國　　立　　中　　央　　大　　學

資　訊　管　理　學　系

108（一）系　統　分　析　與　設　計

(第三次作業重點

第二，三，四章全部

(12/19 繳交初稿(需有 80%正確內容)，12/29 繳交定稿):

第二章須包含全部資料庫表格，

第三章須包含所有 java classes

第四章包含兩個使用案例就好了)

系統軟體設計規格書

第 十 組

資管三 A 106403202 許家瑜

資管三 A 106403003 余若慈

資管三 B 106403034 洪睿甫

資管三 B 106403554 羅御軒

資管三 B 106403551 呂晟維

指導教授：許智誠　教授、陳以錚　教授

中　華　民　國　1　0　8　年　1　2　月　3　1　日

設　計　文　件　評　分　標　準

各項合起來就是這個文件的分數(110%)。

ER 圖(20%):

是否有包含"所有"必須的資料表及欄位

各表格的鍵值設定(主鍵與外鍵 primary key/ foreign key)是否正確

類別圖(30%):

是否有包含"所有"必須的 class (反映強韌圖與 ER 描述的類別)

各個類別的欄位、方法定義是否正確

各個類別之間相互關係的表達是否正確

循序圖(40%):

是否有包含"所有"必須的 class (包含整個 use case 的完整執行)

整個循序圖的呼叫順序是否正確

每個 method 的呼叫與傳回值是否正確標示

循序圖的各種 opt, alt 等等的相互關係是否正確表達

整體文件部分(20%):

在最後，將第二、三、四章以外的部分正確撰寫，反映該專案的實際規格

目錄

表目錄

圖目錄

版本修訂

版本	修訂者	修訂簡述	日期
V0.1.0	林泓志	Draft	2019/08/29
V1.1.0	呂晟維	資料庫	2019/12/16
V1.2.0	許家瑜 余若慈	循序圖	2019/12/18
V1.3.0	羅御軒 洪睿甫	類別圖	2019/12/18
V2.1.0	全體	循序圖大更新	2019/12/25
V2.1.1	許家瑜 余若慈	循序圖物件存活時間	2019/12/27

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，SRS）、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

本系統範圍用於電子商務，其中主要包含會員管理、商品管理、訂購商品與結帳等四個模組，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電子商務所需的管理流程。

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. 系統分析與設計—系統環境架設

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）、 3 個關係（Relationship）、 1 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）進行取代：

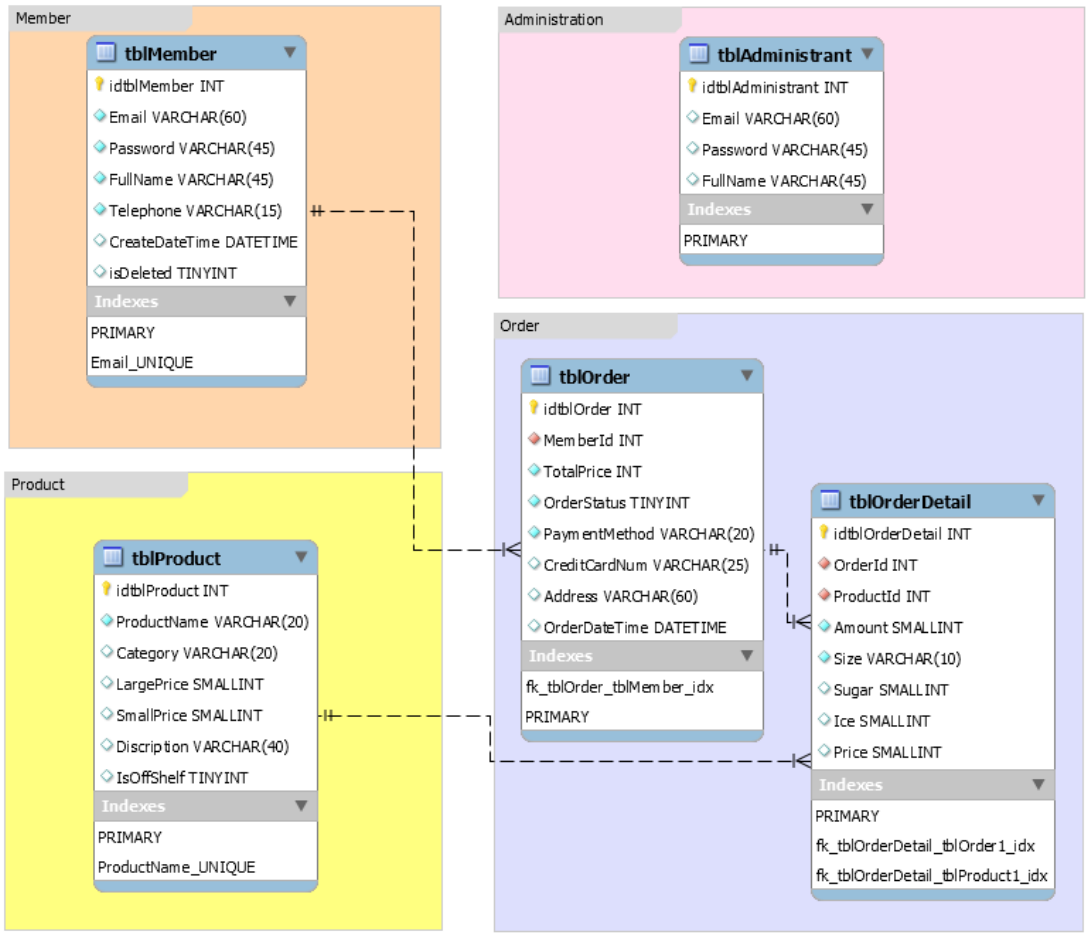

圖 1：設計階段之實體關係圖

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：

1. 會員資料表（tblmember）：

表 1：會員資料表（tblmember）之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K	idtblMember	Int(10)	無	否	V	binary
	Email	Varchar(45)	無	否		utf8mb4
	Password	Varchar(45)	無	否		utf8mb4
	Fullname	Varchar(45)	無	否		utf8mb4
	Telephone	Varchar(15)	無	否		utf8mb4
	CreateDateTime	Datetime	null	是		binary
	isDeleted	Tinyint	0	是		binary

- ✓ idtblMember：為自動增加作為會員編號，不可更動由資料庫系統自動產生。
- ✓ Email：用於紀錄會員的電子郵件信箱。需獨一無二。
- ✓ Password：用於紀錄該名會員的密碼。暫時不加密。
- ✓ Fullname：用於記錄該名會員的全名。
- ✓ Telephone：用於記錄該名會員的電話。
- ✓ CreateDateTime：用於記錄該名會員創立帳號的日期與時間。
- ✓ isDeleted：會員刪除的 boolean 值 flag。如果會員已遭刪除，系統不會直接從資料庫將該名會員 record，而是將 isDeleted 欄位設成 1。此欄位預設值為 0。

2. 管理者資料表（tbladministrant）：

表 2：管理者資料表（tbladministrant）之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K	idtblAdministrant	Int(10)	無	否	V	binary
	Email	Varchar(45)	無	否		utf8mb4
	Password	Varchar(45)	無	否		utf8mb4
	FullName	Varchar(45)	無	否		utf8mb4

- ✓ idtblAdministrant：為自動增加作為管理者編號，不可更動由資料庫系統自動產生。
- ✓ Email：用於紀錄管理者的電子郵件信箱。需獨一無二。
- ✓ Password：用於紀錄該名管理者的密碼。暫時不加密。
- ✓ Fullname：用於記錄該名管理者的全名。

3. 商品資料表（tblproduct）：

表 3：商品資料表（tblproduct）之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K	idtblProduct	Int(10)	無	否	V	binary
	ProductName	Varchar(20)	無	否		utf8mb4
	Category	Varchar(20)	無	否		utf8mb4
	LargePrice	SmallInt(5)	無	否		binary
	SmallPrice	SmallInt(5)	無	否		binary
	Discription	Varchar(40)	null	是		utf8mb4
	IsOffShelf	TinyInt(4)	0	是		binary

- ✓ idtblProduct：為自動增加作為商品編號，不可更動由資料庫系統自動產生。
- ✓ ProductName：用於紀錄商品之商品名稱。
- ✓ Category：用於紀錄商品之商品分類，有”想喝茶”、”找鮮奶” 、”手作特調”三類。
- ✓ LargePrice：用於紀錄商品之大杯價格。非負 0 to 65535。
- ✓ SmallPrice：用於紀錄商品之小杯價格。非負 0 to 65535。
- ✓ Discription：用於紀錄商品之商品描述。
- ✓ IsOffShelf：商品刪除的 boolean 值 flag。如果商品已遭刪除，系統不會直接從資料庫將該項商品 record，而是將 isDeleted 欄位設成 1。此欄位預設值為 0。

