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

第五組

資管三Ｂ　109403048　林子恩

資管三Ｂ　109403533　林采璇

資管三Ｂ　109403535　洪琬哲

資管三Ｂ　109403538　江珮彤

資管三Ｂ　109403543　賴立彬

資管三Ｂ　109403546　謝諶煜

指導教授：葉羅堯　教授

中 華 民 國　１１２　年　１　月　６　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
V1.0.0	全員	Draft	2023/01/06

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 7 個實體（Entity）、6 個關係（Relationship）、2 個複合性實體（Compound Entity）、1 個弱實體（Weak 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：

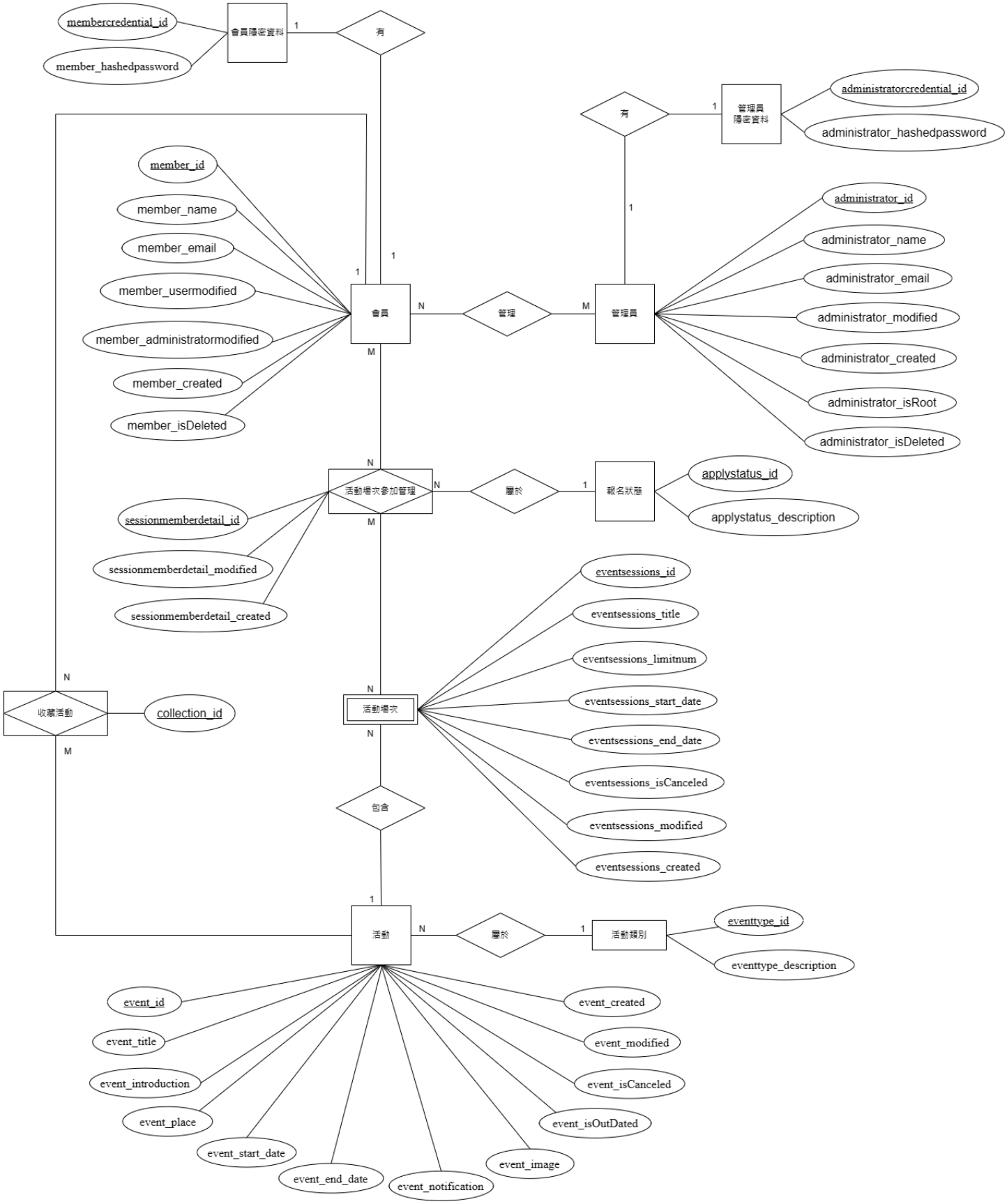

圖 1：設計階段之實體關係圖

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， 本範例由於僅實作後台管理者會員模組，因此資料表僅就會員與商品進行呈 現，實際上仍需將所有資料表呈現於此：

1. 會員資料表（TB_Member）：

表 1：會員資料表（TB_Member）之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K.	member_id	Int(11)	無	否	V	
	member_name	Varchar(30)	無	否		utf8mb4_0900_ai_ci
	member_email	Varchar(50)	無	否		utf8mb4_0900_ai_ci
	member_usermodified	Datetime	無	否		
	member_administratormodified	Datetime	無	否		
	member_created	Datetime	無	否		
	member_isDeleted	TinyInt	無	否		

- ✓ member_member_id：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。
- ✓ member_usermodified：用於記錄會員更動會員資料的最後更新時間。
- ✓ member_administratormodified：用於記錄管理員更動會員資料的最後更新時間。
- ✓ member_created：用於記錄該名會員註冊的時間。
- ✓ member_isDeleted：用於記錄該名會員是否被刪除，0 為未刪除、1 為已刪除，已刪除的會員無法再登入。

2. 會員隱密資料表（TB_MemberCredential)：

表 2：會員隱密資料表（TB_MemberCredential)之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K.	membercredential_id	Int(11)	無	否	V	
F.K.	member_id	Int(11)	無	否		
	member_hashedpassword	Varchar(50)	無	否		utf8mb4_0900_ai_ci

- ✓ membercredential_id：為自動增加作為會員隱密資料編號，不可更動由資料庫系統自動產生。
- ✓ member_hashedpassword：經由 bcrypt 加密後生成的密碼。

