeholder={user?.email}
63             onChange={handleChange}
64             value={email}
65           />
66           <Button buttonName="Save all" onClick={handleSubmit} />
67         </div>
68       </div>
69     </div>
70   );

```

圖 29：取得更新後的資料進行刷新

## 7.2 MongoDB 與 Mongoose

此部分之 class 可自行增加功能。

在NodeJS當中，本專案使用Mongoose用以連線MongoDB Atlas資料庫進行操作，同時為達成更動的便利性，本專案將有關資料庫之溝通 method 以 mongoose之功能進行，以下節錄說明mongoose 之實作方式，詳細之內容可參閱 mongoose docs ( <https://mongoosejs.com/docs/guide.html> )。

以下圖（圖 31）新增點數為例，為使用mongoose之資料庫操作，需要require mongoose套件，其詳細步驟如下：

- 圖 30 第 8 行透過 mongoose.connect()建立連線，並呼叫process.env的MONGO\_URI變數連接資料庫。
- 圖 30 第 14 行為若連線成功會回傳成功訊息，連線失敗則會顯示第 17 行catch內的錯誤訊息。
- 圖 31 第 18 行使用mongoose的updateOne以更新資料庫資料。
- 圖 31 第 18 行找尋當前user的\_id，並將之點數屬性數量更改為newPoints。
- 圖 32 第 6 行userSchema被引用至updateOne method，但因\_id在mongoose中會自動生成，故於Schema內無需特別寫入。

```

server > config > js database.js > ...
You, 3 weeks ago | 1 author (You)
1  const mongoose = require("mongoose");
2
3  const dbName = "carryu";
4
5  const connectDB = () => {
6    console.log("Connecting to database...");
7
8    mongoose
9      .connect(process.env.MONGO_URI, {
10        useNewUrlParser: true,
11        useUnifiedTopology: true,
12        dbName,
13      })
14      .then(() => {
15        console.log(`MongoDB connected! Connected to database ${dbName}`)
16      })
17      .catch((err) => console.log(err));
18  };
19
20  module.exports = connectDB;

```

圖 30：連接資料庫之檔案（節錄）

```

16  //get newpoints
17  const newPoints = user.points + add_points;
18  User.updateOne({ _id: user._id }, { $set: { points: newPoints } })
19    .then(() => {
20      return res.status(200).json({
21        msg: "Points added successfully",
22      });
23    })
24    .catch((err) => {
25      return res.status(400).json({
26        msg: "Failed to add points",
27      });
28    });
29  });
30  };

```

圖 31：新增點數之updateOne method（節錄）

```

server > models > js user-model.js > js userSchema > js password
1  const mongoose = require("mongoose");
2  const bcrypt = require("bcrypt");
3
4  const { Schema } = mongoose;
5
6  const userSchema = new Schema({
7    username: {
8      type: String,
9      required: "Name is required",
10     minLength: 3,
11     maxLength: 255,
12   },
13   avatar: {
14     type: Buffer,
15   },
16   email: {
17     type: String,
18     match: [/.\+@.\+.\+/, "Please fill a valid email address"],
19     required: "Email is required",
20   },

```

圖 32：userSchema（節錄）