4. 訂單資料表（tblorder）：

表 4：訂單資料表（tblorder）之資料結構

Key	名稱	類型	預設值	空值	自動	編碼

					增加	
P.K	idtblOrder	Int(10)	無	否	V	binary
	MemberId	Int(10)	無	否		binary
	TotalPrice	Int(11)	無	否		binary
	OrderStatus	Tinyint(4)	'0'	否		binary
	PaymentMethod	Varchar(20)	無	否		utf8mb4
	CreditCardNum	Varchar(25)	Null	是		utf8mb4
	Address	Varchar(60)	Null	是		utf8mb4
	OrderDateTime	Datetime	Null	是		binary

- ✓ idtblOrder：為自動增加作為訂單編號，不可更動由資料庫系統自動產生。
- ✓ MemberId：用於紀錄訂單之訂購人 id。外來鍵。
- ✓ TotalPrice：用於紀錄訂單總價。
- ✓ OrderStatus：用於紀錄訂單完成狀態，只有 0 和 1。0 為進行中，1 為已完成。
- ✓ PaymentMethod：用於紀錄訂單支付方式，有來店取貨和信用卡。
- ✓ CreditCardNum：若用信用卡支付，需要紀錄卡號。
- ✓ Address：用於紀錄訂單之配送地址。
- ✓ OrderDateTime：用於紀錄訂單之下單時間。

5. 訂單品項資料表（tblOrderDetail）：

表 4：訂單品項資料表（tblOrderDetail）之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K	idtblOrderDetail	Int(10)	無	否	V	binary
	OrderId	Int(10)	無	否		binary
	ProductId	Int(10)	無	否		binary
	Amount	Smallint(6)	無	否		binary
	Size	Varchar(10)	無	否		utf8mb4
	Sugar	Varchar(10)	全糖	否		utf8mb4
	Ice	Varchar(10)	全冰	是		utf8mb4
	Price	Smallint(5)	Null	是		binary

- ✓ idtblOrderDetail：為自動增加作為訂單品項的編號，不可更動由資料庫系統自動產生。

- ✓ OrderId：外來鍵。用於紀錄所屬訂單 id，因為 1 筆訂單可有多筆品項。
- ✓ ProductId：外來鍵。用於紀錄購買商品的 id。
- ✓ Amount；用於紀錄購買商品的數量。
- ✓ Size：用於紀錄中杯或大杯，以 middle large 代表。
- ✓ Sugar：用於紀錄飲料甜度，以全糖少糖半糖三分糖無糖表示。
- ✓ Ice：用於紀錄飲料冰度，以全冰少冰半冰三分冰無冰表示。
- ✓ Price：用於紀錄訂單品項的總價，等於單價*數量。

第 3 章　類別圖

下圖（**錯誤! 找不到參照來源。**、圖 2、**錯誤! 找不到參照來源。**）係依據電子商務線上訂購系統的分析模型和建立的互動圖，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：類別圖除包含與資料庫相對應之物件外，亦包含相關之控制物件（controller）、DBMgr 與各功能相對應資料庫操作類別（例如：MemberHelper）和相對應之類別工具（JsonReader）。

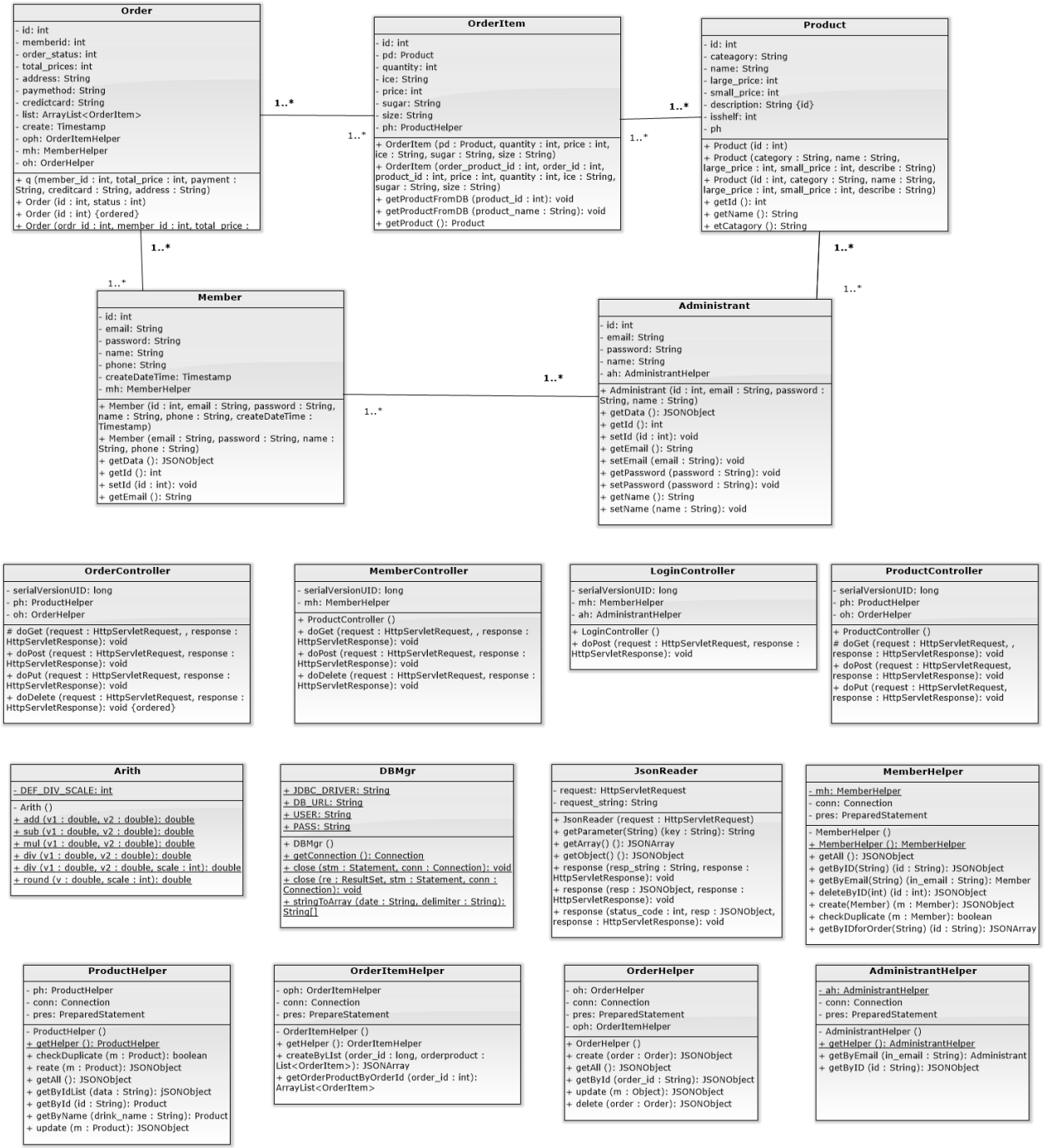
Order
- id: int
- memberid: int
- order_status: int
- total_prices: int
- address: String
- paymethod: String
- creditcard: String
- list: ArrayList<OrderItem>
- create: Timestamp
- oph: OrderItemHelper
- mh: MemberHelper
- oh: OrderHelper
+ q (member_id : int, total_price : int, payment : String, creditcard : String, address : String)
+ Order (id : int, status : int)
+ Order (id : int) {ordered}
+ Order (ordr_id : int, member_id : int, total_price :
OrderItem
- id: int
- pd: Product
- quantity: int
- ice: String
- price: int
- sugar: String
- size: String
- ph: ProductHelper
+ OrderItem (pd : Product, quantity : int, price : int, ice : String, sugar : String, size : String)
+ OrderItem (order_product_id : int, order_id : int, product_id : int, price : int, quantity : int, ice : String, sugar : String, size : String)
+ getProductFromDB (product_id : int): void
+ getProductFromDB (product_name : String): void
+ getProduct (): Product
Product
- id: int
- cateagory: String
- name: String
- large_price: int
- small_price: int
- description: String {id}
- isshelf: int
- ph
+ Product (id : int)
+ Product (category : String, name : String, large_price : int, small_price : int, describe : String)
+ Product (id : int, category : String, name : String, large_price : int, small_price : int, describe : String)
+ getId (): int
+ getName (): String
+ etCatagory (): String
Member
- id: int
- email: String
- password: String
- name: String
- phone: String
- createDateTime: Timestamp
- mh: MemberHelper
+ Member (id : int, email : String, password : String, name : String, phone : String, createDateTime : Timestamp)
+ Member (email : String, password : String, name : String, phone : String)
+ getData (): JSONObject
+ getId (): int
+ setId (id : int): void
+ getEmail (): String
Administrant
- id: int
- email: String
- password: String
- name: String
- ah: AdministrantHelper
+ Administrant (id : int, email : String, password : String, name : String)
+ getData (): JSONObject
+ getId (): int
+ setId (id : int): void
+ getEmail (): String
+ setEmail (email : String): void
+ getPassword (password : String): void
+ setPassword (password : String): void
+ getName (): String
+ setName (name : String): void
1..*
1..*
1..*
1..*
1..*
1..*
1..*
1..*
OrderController
- serialVersionUID: long
- ph: ProductHelper
- oh: OrderHelper
doGet (request : HttpServletRequest, , response : HttpServletResponse): void
+ doPost (request : HttpServletRequest, response : HttpServletResponse): void
+ doPut (request : HttpServletRequest, response : HttpServletResponse): void
+ doDelete (request : HttpServletRequest, response : HttpServletResponse): void {ordered}
MemberController
- serialVersionUID: long
- mh: MemberHelper
+ ProductController ()
+ doGet (request : HttpServletRequest, , response : HttpServletResponse): void
+ doPost (request : HttpServletRequest, response : HttpServletResponse): void
+ doDelete (request : HttpServletRequest, response : HttpServletResponse): void
LoginController
- serialVersionUID: long
- mh: MemberHelper
- ah: AdministrantHelper
+ LoginController ()
+ doPost (request : HttpServletRequest, response : HttpServletResponse): void
ProductController
- serialVersionUID: long
- ph: ProductHelper
- oh: OrderHelper
+ ProductController ()
# doGet (request : HttpServletRequest, , response : HttpServletResponse): void
+ doPost (request : HttpServletRequest, response : HttpServletResponse): void
+ doPut (request : HttpServletRequest, response : HttpServletResponse): void
Arith
- DEF_DIV_SCALE: int
- Arith ()
+ add (v1 : double, v2 : double): double
+ sub (v1 : double, v2 : double): double
+ mul (v1 : double, v2 : double): double
+ div (v1 : double, v2 : double): double
+ div (v1 : double, v2 : double, scale : int): double
+ round (v : double, scale : int): double
DBMgr
+ JDBC_DRIVER: String
+ DB_URL: String
+ USER: String
+ PASS: String
+ DBMgr ()
+ getConnection (): Connection
+ close (stm : Statement, conn : Connection): void
+ close (re : ResultSet, stm : Statement, conn : Connection): void
+ stringToArray (date : String, delimiter : String): String[]
JsonReader
- request: HttpServletRequest
- request_string: String
+ JsonReader (request : HttpServletRequest)
+ getParameter(String) (key : String): String
+ getArray() (): JSONArray
+ getObject() (): JSONObject
+ response (resp_string : String, response : HttpServletResponse): void
+ response (resp : JSONObject, response : HttpServletResponse): void
+ response (status_code : int, resp : JSONObject, response : HttpServletResponse): void
MemberHelper
- mh: MemberHelper
- conn: Connection
- pres: PreparedStatement
- MemberHelper ()
+ MemberHelper (): MemberHelper
+ getAll (): JSONObject
+ getByID(String) (id : String): JSONObject
+ getByEmail(String) (in_email : String): Member
+ deleteByID(int) (id : int): JSONObject
+ create(Member) (m : Member): JSONObject
+ checkDuplicate (m : Member): boolean
+ getByIDforOrder(String) (id : String): JSONArray
ProductHelper
- ph: ProductHelper
- conn: Connection
- pres: PreparedStatement
- ProductHelper ()
+ getHelper (): ProductHelper
+ checkDuplicate (m : Product): boolean
+ reate (m : Product): JSONObject
+ getAll (): JSONObject
+ getByIdList (data : String): jSONObject
+ getById (id : String): Product
+ getByName (drink_name : String): Product
+ update (m : Product): JSONObject
OrderItemHelper
- oph: OrderItemHelper
- conn: Connection
- pres: PrepareStatement
- OrderItemHelper ()
+ getHelper (): OrderItemHelper
+ createByList (order_id : long, orderproduct : List<OrderItem>): JSONArray
+ getOrderProductByOrderId (order_id : int): ArrayList<OrderItem>
OrderHelper
- oh: OrderHelper
- conn: Connection
- pres: PreparedStatement
- oph: OrderItemHelper
+ OrderHelper ()
+ create (order : Order): JSONObject
+ getAll (): JSONObject
+ getById (order_id : String): JSONObject
+ update (m : Object): JSONObject
+ delete (order : Order): JSONObject
AdministrantHelper
- ah: AdministrantHelper
- conn: Connection
- pres: PreparedStatement
- AdministrantHelper ()
+ getHelper (): AdministrantHelper
+ getByEmail (in_email : String): Administrant
+ getByID (id : String): JSONObject