3. 管理員資料表（TB_Administrator）：

表 3：管理員資料表（TB_Administrator）之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K.	administrator_id	Int(11)	無	否	V	
	administrator_name	Varchar(30)	無	否		utf8mb4_0900_ai_ci
	administrator_email	Varchar(50)	無	否		utf8mb4_0900_ai_ci
	administrator_modified	Datetime	無	否		
	administrator_created	Datetime	無	否		
	administrator_isRoot	TinyInt	無	否		
	administrator_isDeleted	TinyInt	無	否		

- ✓ administrator_id：為自動增加作為管理員編號，不可更動由資料庫系統自動產生。
- ✓ administrator_modified：用於記錄管理員更動管理員資料的最後更新時間。
- ✓ administrator_created：用於記錄該名管理員創建的時間。
- ✓ administrator_isRoot：用於記錄該名管理員是否為 root 管理員，0 為否、1 為是，root 管理員可以執行新增、修改、刪除、檢視其他管理員。
- ✓ administrator_isDeleted：用於記錄該名管理員是否被刪除，0 為未刪除、1 為已刪除，已刪除的管理員無法再登入。

4. 管理員隱密資料表（TB_AdministratorCredential）：

表 4：管理員隱密資料表（TB_AdministratorCredential）之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K.	administratorcredential_id	Int(11)	無	否	V	
F.K.	administrator_id	Int(11)	無	否		
	administrator_hashedpassword	Varchar(50)	無	否		utf8mb4_0900_ai_ci

- ✓ administratorcredential_id：為自動增加作為管理員隱密資料編號，不可更動由資料庫系統自動產生。
- ✓ administrator_hashedpassword：經由 bcrypt 加密後生成的密碼。

5. 活動資料表（TB_Event）：

表 5：活動資料表（TB_Event）之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K.	event_id	Int(11)	無	否	V	
	event_title	Varchar(30)	無	否		utf8mb4_0900_ai_ci
F.K.	eventtype_id	Int(11)	無	否		
	event_introduction	Varchar()	無	否		utf8mb4_0900_ai_ci
	event_place	Varchar(50)	無	否		utf8mb4_0900_ai_ci
	event_start_date	Datetime	無	否		
	event_end_date	Datetime	無	否		
	event_notification	Varchar()	NULL	是		utf8mb4_0900_ai_ci
	event_image	Varchar()	NULL	是		utf8mb4_0900_ai_ci
	event_isOutDated	TinyInt	無	否		
	event_isCanceled	TinyInt	無	否		
	event_modified	Datetime	無	否		
	event_created	Datetime	無	否		

- ✓ event_id：為自動增加作為活動編號，不可更動由資料庫系統自動產生。
- ✓ event_image：用於紀錄商品圖檔之路徑位置。
- ✓ event_isOutDated：用於記錄該活動是否已過期，0 為否、1 為是，若活動已過期，則會員無法再報名、管理員可以刪除過期的活動，若管理員沒有刪除，則過期的活動仍然會顯示在頁面上。
- ✓ event_isCanceled：用於記錄該活動是否被取消，0 為否、1 為是，若活動被取消則會員無法再報名。
- ✓ event_modified：用於記錄管理員更動活動內容的最後更新時間。
- ✓ event_created：用於記錄活動被創建的時間。

6. 活動類別資料表（TB_EventType）：

表 6：活動類別資料表（TB_EventType）之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K.	eventtype_id	Int(11)	無	否	V	
	eventtype_description	Varchar(10)	無	否		utf8mb4_0900_ai_ci

- ✓ eventtype_id：為自動增加作為活動類別編號，不可更動由資料庫系統自動產生。
- ✓ eventtype_description：1:演講座談、2:藝文特展、3:音樂舞台、4:107 影享會、5:工作坊、6:崑曲展演、7:其他。

7. 活動場次資料表（TB_EventSessions）：

表 7：活動場次資料表（TB_EventSessions）之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K.	eventsessions_id	Int(11)	無	否	V	
F.K.	event_id	Int(11)	無	否		
	eventsessions_title	Varchar(30)	無	否		utf8mb4_0900_ai_ci
	eventsessions_limitnum	Int(11)	無	否		
	eventsessions_start_date	Datetime	無	否		
	eventsessions_end_date	Datetime	無	否		
	eventsessions_isCanceled	TinyInt	無	否		
	eventsessions_modified	Datetime	無	否		
	eventsessions_created	Datetime	無	否		

- ✓ eventsessions_id：為自動增加作為活動場次編號，不可更動由資料庫系統自動產生。
- ✓ eventsessions_isCanceled：用於記錄該活動場次是否被取消，0 為否、1 為是，若活動場次被取消，則會員無法再報名。
- ✓ eventsessions_modified：用於記錄管理員更動活動場次內容的最後更新時間。
- ✓ eventsessions_created：用於記錄活動場次被創建的時間。

8. 活動場次參加管理資料表(TB_SessionMemberDetail)：

表 8：活動場次參加管理資料表(TB_SessionMemberDetail)之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K.	sessionmemberdetail_id	Int(11)	無	否	V	
F.K.	member_id	Int(11)	無	否		
F.K.	eventsessions_id	Int(11)	無	否		
	sessionmemberdetail_modified	Datetime	無	否		
	sessionmemberdetail_created	Datetime	無	否		
F.K.	applystatus_id	Int(11)	無	否		

- ✓ sessionmemberdetail_id：為自動增加作為活動場次參加管理編號，不可更動由資料庫系統自動產生。
- ✓ sessionmemberdetail_modified：用於記錄會員更動活動場次內容的最後更新時間。
- ✓ sessionmemberdetail_created：用於記錄會員報名該活動場次的時間。

9. 活動報名狀態資料表(TB_ApplyStatus)：

表 9：活動報名狀態資料表(TB_ApplyStatus)之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K.	applystatus_id	Int(11)	無	否	V	
	applystatus_description	Varchar(10)	無	否		utf8mb4_0900_ai_ci

- ✓ applystatus_id：為自動增加作為活動報名狀態編號，不可更動由資料庫系統自動產生。
- ✓ applystatus_description：1:報名成功、2:取消報名、3:刪除紀錄。

10. 收藏資料表（TB_Collection）：

表 10：收藏資料表（TB_Collection）之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K.	collection_id	Int(11)	無	否	V	
F.K.	event_id	Int(11)	無	否		
F.K.	member_id	Int(11)	無	否		