圖 2：類別圖

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，本電子商務線上訂購系統預計共有 4 位動作者與 27 個使用案例，並依照不同之模組區分成不同子系統共計七個子系統，其中包含以下：①會員子系統、②商品資訊子系統、③商品管理子系統、④訂購商品子系統、⑤結帳商品子系統、⑥訂單管理子系統、⑦管理者子系統，下圖（**錯誤! 找不到參照來源。**）為本系統之使用案例圖：

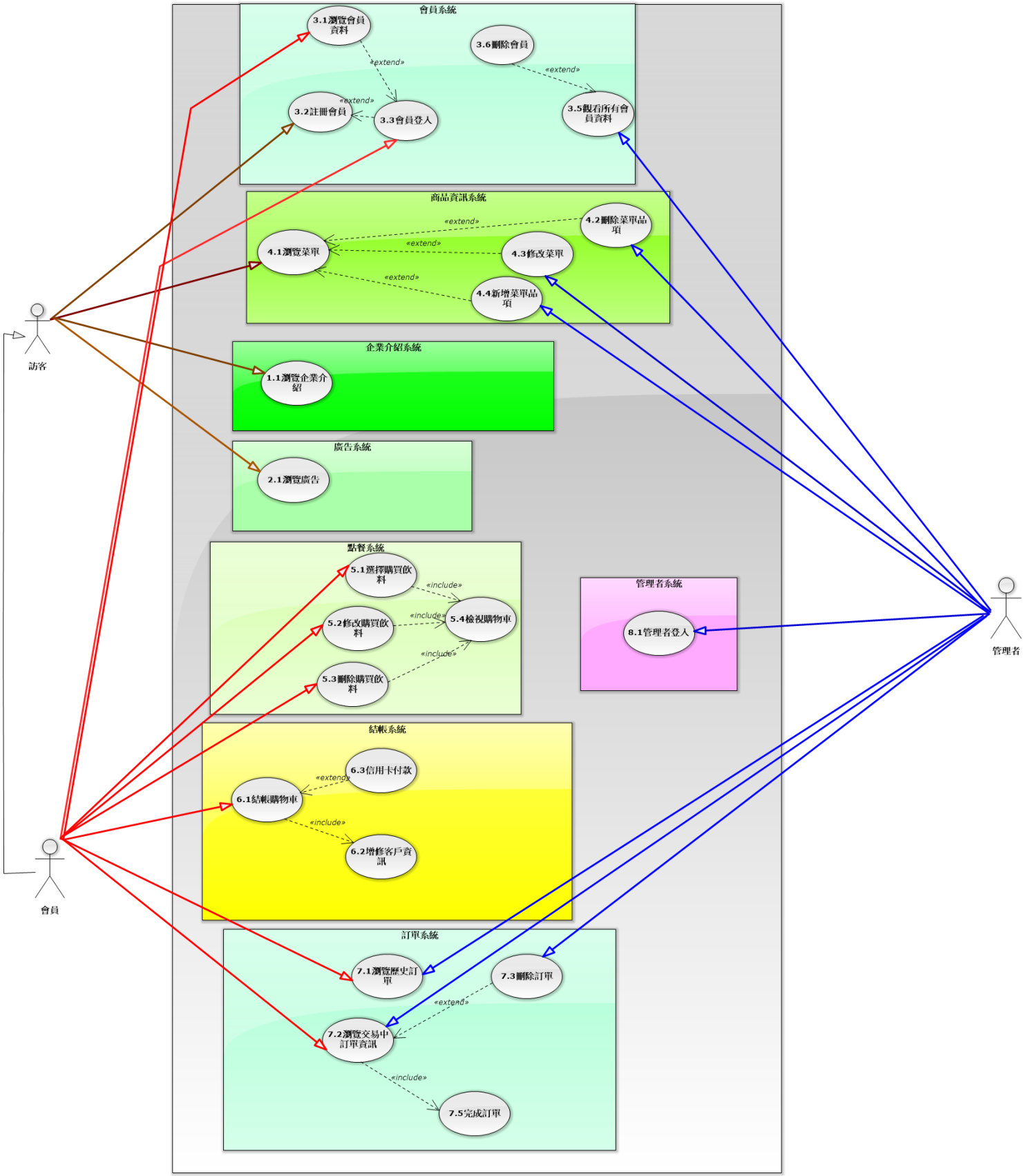

圖 3：嵐拾伍電子商務系統使用案例圖

4.2 Use Case 實做之循序圖

4.2.1 商業流程編號 5.1 會員挑選飲品

5.0 模組：點餐

點餐包含「5.1 選擇購買飲料」至「5.4 檢視購物車」四個使用案例，主要描述使用者針對所檢索之商品新增、移除至購物之行為，並可針對所訂購之商品數量進行修改。(購物車之資料將存於使用者本機中，將不會暫存或同步至本專案之資料庫當中)