- ✓ collection_id：為自動增加作為收藏編號，不可更動由資料庫系統自動產生。

第 3 章 類別圖

下圖（圖 2、圖 3、圖 4）係依據藝文中心報名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）。

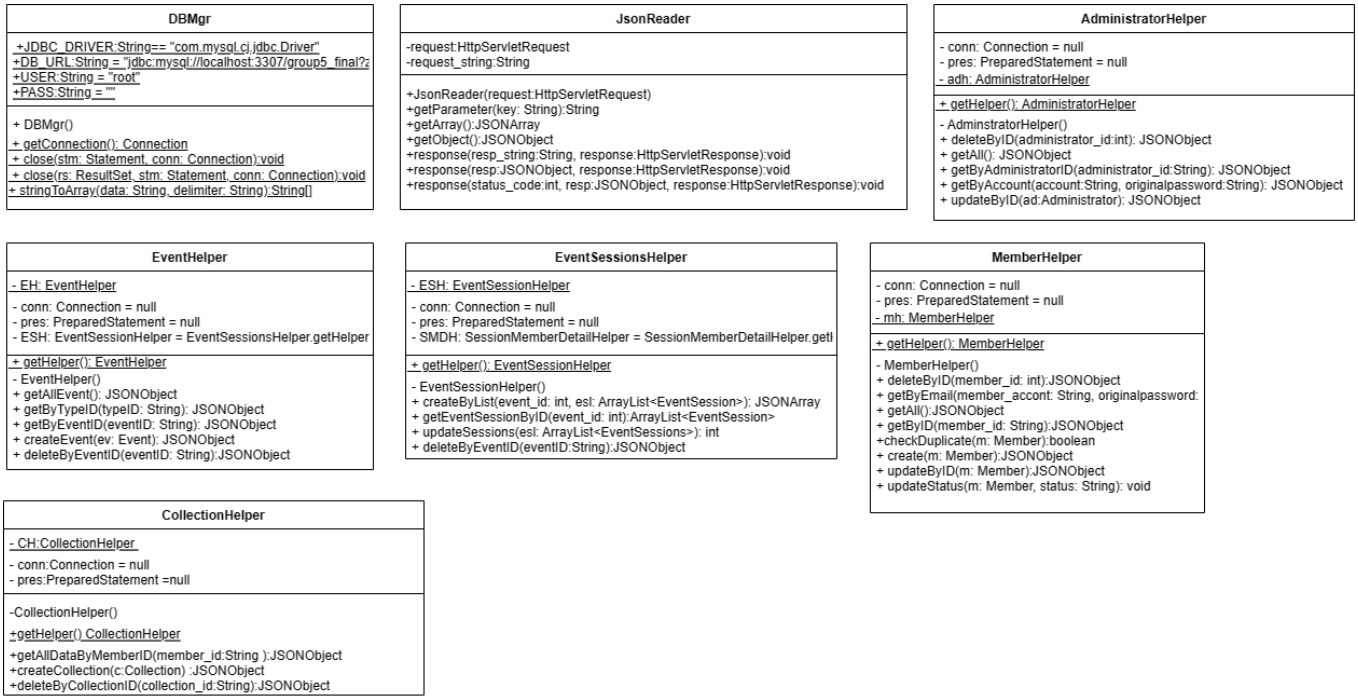

圖 2：類別圖(1/3)


TB_Member
- member_id:int
- member_name:String
- member_email:String
- member_hashedpassword:String
- member_usermodified:String
- member_adminstratormodified:String
- member_created:String
- member_isDeleted:int
- checkpw:String
- mh:MemberHelper
+ Member(member_id:int, member_hashedpassword:String)
+ Member(member_id:int, member_name:String, member_email:String)
+ Member(member_id:int, member_name:String, member_email:String, member_hashedpassword:String, member_isDeleted:int, checkpw:String)
+ Member(member_name:String, member_email:String, member_hashedpassword:String)
+ Member(member_id:int, member_email:String, member_hashedpassword:String, checkpw
+ Member(member_id:int, member_name:String, member_email:String, member_hashedpassword:String, member_isDeleted:int)
+ getData():JSONObject
+ getLoginData():JSONObject
+ getID():int
+ getEmail():String
+ getName():String
+ getMemberhashedpassword():String
+ getUserModified():String
+ getAdministratorModified():String
+ getCreated():String
+ getIsDeleted():int
+ getCheckpw():String
TB_Administrator
- administrator_id:int
- name:String
- email:String
- hashedpassword:String
- modified:String
- created:String
- isRoot:int
- isDeleted:int
- checkpw:String
- adh:AdministratorHelper
+ Administrator(administrator_id:int, name:String, email:String, isRoot:int, hashedpassword:String)
+ Administrator(administrator_id:int, name:String, email:String, hashedpassword:String, checkpw:String)
+ Administrator(administrator_id:int, name:String, email:String)
+ getData():JSONObject
+ getLoginData():JSONObject
+ getAdministratorID():int
+ getAdministratorName():String
+ getAdministraterEmail():String
+ getAdministratorHashedpassword():String
+ getAdministratorModified():String
+ getAdministratorCreated():String
+ getAdministratorIsRoot():int
+ getAdministratorIsDeleted():int
+ getCheckpw():String
1...*
1...*
1
1
0...*
TB_Collection
- collection_id:int
- event_id:int
- member_id:int
+ Collection(event_id:int, member_id:int)
+ setCollectionID(collection_id:int):void
+ getCollectionID():int
+ getEventID():int
+ getMemberID():int
0...*
1
TB_Event
- event_id:int
- title:String
- eventtype_id:int
- introduction:String
- place:String
- start_date:Timestamp
- end_date:Timestamp
- notification:String
- imagePath:String
- isOutDated:byte
- isCanceled:byte
- modified:Timestamp
- created:Timestamp
- SessionList:ArrayList<EventSessions>
- ESH:EventSessionHelper
+ Event(event_id:int, title:String, eventtype_id:int, introduction:String, place:String, start_date:Timestamp, end_date:Timestamp, notification:String, imagePath:String, isOutDated:byte, isCanceled:byte, modified:Timestamp, created:Timestamp)
+ Event(title:String, eventtype_id:int, introduction:String, place:String, start_date:Timestamp, end_date:Timestamp, notification:String, imagePath:String)
+ Event(event_id:int, title:String, eventtype_id:int, introduction:String, place:String, start_date:Timestamp, end_date:Timestamp, notification:String, imagePath:String, isCanceled:byte, modified:Timestamp)
+ addEventSessions(es:EventSessions):void
+ setEventID(event_id:int):void
+ getEventID():int
+ getTitle():String
+ getEventTypeID():int
+ getIntroduction():String
+ getPlace():String
+ getStartDate():Timestamp
+ getEndDate():Timestamp
+ getNotification():String
+ getImagePath():String
+ checkOutDated():byte
+ checkCanceled():byte
+ getModifiedDate():Timestamp
+ getCreatedDate():Timestamp
+ setEventSessionsID(data:JSONArray):void
+ getEventSessions():ArrayList<EventSessions>
+ getEventData():JSONobject
+ getSessionListData():JSONArray
+ getAllData():JSONobject
1
1...*
0...*
TB_SessionsMemberDetail
- sessionmemberdetail_id:int
- member_id:int
- eventsessions_id:int
- modified;:Timestamp
- created:Timestamp
- applystatus_id:int
+ SessionMemberDetail(sessionmemberdetail_id:int, member_id:int, eventsessions_id:int, modified:Timestamp, created:Timestamp, applystatus_id :int)
+ SessionMemberDetail(member_id:int, eventsessions_id:int, applystatus_id:int)
+ getSessionMemberDetailID():int
+ getMemberID():int
+ getEventSessionsID():int
+ getModified():Timestamp
+ getCreated():Timestamp
+ getApplyStatusID():int
+ getSessionMemberDetailData():JSONObject
0...*
1
TB_EventSessions
- eventsessions_id:int
- event_id:int
- title:String
- limitnum:int
- start_date:Timestamp
- end_date:Timestamp
- isCanceled:byte
- modified:Timestamp
- created:Timestamp
+ EventSessions(eventsessions_id:int, event_id:int, title:String, limitnum:int, start_date:Timestamp, end_date:Timestamp, isCanceled:byte, modified:Timestamp, created:Timestamp)
+ EventSessions(title:String, limitnum:int, start_date:Timestamp, end_date:Timestamp, isCanceled:byte, modified:Timestamp, created:Timestamp)
+ EventSessions(eventsessions_id:int, title:String, limitnum:int, start_date:Timestamp, end_date:Timestamp, isCanceled:byte, modified:Timestamp)
+ setSessionID(evetsessions_id:int):void
+ getEventSessionID():int
+ getEventID():int
+ getTitle():String
+ getLimitNum():int
+ getStartDate():Timestamp
+ getEndDate():Timestamp
+ checkCanceled():byte
+ getModifiedDate():Timestamp
+ getCreatedDate():Timestamp
+ getEventSessionsData():JSONObject

圖 3：類別圖(2/3)

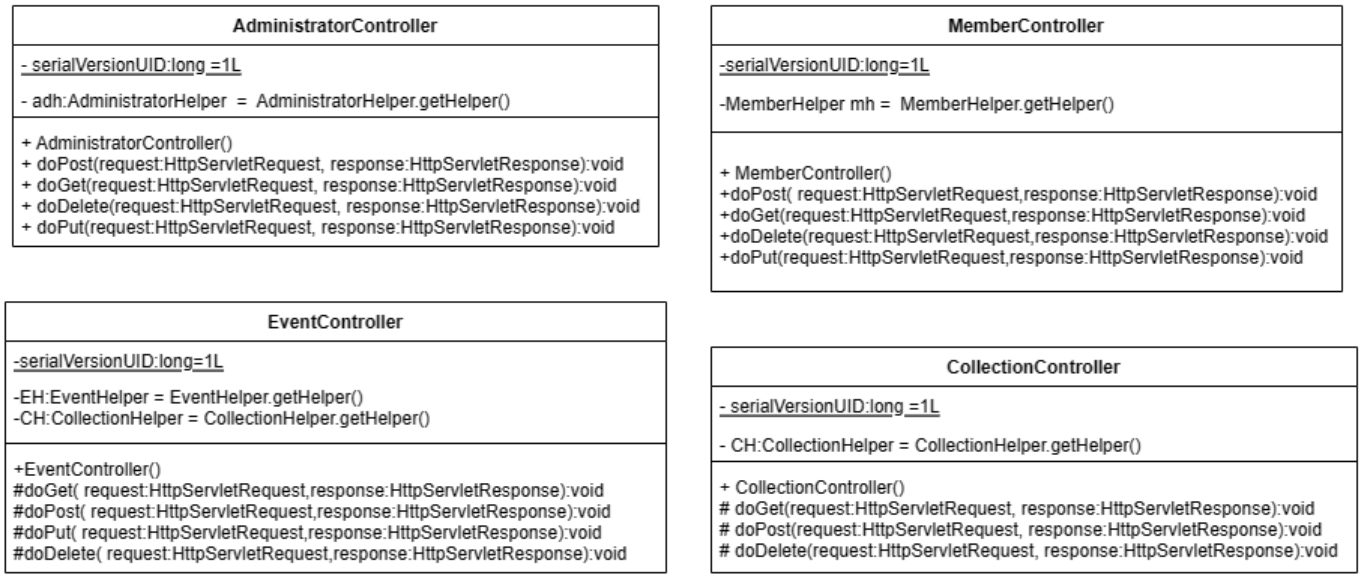
AdministratorController
- serialVersionUID:long =1L
- adh:AdministratorHelper = AdministratorHelper.getHelper()
+ AdministratorController()
+ doPost(request:HttpServletRequest, response:HttpServletResponse):void
+ doGet(request:HttpServletRequest, response:HttpServletResponse):void
+ doDelete(request:HttpServletRequest, response:HttpServletResponse):void
+ doPut(request:HttpServletRequest, response:HttpServletResponse):void
MemberController
-serialVersionUID:long=1L
-MemberHelper mh = MemberHelper.getHelper()
+ MemberController()
+doPost(request:HttpServletRequest,response:HttpServletResponse):void
+doGet(request:HttpServletRequest,response:HttpServletResponse):void
+doDelete(request:HttpServletRequest,response:HttpServletResponse):void
+doPut(request:HttpServletRequest,response:HttpServletResponse):void
EventController
-serialVersionUID:long=1L
-EH:EventHelper = EventHelper.getHelper()
-CH:CollectionHelper = CollectionHelper.getHelper()
+EventController()
#doGet(request:HttpServletRequest,response:HttpServletResponse):void
#doPost(request:HttpServletRequest,response:HttpServletResponse):void
#doPut(request:HttpServletRequest,response:HttpServletResponse):void
#doDelete(request:HttpServletRequest,response:HttpServletResponse):void
CollectionController
- serialVersionUID:long =1L
- CH:CollectionHelper = CollectionHelper.getHelper()
+ CollectionController()
# doGet(request:HttpServletRequest, response:HttpServletResponse):void
# doPost(request:HttpServletRequest, response:HttpServletResponse):void
# doDelete(request:HttpServletRequest, response:HttpServletResponse):void