5.1 選擇挑選飲料：

若一般訪客尚未登入，則進入點餐分頁前必須先登入會員。

會員可將欲想購買的商品加入至購物車，購物車的資料由 Cookie 儲存，直到完成結帳才會進入後端和資料庫。同一品項飲料但是甜度冰塊不同則記成 2 項品。

5.2 修改購買飲料：

會員可修改購物車裡的商品杯數和甜度、冰塊。

5.3 刪除購買飲料：

會員可刪除購物車裡的商品項目。

5.4 檢視購物車：

會員可瀏覽其購物車裡，目前的商品杯數、個別價格和總價格。

與該模組相關之頁面於下表（表 3）進行說明：

HTML	關聯 Use Case	說明
buy.html	Use Case 5.1 到 Use Case 5.4	會員挑選飲品、修改刪除飲品、檢視購物車

表 3：5.0 點餐模組關聯頁面

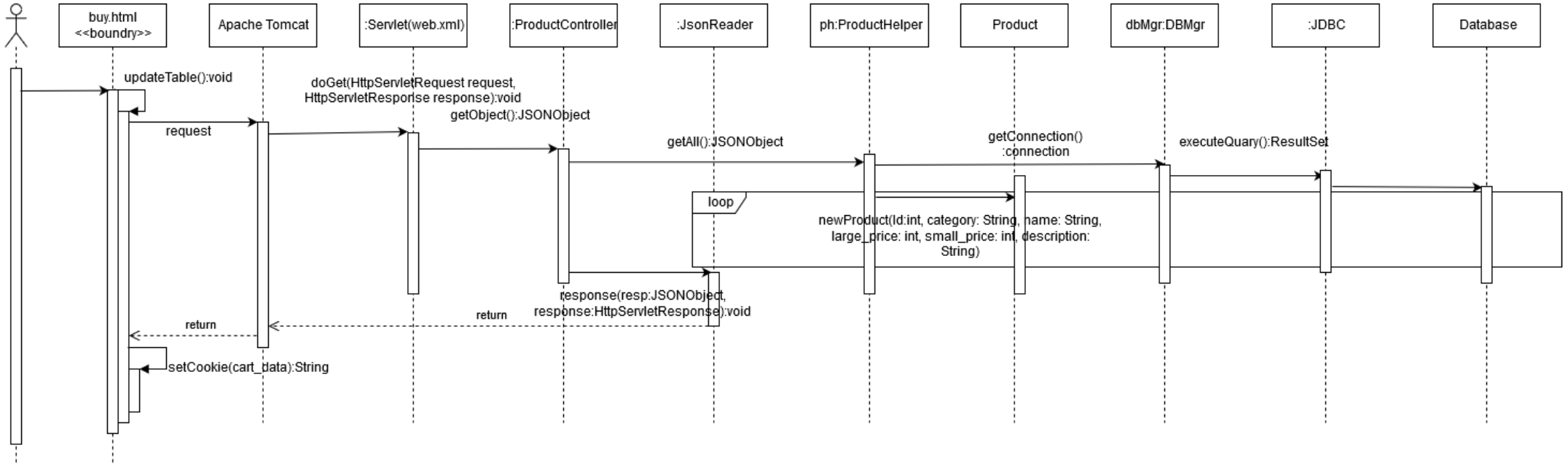

圖 1：商業流程編號 5.1 會員挑選飲料循序圖

1. 前置步驟：會員完成商業流程編號「3.2 會員登入」後，點擊進入檢視所有飲品頁面（buy.html）。
2. 針對會員進入檢視所有飲品頁面，JavaScript 之 updateTable()發送 get 之請求。
3. 後端以 ProductController 之 doGet()進行處理，再透過 ProductHelper 物件的 getAll()方法，以迴圈方式將資料庫之飲品資料撈出，並將其存成商品物件(Product 物件)以商品物件的 getData()方法，將該商品存成 JSONObject 回傳至前端。
4. 以上步驟是在取得所有品項的資訊。在挑選飲品的過程中，網頁前端會透過 JavaScript 之 setCookie ()方法來儲存購物車內的品項至瀏覽器內部，不會與伺服器互動。
5. 一旦會員完成挑選飲品要進行結帳，系統會執行商業流程編號「6.1 結帳購物車」。

表 4：商業流程編號 5.1 會員挑選飲品 Business Exception List

http status code/message	發生之 method	說明
挑選過程只有使用前端 Cookie	無後端 method	無

4.3 管理員修改飲品

4.0　商品資訊

商品資訊模組包含「4.1 瀏覽菜單」至「4.4 新增菜單品項」四個使用案例，主要用於描述使用者對於業主之商品進行檢索、瀏覽之動作，以及管理員對菜單的新增、修改、刪除菜單品項。

4.1 瀏覽菜單：

一般訪客與會員可以進入網站瀏覽以表格呈現之商品，菜單共有 3 大類飲品：茶類、拿鐵和特調。

4.2 刪除菜單品項：

管理員可以透過此功能刪除菜單品項。

4.3 修改菜單：

管理員可以透過此功能修改菜單品項和其價格，還有商品文字描述。

4.4 新增菜單品項：

管理員可以透過此功能新增菜單品項和其價格，並設定飲料類別。

與該模組相關之頁面於下表（表 3）進行說明：

HTML	關聯 Use Case	說明
edit_menu.html	Use Case 4.2~ Use Case 4.4	菜單列表、可刪除新增修改品項資訊
menu.html	Use Case 4.1	任何使用者都可以瀏覽菜單。該頁面也只提供菜單給使用者觀看，沒有其餘按鈕及事件。

表 4：後台管理者會員管理模組關聯頁面

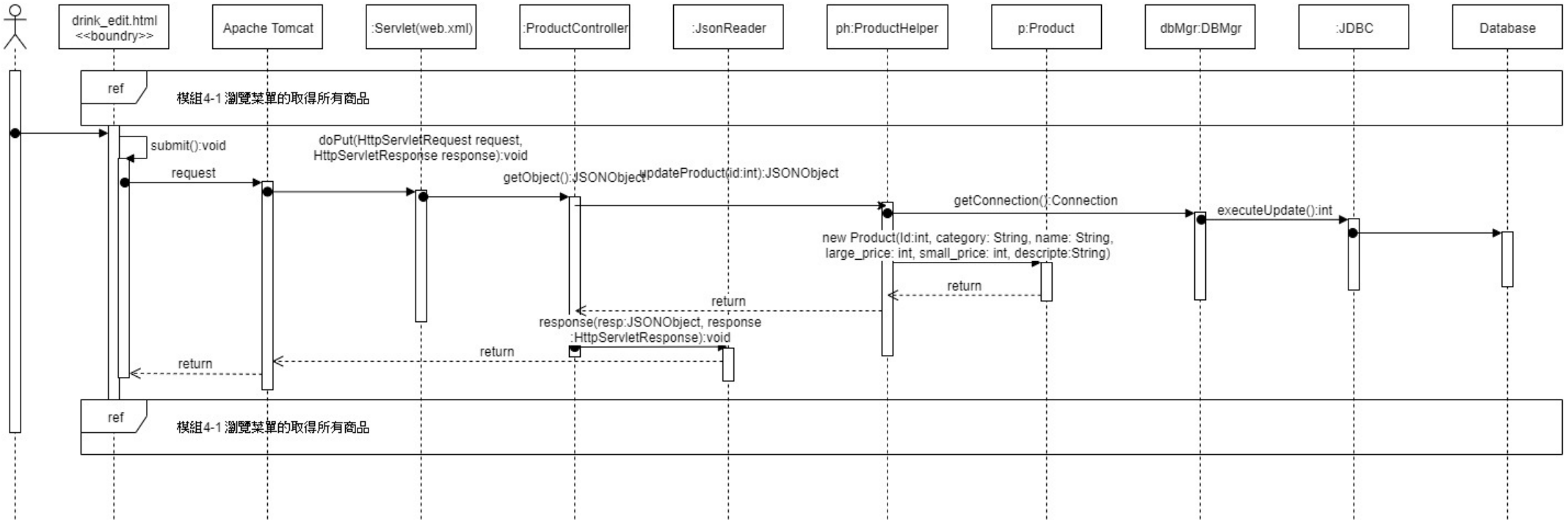

圖 2：商業流程編號 4.3 更改飲品（管理者）循序圖

1. 管理員點擊進入更改飲品頁面（drink_edit.html）。
2. 載入菜單：利用 JavaScript 之 getAllProduct ()來取得資料庫所有飲品的資料，此 getProduct()方法相同於商業流程編號「4.1 瀏覽菜單」之循序圖過程。
3. 更改菜單：針對所欲更改之飲品更改完成後，點擊確定按鈕，JavaScript 會由 submit()方法發送 put 請求。
4. 後端以 ProductController 之 doPut()進行處理，以 JsonReader 取得該參數 id 後，透過 ProductHelper 物件的 updateProduct()方法將資料庫之該飲品進行更改。
5. 若更改成功則透過 JavaScript 之 getAllProduct ()再次取得資料庫所有飲品的資料，相同於商業流程編號「4.1 瀏覽菜單」之循序圖過程。
6. 備註：第五點透過 getAllProduct ()再次取得菜單的方法，在實作過程改為 window.reload()方法，直接重整頁面。

http status code/message	發生之 method	說明
400/新增(修改)飲品失敗，飲品名稱重複！	submit()	飲品重複，新增(修改)失敗

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：Microsoft Windows 10 Pro 1903 x64

2　CPU：Intel(R) Core(TM) i7-4790 CPU @ 3.6GHz　3.6GHz

3　RAM：16.0 GB

4　Network：臺灣學術網路（TANet） — 國立中央大學校園網路 1.0Gbps

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：

1　Java JDK Version：Oracle JDK 1.8.0_202

2　Application Server：Apache Tomcat 9.0.24

3　Database：Oracle MySQL Community 8.0.17.0

4　Workbench：Oracle MySQL Workbench Community Edition 8.0.17

5　IDE：Microsoft Visual Studio Code 1.37.1

5.1　專案類型：Java Servlet 4.0

5.2　程式語言：Java

5.1.3　前端套件

1. JQuery-3.4.1
2. Js-cookie https://github.com/js-cookie/js-cookie
3. Bootstrap CSS

5.1.4 後端套件

1. json-20180813.jar：用於解析 JSON 格式
2. mysql-connector-java-8.0.17.jar：用於進行 JDBC 連線
3. servlet-api.jar：tomcat 運行 servlet 所需

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、平板電腦等）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：

1. Google Chrome 76 以上
2. Mozilla Firefox 69 以上
3. Microsoft Edge 44 以上
4. 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 4）所示，主要使用 Java 語言撰寫，伺服器端（Server）三層式 (Three-Tier) 架構，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。

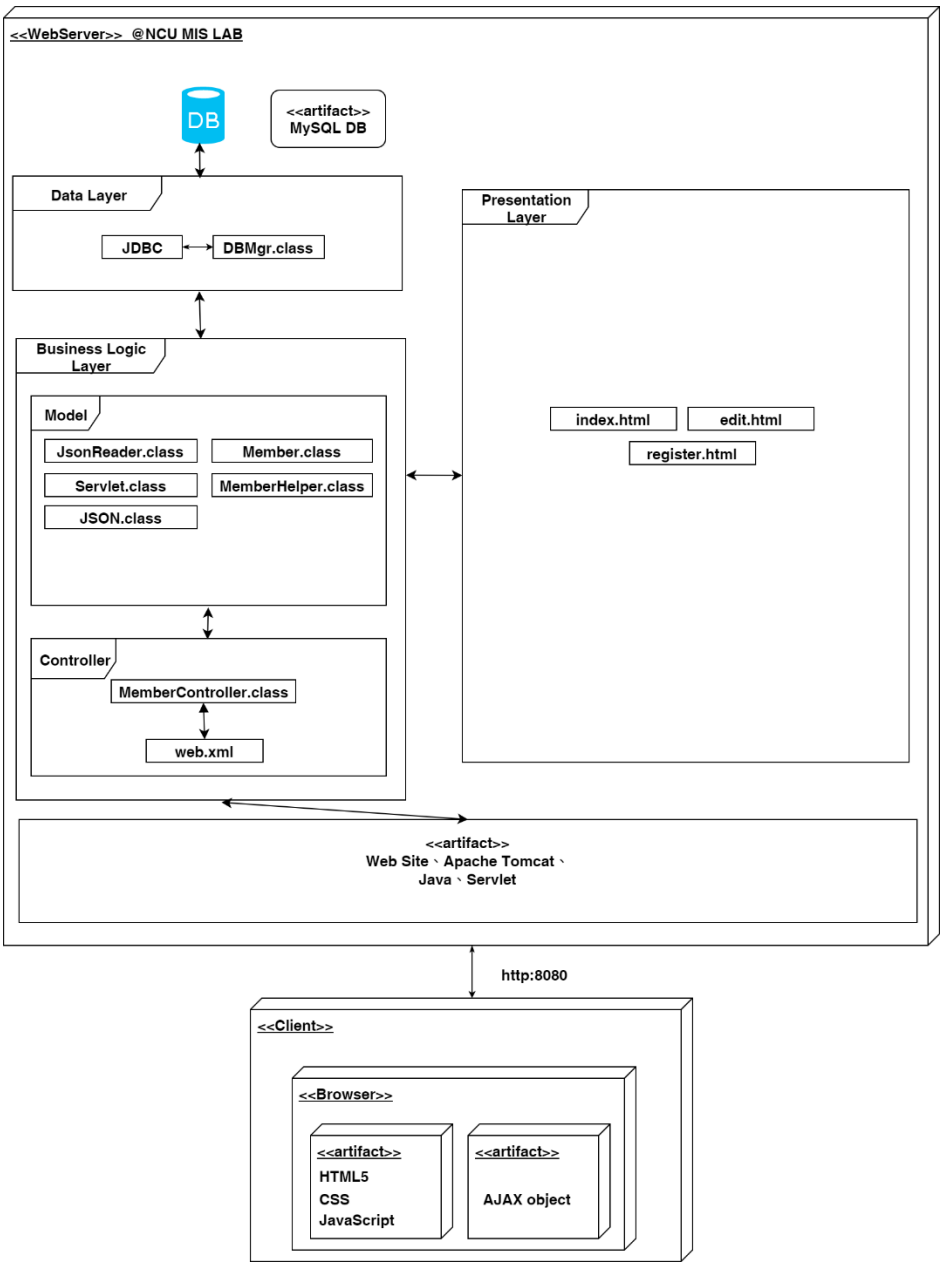

圖 4：系統架構圖

5.2.2 MVC 架構

若有使用到與此不同之方式的部分，請額外標示出來。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 Java Servlet 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（HTML）與後端（Java）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 HTML 檔案，放置於專案資料夾之根目錄，其中靜態物件則放置於「statics」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 JQuery 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 Java 程式語言進行編寫，其中可以分為「Business Model」和「View Model」兩部分，所有類別（class）需要將「*.java」檔案編譯成「*class」以繼續進行。
   - ✓ 編譯需切換到 WEB-INF 目錄下
   - ✓ 編譯指令：javac -encoding UTF-8 -classpath lib/* classes/*/*.java
2. Business Model：MVC-Model
   - ✓ 主要放置於「WEB-INF/classes/ncu/im3069/*/app/*」資料夾當中，主要有許多類別（class）於其中，如：Member.java 等。
   - ✓ 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 JDBC 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「WEB-INF/

classes/ncu/im3069/*/util/*」資料夾當中，其中包含 DBMgr.java 等

- ✓ 不同專案共用之工具則會放置到「WEB-INF/classes/ncu/im3069/tools/*」資料夾當中，如；JsonReader.java 其類別可以被其他工具共用。
- ✓ 不同專案可以使用不同資料夾（package）進行區分，以增加分類之明確度。

3. View Controls：MVC-Controller
   - ✓ 主要放置於「WEB-INF/classes/ncu/im3069/*/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 Servlet 提供之 WEB-INF 內的 web.xml 檔案指定處理的 Controller，並由後端之 Java 進行承接。
   - ✓ 主要用於控制各頁面路徑（Route）與功能流程，規劃之 use case 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 應依照以下之類別（class）之範例進行撰寫。
   - ✓ 命名會以*Controller 進行命名，同時本類別必須將 HttpServlet 進行 extends，下圖（圖 5）顯示 Controller 之範本，對應於 http method 需要時做不同的 method，如：POST-doPost()等。
   - ✓ 呼叫此 class 之路徑可於 web.xml 內進行設定或是直接於該 class 當中指定（例如：@WebServlet("/api/members")），並將其放置於命名該 class 之上（下圖為例為第 7 行之上），也可指定多路徑到此 class 中。

```java
package servletclass;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MemberController extends HttpServlet {

    public void init() throws ServletException {
        // Do required initialization
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
      // Do POST Request
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
      // Do GET Request
    }

    public void doDelete(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
      // Do DELETE Request
    }

    public void doPut(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
      // Do PUT Request
    }
}
```

圖 5：Servlet 之 Controller 範例模板（以 MemberController 為例）

✓ Servlet 提供許多 http method 之實作，詳請可查閱官方文件（https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/http/HttpServlet.html）

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 JDBC 進行實現。
2. 透過編寫 DBMgr 之 class 進行客製化本專案所需之資料庫連線內容。
3. 透過 import 此 DBMgr class 能套用到不同的功能當中，以 MemberHelper.class 為例，該 class 管理所有有關會員之方法。