圖 4：類別圖（3/3）

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，本活動報名系統預計共有 3 位動作者與 24 個使用案例，並依照不同之模組區分成不同子系統共計六個子系統，其中包含以下：① 會員子系統、② 活動資訊子系統、③ 活動收藏子系統、④ 活動參加管理子系統、⑤ 活動管理子系統、⑥管理員子系統，如下圖（圖 5）所示：

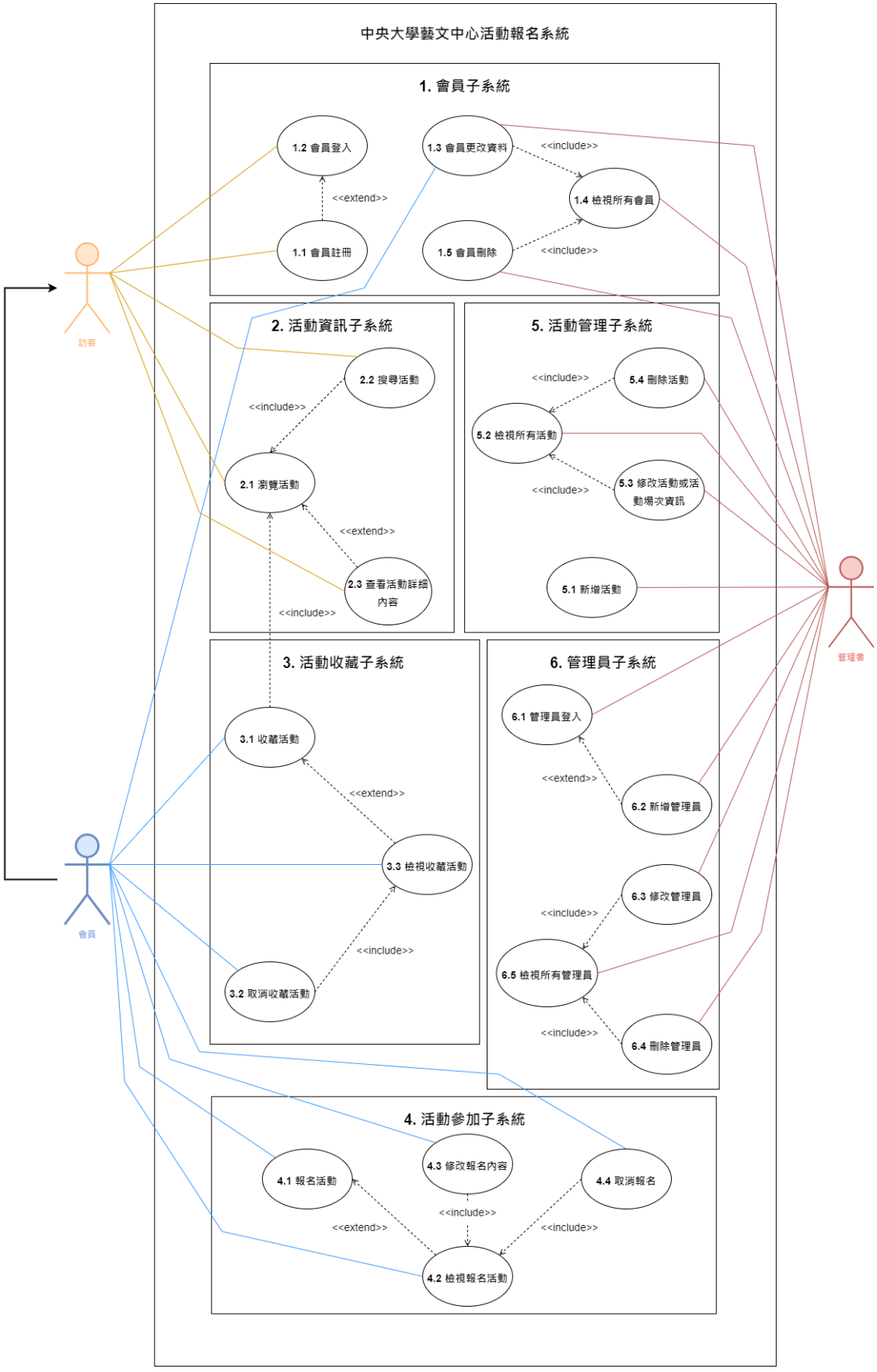

圖 5：中央大學藝文中心報名系統使用案例圖

4.2 Use Case 實做之循序圖

4.2.1 商業流程編號 4.0：活動參加管理模組

在 Use Case 4.0 中所使用到的功能（包含：報名活動、檢視活動報名紀錄、變更活動報名紀錄、取消報名），以下簡要說明各功能：

1. Use Case 4.1：報名活動

此使用案例用於說明會員可透過此功能對有興趣的活動進行報名。若為一般訪客，需進行會員註冊才可進行報名活動，僅限完成使用案例 1.2 會員登入的訪客或已登入之會員，才得以執行。此功能需要藉由使用者輸入所需欄位資料(日期、場次)，並且由系統檢查該活動人數是否已額滿，若已額滿則會回傳報名失敗訊息。

2. Use Case 4.2：檢視活動報名紀錄

此使用案例用於說明會員可透過此功能檢視自身的活動報名紀錄。若為一般訪客，需進行會員註冊才可進行報名活動，僅限完成使用案例 1.2 會員登入的訪客或已登入之會員，才得以執行。此功能會顯示該會員所有的活動報名紀錄，並且顯示與活動報名有關的相關資訊。