✓ 以下以檢查會員帳號是否重複為例，如下圖（圖 6）所示：

A. 使用 try-cache 寫法將進行 JDBC 之 SQL 查詢。

B. 使用 prepareStatement()將要執行之 SQL 指令進行參數化查詢。

C. 透過 setString()將參數進行回填。

D. 透過 executeQuery()進行資料庫查詢動作，並將結果以 ResultSet 儲存。

E. 若要使用新增/更新/刪除，則是使用 executeUpdate()

```java
public boolean checkDuplicate(Member m){
    /** 紀錄SQL總行數，若為「-1」代表資料庫檢索尚未完成 */
    int row = -1;
    /** 儲存JDBC檢索資料庫後回傳之結果，以 pointer 方式移動到下一筆資料 */
    ResultSet rs = null;

    try {
        /** 取得資料庫之連線 */
        conn = DBMgr.getConnection();
        /** SQL指令 */
        String sql = "SELECT count(*) FROM `missa`.`members` WHERE `email` = ?";

        /** 取得所需之參數 */
        String email = m.getEmail();

        /** 將參數回填至SQL指令當中 */
        pres = conn.prepareStatement(sql);
        pres.setString(1, email);
        /** 執行查詢之SQL指令並記錄其回傳之資料 */
        rs = pres.executeQuery();

        /** 讓指標移往最後一列，取得目前有幾行在資料庫內 */
        rs.next();
        row = rs.getInt("count(*)");
        System.out.print(row);

    } catch (SQLException e) {
        /** 印出JDBC SQL指令錯誤 **/
        System.err.format("SQL State: %s\n%s\n%s", e.getErrorCode(), e.getSQLState(),
e.getMessage());
    } catch (Exception e) {
        /** 若錯誤則印出錯誤訊息 */
        e.printStackTrace();
    } finally {
        /** 關閉連線並釋放所有資料庫相關之資源 **/
        DBMgr.close(rs, pres, conn);
    }

    /**
     * 判斷是否已經有一筆該電子郵件信箱之資料
     * 若無一筆則回傳False，否則回傳True
     */
    return (row == 0) ? false : true;
}
```

圖 6：MemberHelper 之檢查會員電子郵件是否重複之 Method

✓ 本專案所使之資料庫參數則透過 Java 類別的 static final 進行宣告，透過其可快速移轉至另一個環境，如下圖（圖 7）所示：

A. 其中必須指定 JDBC_DRIVER 之名稱

B. DB_URL 必須指定資料庫所在之網址、所使用之 Port 與愈操作之資料庫。

C. 透過 getConnection()建立連線的同時，必須包含允許使用 Unicode 和 characterEncoding=utf8 之參數以避免中文字會造成異常。最終需要指定使用時區與可以不用 SSL 連線和帳號、密碼。

```java
static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
static final String DB_URL = "jdbc:mysql://localhost:3306/missa";
static final String USER = "root";
static final String PASS = "123456";

static {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public DBMgr() {

}

public static Connection getConnection() {
    Properties props = new Properties();
    props.setProperty("useSSL", "false");
    props.setProperty("serverTimezone", "UTC");
    props.setProperty("useUnicode", "true");
    props.setProperty("characterEncoding", "utf8");
    props.setProperty("user", DBMgr.USER);
    props.setProperty("password", DBMgr.PASS);

    Connection conn = null;

    try {
        conn = DriverManager.getConnection(DBMgr.DB_URL, props);
    } catch (Exception e) {
        e.printStackTrace();
    }

    return conn;
}
```

圖 7：DBMgr 類別內之資料庫參數

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）。本專案電子商務線上訂購系統，使用 Java 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 8）所示：

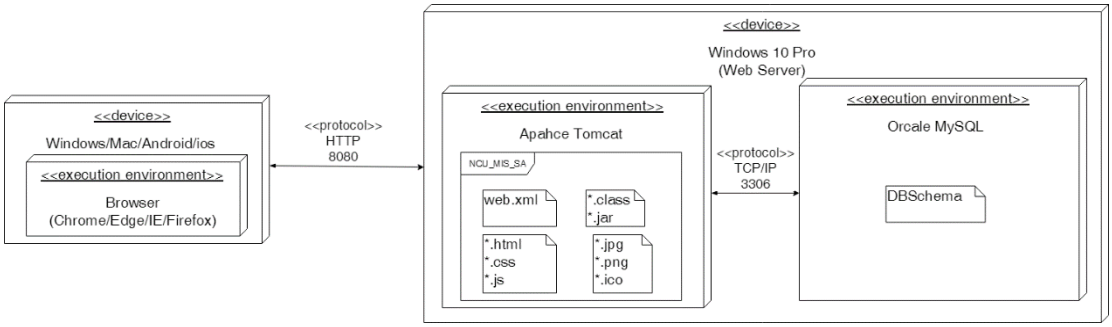

圖 8：專案部署圖

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透過匯出封裝成 Web 應用程式歸檔檔案（Web application Archive，WAR），並將所有專案檔案放置於 tomcat 指定專案資料夾當中，進行部署。
4. 本專案之資料庫將.sql 檔案之資料進行匯入，並設定完成所需之帳號、密碼與埠號（port）需與 DBMgr 類別所指定之靜態常數（static final）相同。
5. 本專案之網頁伺服器埠號採用 Apache Tomcat 預設之 8080、資料庫埠號採用 Oracle MySQL 預設之 3306。
6. 客戶端（Client）僅需使用裝置上瀏覽器，藉 http 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。

1. 通用規則

1.1. 縮行四個空白，不使用 tab。

1.2. code 一行超過 80 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. 類別（Class）

3.1. 採用「Pascal 命名法」單一單詞字首皆大寫，不包含底線或連接號，例如： DBMgr。

3.2. 主要依照 ER Diagram 進行建立，同時包含所需之 Controller，並加入另外不同兩個部分：

- ✓ Controller：命名以*Controller 為主（例如：MemberController）。
- ✓ DBMgr.java：管理所有與資料庫相關聯之函式。
- ✓ JsonReader.java：讀取 Request 之資料與回傳 JSON 格式之函式。

4. 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。

4.2. 例如：getPassword()、updateMember()、getAllMembers()等。

5. 變數（Variable）

5.1. 採用「首字皆小寫以底線區隔」之命名方式。

5.2. 例如：exexcute_sql、pre_stmt、start_time 等。

6.2 回傳訊息規範

1. 透過 JsonReader 類別之 response()的 method 進行回傳，主要需要傳入要回傳之物件與將 servlet 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 9）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。
   - ✓ message
     - 主要以中文回傳所執行之動作結果。
     - 可用於後續渲染（Render）至前端畫面。
   - ✓ response
     - 儲存另一個 JSON 格式物件，可跟隨所需資料擴充裡面的值。

圖 9：MemberController 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 Servlet 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。
5. 回傳之資料透過 JsonReader()內之 method 封裝成 JSONObject 物件（同為 JSON 格式）進行回傳，同時回傳之物件帶有狀態碼之資料。

6.4 專案資料夾架構

下圖（圖 10）為本系統之專案資料夾整體架構：

1. 根目錄主要存放 Eclipse 相關設定檔案、git 相關檔案、View 之相關文件（包含 HTML 等）、 網站之靜態文件（CSS、image 等）則存放於 statics 資料夾當中。
2. WEB-INF 則存放 Controller 與 Model 之後端檔案：
   A. web.xml：存放網站之路徑（route）資料
   B. lib 則存放第三方套件，classes 則存放相關的類別（包含所有*.java 和 *.class），同時依照其需求將不同應用程式以不同 package 區分。
      i. 共同使用之 tools package：包含 JSONReader.class 等。
      ii. 本範例檔案之 demo package：又細分為 app、controller 和 util
   C. 網站上線時，須將所有*.java 檔案編譯成*.class。
   D. 相關之編譯指令請參閱先前所論述之小節（5.2.2.2 商業邏輯層（Business Logic Layer））。
3. doc 資料夾則是存放 javadoc 相關之文件，本資料夾內所有文件係由所有 java 程式之 java doc comment 所自動產生，其檔案可由瀏覽器所開啟。

```
NCU_MIS_SA
|   .classpath
|   .gitignore
|   .project
|   .tomcatplugin
|   edit.html
|   index.html
|   register.html
|
+---.settings
|       .jsdtscope
|       org.eclipse.core.resources.prefs
|       org.eclipse.jdt.core.prefs
|       org.eclipse.wst.common.component
|       org.eclipse.wst.common.project.facet.core.xml
|       org.eclipse.wst.jsdt.ui.superType.container
|       org.eclipse.wst.jsdt.ui.superType.name
|
+---doc
|
+---statics
|   +---css
|   |
|   +---icon
|   |
|   +---img
|   |
|   \---js
|
\---WEB-INF
    |   web.xml
    |
    +---classes
    |   \---ncu
    |       |   .gitignore
    |       |
    |       \---im3069
    |           |   .gitignore
    |           |
    |           +---demo
    |           |   +---app
    |           |   |       .gitignore
    |           |   |       Member.class
    |           |   |       Member.java
    |           |   |       MemberHelper.class
    |           |   |       MemberHelper.java
    |           |   |
    |           |   +---controller
    |           |   |       .gitignore
    |           |   |       MemberController.class
    |           |   |       MemberController.java
    |           |   |
    |           |   \---util
    |           |           .gitignore
    |           |           DBMgr.class
    |           |           DBMgr.java
    |           |
    |           \---tools
    |                   .gitignore
    |                   JsonReader.class
    |                   JsonReader.java
    |
    \---lib
            json-20180813.jar
            mysql-connector-java-8.0.17.jar
            servlet-api.jar
```

圖 10：專案之資料夾架構

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。

表 5：Route 表格

Index	Route	Action	網址參數	功能描述/作法
1	/index.html	GET	None	首頁（檢視所有會員）
2	/edit.html	GET	id	管理者編輯會員資料頁面
3	/register.html	GET	None	管理者註冊會員頁面
4	/api/member.do	GET	None	取得會員資料
5	/api/member.do	POST	None	新增會員
6	/api/member.do	DELETE	None	刪除會員
7	/api/member.do	PUT	None	更新會員

下列將根據上表進行更詳細之說明與其運行步驟：

1. /index.html：取得首頁（render page），運行步驟如下：
   A. 進入頁面，透過 AJAX 發送不帶參數之 GET 請求。
   B. 將取回之資料庫會員資料 Render 至表格當中。
   C. 將取回之所下 SQL 指令與花費時間更新至表格當中。
2. /edit.html：取得會員資料更新頁面（render page），運行步驟如下：
   A. 取得網址所傳入之會員編號（id）參數
   B. 透過 AJAX 發送帶參數之 GET 請求。
   C. 將取回資料回填至原先欄位。
3. /register.html：取得註冊頁面（render page）
4. /api/member.do：取得會員資料之 API，運行步驟如下：
   A. 前端可選擇是否傳入會員編號參數（id）。
   B. 若不帶參數則表示為請求資料庫所有會員資料。
   C. 帶參數則表示為請求資料庫中該名會員資料。
   D. 回傳取得結果
5. /api/member.do：註冊會員之 API，運行步驟如下：
   A. 前端傳入 email、name、password 之 JSON 物件。
   B. 確認帳號是否重複，若重複則回傳錯誤資訊

C. 建立 member

D. 回傳註冊成功資訊

6. /api/member.do：刪除會員之 API，運行步驟如下：

A. 前端傳入會員編號之 JSON 物件。

B. 刪除會員資料

C. 回傳刪除成功訊息

7. /api/member.do：更新會員資料之 API，運行步驟如下：

A. 前端傳入更新後之 email、name、password 之 JSON 物件。

B. 更新 member 資料。

C. 回傳更新成功訊息。

6.6 程式碼版本控制（參考用）

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/jason40418/ncu_mis_sa。

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。

本專案所需要 import 之項目為下圖（圖 11）所示，若需要額外 import 不同的 package 物件請如同第 3 行方式進行 import。

```
1 // 操作Controller
2 // import專案所需之Controller package和JsonReader
3 package ncu.im3069.demo.controller;
4 import ncu.im3069.tools.JsonReader;
5
6 // 操作Controller
7 // import servlet所需
8 import java.io.*;
9 import java.util.*;
10 import javax.servlet.*;
11 import javax.servlet.http.*;
12
13 // 操作JSON相關物件
14 import org.json.*;
15
16 // 操作JDBC資料庫相關
17 import java.sql.*;
18
19
```

圖 11：本專案所必須 import

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」。

圖 12：常見之 JSON 格式範例

上圖（圖 12）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。

於後續章節說明使用 JsonReader 時，必須對於 JSON 之格式有明確之了解，以在後端取回 Request 之 JSON 資料不會取值錯誤導致資料無法存取。

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 JQuery 之 AJAX 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等透過 JSON 和 JQuery 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）之資料驗證依憑 JavaScript 之正規表達式（Regular Expression）進行，並以 alert()方式通知使用者錯誤之訊息，如下圖（圖 13）為例：

```javascript
function submit() {
    var name = $('#member_name').val();
    var email = $('#member_email').val();
    var password = $('#member_password').val();
    var email_rule = /^\w+((-\w+)|(\.\w+))*\@[A-Za-z0-9]+((\.|-)[A-Za-z0-9]+)*\.[A-Za-z]+$/;
    var password_rule = /^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{8,}$/;
    if (!email_rule.test(email)) {
        alert("Email格式不符！");
    }
    else if(!password_rule.test(password)) {
        alert("密碼格式不符，長度至少8，且至少包含一個數字和英文字母！");
    }
    else {
        // 將資料組成JSON格式
        var data_object = {
            "name": name,
            "email": email,
            "password": password
        };
        // 將JSON格式轉換成字串
        var data_string = JSON.stringify(data_object);
        // 發出POST的AJAX請求
        $.ajax({
                type: "POST",
                url: "api/member.do",
                data: data_string,
                crossDomain: true,
                cache: false,
                dataType: 'json',
                timeout: 5000,
                success: function (response) {
                    $('#flashMessage').html(response.message);
                    $('#flashMessage').show();
                    if(response.status == 200){
                        updateSQLTable(response.response);
                    }
                },
                error: function () {
                    alert("無法連線到伺服器！");
                }
        });
    }
}
```

圖 13：註冊會員之程式碼

3. url：指定之 API 路徑。
4. method：需依照 API 指定 GET/POST/DELETE/PUT。
5. data：傳送資料須以 JSON.stringfy (data_object) 打包成 JSON 格式。
6. dataType：需指定回傳格式為 JSON。
7. timeout：設定 AJAX 最多等待時間，避免檢索時間過久。

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 14）顯示會員更新之欄位，根據檢所之結果進行回填方式：

```
if(response.status == 200){
    updateSQLTable(response.response);
    document.getElementById('member_name').value = response['response']['data'][0]['name'];
    document.getElementById('member_email').value = response['response']['data'][0]['email'];
    document.getElementById('member_password').value = response['response']['data'][0]['password'];
    document.getElementById('member_login_times').value = response['response']['data'][0]
['login_times'];
    document.getElementById('member_status').value = response['response']['data'][0]['status'];
}
```

圖 14：更新會員欄位回填

若查詢之資料要以表格方式呈現，可先將表格之 tbody 部分進行清空，爾後使用字串方式組成 table 之元素，最終使用 append()之 method 將元素回填，下圖（圖 15）以更新 SQL 查詢結果之表格為例：

```
function updateSQLTable(data) {
    $("#sql_log > tbody").empty();
    var time = (data.time / 1000000).toFixed(2);
    var table_html = "";
    table_html += '<tr>';
    table_html += '<td>' + '1' + '</td>';
    table_html += '<td>' + data.sql + '</td>';
    table_html += '<td style="text-align: right">' + '0' + '</td>';
    table_html += '<td style="text-align: right">' + data.row + '</td>';
    table_html += '<td style="text-align: right">' + data.row + '</td>';
    table_html += '<td style="text-align: right">' + time + '</td>';
    table_html += '</tr>';
    $("#sql_log > tbody").append(table_html);
    $("#sql_summary").html("(default) " + data.row + " queries took " + time + " ms");
}
```

圖 15：更新 SQL 查詢結果之表格

7.2 JsonReader、JSONObject 與 JSONArray 操作

為簡化取回前端所傳入之 JSON 資料，本範例提供 JsonReader class 用於協助處理，為使用此 class 需要將該 JsonReader.java 檔案放置，並且將整個 java 專案資料夾 import（import ncu.im3069.tools.JsonReader;），若要操作 JSONObject 則必須 import（import org.json.*;），程式碼範例如下圖（圖 16）所示：

```java
import ncu.im3069.tools.JsonReader;
import org.json.*;

/** 透過JsonReader類別將Request之JSON格式資料解析並取回 */
JsonReader jsr = new JsonReader(request);
JSONObject jso = jsr.getObject();

/** 取出經解析到JSONObject之Request參數 */
String email = jso.getString("email");
String password = jso.getString("password");
String name = jso.getString("name");

/** 方法一：以字串組出JSON格式之資料 */
String resp = "{\"status\": \'400\', \"message\": \'欄位不能有空值\', \'response\': \'\'}";
jsr.response(resp, response);


/** 方法二：新建一個JSONObject用於將回傳之資料進行封裝 */
JSONObject resp = new JSONObject();
resp.put("status", "200");
resp.put("message", "成功! 註冊會員資料...");
resp.put("response", data);
jsr.response(resp, response);
```

圖 16：JsonReader 操作之範例

下表（表 6）將呈現 JSONObejct 和 JSONArray 之取值方法，更多範例與使用實例可參閱 MemberController.java 主要取值必須要有對應的 key 進行一層層取出 value。