3. Use Case 4.3：會員變更活動報名紀錄

此使用案例用於說明會員變更自身的活動報名紀錄。若為一般訪客，需進行會員註冊才可進行報名活動，僅限完成使用案例 1.2 會員登入的訪客或已登入之會員，才得以執行。此功能僅能提供會員更動自身報名資料，無法對活動的相關資訊進行修改。

4. Use Case 4.4：會員取消報名

此使用案例用於該會員取消報名的活動。若為一般訪客，需進行會員註冊才可進行報名活動，僅限完成使用案例 1.2 會員登入的訪客或已登入之會員，才得以執行。此功能提供會員取消已報名的活動。

與該模組相關之頁面於下表（表 11）進行說明：

表 11：活動參加管理模組關聯頁面

HTML	關聯 Use Case	說明
register.html	Use Case 1.1	註冊會員頁面
index.html	Use Case 2.1	瀏覽活動
eventinfoprototype.html	Use Case 2.3	會員查看詳細活動內容
collect.html	Use Case 3.1 Use Case 3.2 Use Case 3.3	會員收藏活動相關功能
center-product-addNew.html	Use Case 5.1	管理員新增活動
manager-signup_List.html manager-activityList.html	Use Case 5.2	管理員檢視所有活動與檢視活動相關資訊
manager-activityList-editActivity	Use Case 5.3	管理員修改活動相關資訊

4.2.1.1 Sequence Diagram—Use Case 4.1 報名活動

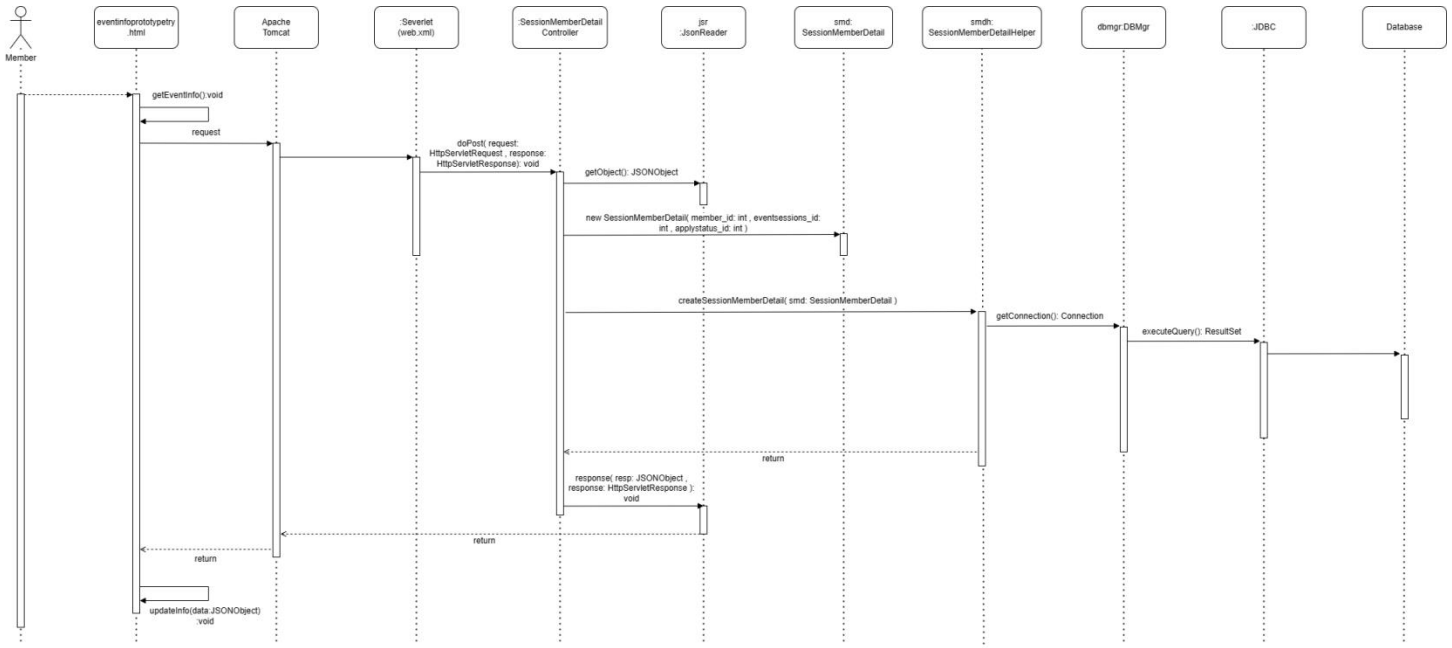

圖 6 ：商業流程編號 4.1 報名活動循序圖（取回資料）

1. 會員完成「1.2 會員登入」後，進入報名活動頁面（eventinfoprototypetry.html）。
2. 當會員完成場次選填並通過前端之資料驗證後，送出 POST 請求。
3. 後端以 SessionMemberDetailController 之 doPost()進行處理，以 JsonReader 取回 request 之參數。

4. 透過 SessionMemberDetailHelper 物件的 createSessionMemberDetail(smd: SessionMemberDetail) 建立報名資訊。

5. 回傳報名之結果後，若報名成功則透過 JavaScript 之 updateInfo() 更新 SQL 表格內之內容。

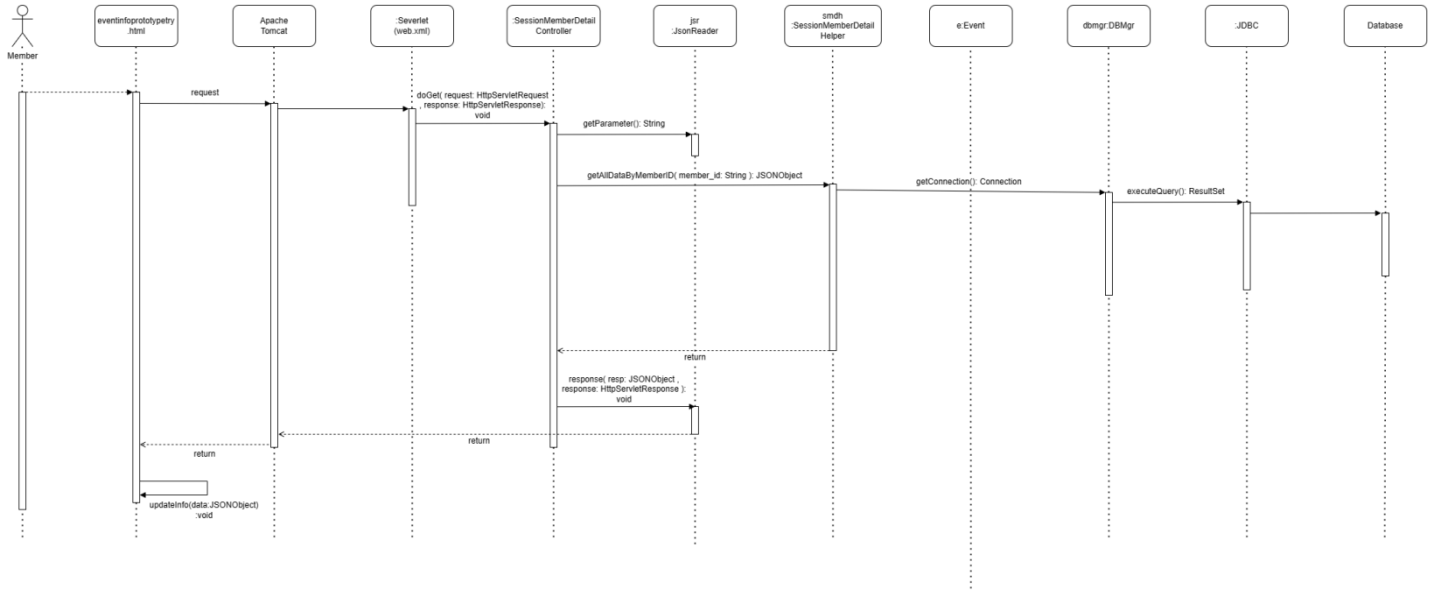

圖 7 ：商業流程編號 4.1 報名活動循序圖（更新資料）

1. 會員修改完成後，點擊「我要報名」按鈕並通過前端驗證後，送出 GET 請求。

2. 後端以 SessionMemberDetailController 之 doGet()進行處理，以 JsonReader 取回 Request 之 JSON 格式參數。

3. 透過 SessionMemberDetailHelper 物件的 getConnection() 取得活動場次報名相關資料。

4. 透過 JavaScript 之 updateInfo()更新 SQL 表格內之內容。

4.2.2 商業流程編號 5.0：活動管理模組

在 Use Case 5.0 中所使用到的功能（包含：新增活動、檢視所有活動、修改活動或活動場次資訊、刪除活動），以下簡要說明各功能：

1. Use Case 5.1：新增活動

此使用案例用於說明管理者可透過此功能對活動進行新增，管理者需先登入，才可執行相關功能。此功能需要藉由管理者輸入所需欄位資料(名稱、介紹、日期、場次等)來新增活動，並且由系統檢查各項活動資訊是否符合要求。

2. Use Case 5.2：檢視所有活動

此使用案例用於說明管理者可透過此功能檢視所有活動與相關資訊，管理者需先登入，才可執行相關功能。

3. Use Case 5.3：修改活動或活動場次資訊

此使用案例用於說明管理者可透過此功能修改活動與活動場次相關資訊，管理者需先登入，才可執行相關功能。此功能可使管理者有修改活動與相關報名資訊的權限。

4. Use Case 5.4：刪除活動

此使用案例用於說明管理者可透過此功能修改活動內容與其相關資訊，管理者需先登入，才可執行相關功能。此功能用於管理者刪除活動與相關資訊。

表 12：活動管理模組關聯頁面

HTML	關聯 Use Case	說明
index.html	Use Case 2.1	瀏覽活動
eventinfoprototype.html	Use Case 2.3	查看活動詳細內容
collect.html	Use Case 3.1	會員可以收藏活動
center-myOrder-browse.html	Use Case 3.3	會員檢視收藏活動的內容
center-product-revise-signup.html	Use Case 4.3	會員可以修改報名資訊
manager-activityList.html	Use Case 5.2	管理者可以檢視所有活動