表 6：JSONObject 和 JSONArray 之操作範例

<table>
<tr>
<td>

```
// 取回
{
  "name"    : "test",
  "age"     : 20,
  "major"   : ["資訊管理", "企業管理"],
  "other"   : {
      "birthday"    : "1998-01-01",
      "like"        : "dance"
  }
}
```

</td>
<td>

```
JSONObject o = new JSONObject(str);
System.out.println(o.getString("name"));
System.out.println(o.getInt("age"));
System.out.println(o.getJSONArray("major"));
System.out.println(o.getJSONObject("other"));
```

</td>
</tr>
<tr>
<td>範例 JSON 格式</td>
<td>範例取出 JSONObject 之內容</td>
</tr>
<tr>
<td>

```
// JSONObject操作
// 新增物件
JSONObject jsonObject = new JSONObject();
// 放入key-value
jsonObject.put("UserName", "ZHULI");
// 放入Array
jsonObject.element("Array", arrayList);
```

</td>
<td>

```
// JSONArray操作
// 新增物件
JSONArray jsonArray = new JSONArray();
// 放入key-value
jsonArray.add(0, "ZHULI");
// 印出
print：System.out.println("jsonArray1："   jsonArray);
```

</td>
</tr>
<tr>
<td>JSONObject 之操作方式</td>
<td>JSONArray 之操作方式</td>
</tr>
</table>

7.3 DBMgr 與 JDBC

此部分之 class 可自行增加功能。

在 Java 當中，本專案使用 JDBC 用以連線資料庫進行操作，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 DBMgr 之類別進行封裝，以下節錄說明 DBMgr 之實作方式，詳細之內容可參閱 DBMgr 類別。

以下圖（圖 18）取得所有資料庫會員為例，為使用 JDBC 之資料庫操作，需要 import（import java.sql.*）同時需要宣告固定的資料庫參數組，並且需要使用 try-catch 方式進行，其詳細步驟如下：

A. 圖 18 第 16 行透過 DBMgr.getConnection()建立連線。

B. 圖 18 第 21 行將愈查詢之 SQL 指令以 prepareStatement()方式儲存，若 SQL 指令有參數則以「?」進行放置，如下圖（圖 17）所示，並以 setString()、setInt()等方式回填（詳細可設定之參數格式請參閱官方文件：https://docs.oracle.com/javase/7/docs/api/java/sql/PreparedStatement.html）。

C. 圖 18 第 23 行若為檢索則是 executeQuery()，回傳為 ResultSet、其餘指令如圖 17 第 33 行為 executeUpdate()回傳為 int 整數，影響之行數。

D. 圖 18 第 12 和 23 行檢索後透過 ResultSet 將結果儲存，圖 18 第 30 行並以 while 迴圈移動指標，將每筆查詢結果依序進行操作。

```java
/** 記錄實際執行之SQL指令 */
String exexcute_sql = "";
/** 紀錄程式開始執行時間 */
long start_time = System.nanoTime();
/** 紀錄SQL總行數 */
int row = 0;

try {
    /** 取得資料庫之連線 */
    conn = DBMgr.getConnection();
    /** SQL指令 */
    String sql = "INSERT INTO `missa`.`members`(`name`, `email`, `password`, `modified`, `created`, `login_times`, `status`)"
            + " VALUES(?, ?, ?, ?, ?, ?, ?)";

    /** 取得所需之參數 */
    String name = m.getName();
    String email = m.getEmail();
    String password = m.getPassword();
    int login_times = m.getLoginTimes();
    String status = m.getStatus();

    /** 將參數回填至SQL指令當中 */
    pres = conn.prepareStatement(sql);
    pres.setString(1, name);
    pres.setString(2, email);
    pres.setString(3, password);
    pres.setTimestamp(4, Timestamp.valueOf(LocalDateTime.now()));
    pres.setTimestamp(5, Timestamp.valueOf(LocalDateTime.now()));
    pres.setInt(6, login_times);
    pres.setString(7, status);

    /** 執行新增之SQL指令並記錄影響之行數 */
    row = pres.executeUpdate();

    /** 紀錄真實執行的SQL指令，並印出 **/
    exexcute_sql = pres.toString();
    System.out.println(exexcute_sql);

} catch (SQLException e) {
    /** 印出JDBC SQL指令錯誤 **/
    System.err.format("SQL State: %s\n%s\n%s", e.getErrorCode(), e.getSQLState(), e.getMessage());
} catch (Exception e) {
    /** 若錯誤則印出錯誤訊息 */
    e.printStackTrace();
} finally {
    /** 關閉連線並釋放所有資料庫相關之資源 **/
    DBMgr.close(pres, conn);
}
```

圖 17：新增會員之 MemberHelper create() method（節錄）

```java
/** 新建一個 Member 物件之 m 變數，用於紀錄每一位查詢回之會員資料 */
Member m = null;
/** 用於儲存所有檢索回之會員，以JSONArray方式儲存 */
JSONArray jsa = new JSONArray();
/** 記錄實際執行之SQL指令 */
String exexcute_sql = "";
/** 紀錄程式開始執行時間 */
long start_time = System.nanoTime();
/** 紀錄SQL總行數 */
int row = 0;
/** 儲存JDBC檢索資料庫後回傳之結果，以 pointer 方式移動到下一筆資料 */
ResultSet rs = null;

try {
    /** 取得資料庫之連線 */
    conn = DBMgr.getConnection();
    /** SQL指令 */
    String sql = "SELECT * FROM `missa`.`members`";

    /** 將參數回填至SQL指令當中，若無則不用只需要執行 prepareStatement */
    pres = conn.prepareStatement(sql);
    /** 執行查詢之SQL指令並記錄其回傳之資料 */
    rs = pres.executeQuery();

    /** 紀錄真實執行的SQL指令，並印出 **/
    exexcute_sql = pres.toString();
    System.out.println(exexcute_sql);

    /** 透過 while 迴圈移動pointer，取得每一筆回傳資料 */
    while(rs.next()) {
        /** 每執行一次迴圈表示有一筆資料 */
        row += 1;

        /** 將 ResultSet 之資料取出 */
        int member_id = rs.getInt("id");
        String name = rs.getString("name");
        String email = rs.getString("email");
        String password = rs.getString("password");
        int login_times = rs.getInt("login_times");
        String status = rs.getString("status");

        /** 將每一筆會員資料產生一名新Member物件 */
        m = new Member(member_id, email, password, name, login_times, status);
        /** 取出該名會員之資料並封裝至 JSONsonArray 內 */
        jsa.put(m.getData());
    }

} catch (SQLException e) {
    /** 印出JDBC SQL指令錯誤 **/
    System.err.format("SQL State: %s\n%s\n%s", e.getErrorCode(), e.getSQLState(), e.getMessage());
} catch (Exception e) {
    /** 若錯誤則印出錯誤訊息 */
    e.printStackTrace();
} finally {
    /** 關閉連線並釋放所有資料庫相關之資源 **/
    DBMgr.close(rs, pres, conn);
}
```

圖 18：檢索所有會員之 MemberHelper getAll() method（節錄）

SQL 資料庫之操作，大致上以下圖（圖 19）為模板進行，未來若要寫作其他 method 亦可以此為藍圖進行發展，並依照需求進行修改。

```java
/** 記錄實際執行之SQL指令 */
String exexcute_sql = "";
/** 紀錄SQL總行數 */
int row = 0;
/** 儲存JDBC檢索資料庫後回傳之結果，以 pointer 方式移動到下一筆資料 */
ResultSet rs = null;

try {
    /** 取得資料庫之連線 */
    conn = DBMgr.getConnection();
    /** SQL指令 */
    String sql = "";

    /** 將參數回填至SQL指令當中，若無則不用只需要執行 prepareStatement */
    pres = conn.prepareStatement(sql);
    /** 回填參數 */
  pres.setString({location}, {varialbe})

    /** 若為查詢之外指令則回傳為影響行數 */
    row = pres.executeUpdate();

    /** 執行查詢之SQL指令並記錄其回傳之資料 */
    rs = pres.executeQuery();

    /** 紀錄真實執行的SQL指令，並印出 **/
    exexcute_sql = pres.toString();

    /** 透過 while 迴圈移動pointer，取得每一筆回傳資料 */
    while(rs.next()) {
        /** 每執行一次迴圈表示有一筆資料 */
        row += 1;
    }

} catch (SQLException e) {
    /** 印出JDBC SQL指令錯誤 **/
    System.err.format("SQL State: %s\n%s\n%s", e.getErrorCode(), e.getSQLState(), e.getMessage());
} catch (Exception e) {
    /** 若錯誤則印出錯誤訊息 */
    e.printStackTrace();
} finally {
    /** 關閉連線並釋放所有資料庫相關之資源 **/
    DBMgr.close(rs, pres, conn);
}
```

圖 19：操作 JDBC 之模板