4.2.2.1 Sequence Diagram—Use Case 5.3 修改活動或活動場次紀錄報名紀錄

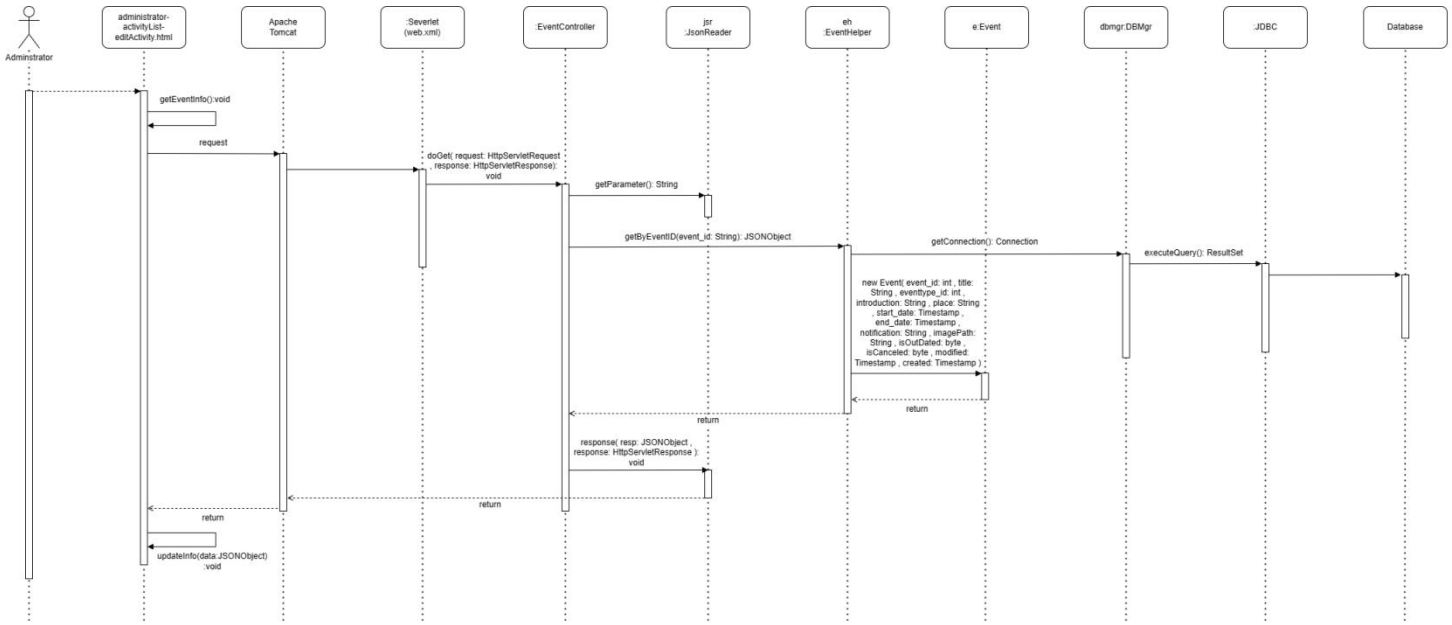

圖 8 ：商業流程編號 5.3 修改活動或活動場次紀錄報名紀錄循序圖（取回資料）

1. 管理員完成商業流程編號「6.1 管理者登入」後，透過商業流程編號「5.2 檢視所有活動」點擊欲修改之活動的「編輯」按鈕進入編輯頁面（administrator-activityList-editActivity.html）。
2. JavaScript 會抓取網址上之參數 id=<event_id>，透過 JavaScript 之 getEventInfo()送出 GET 請求。
3. 後端以 EventController 之 doGet()進行處理，以 JsonReader 取得該參數後，使用 EventHelper 物件之 getByEventID()方法嘗試取回會員資料。
4. 將所有取回之資料透過資料庫檢索將活動相關資料進行回傳。
5. 回傳之資料透過 JavaScript 回填至所屬欄位。
6. 若查詢成功則透過 JavaScript 之 updateInfo()更新 SQL 表格之內容。

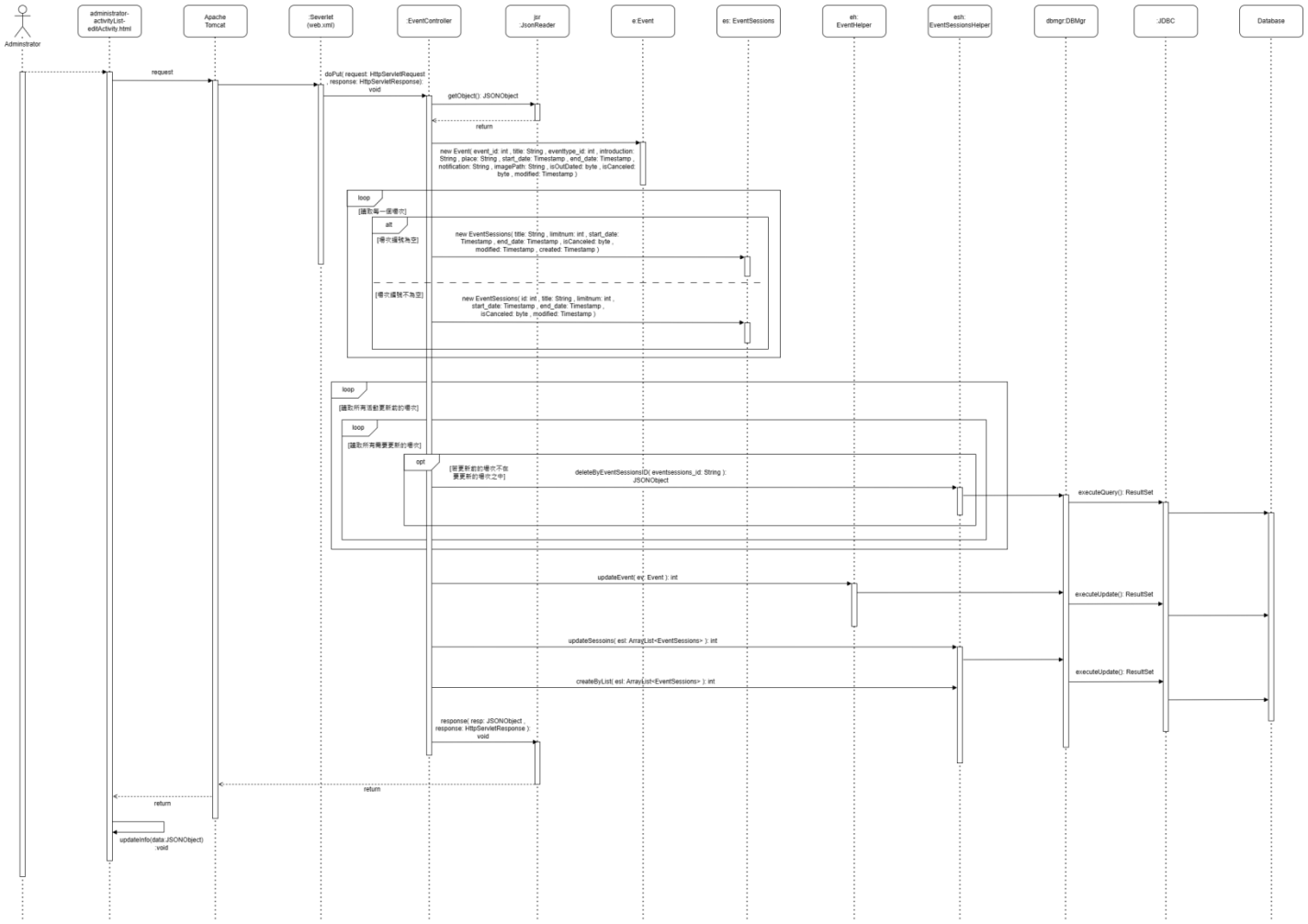

圖 9 ：商業流程編號 5.3 修改活動或活動場次紀錄報名紀錄循序圖（更新資料）

1. 管理者修改完成後，點擊「儲存更動」按鈕並通過前端驗證後，送出 PUT 請求。
2. 後端以 EventController 之 doPut()進行處理，以 JsonReader 取回 Request 之 JSON 格式參數。
3. 讀取每一個場次，若場次編號為空，為新增場次，產生 EventSessions 物件，若場次編號不為空，為修改場次，產生 EventSessions 物件。
4. 讀取更新活動前的所有場次，同時讀取所有需要更新的場次，若更新前的場次不在要更新的場次之中，則刪除該場次。
5. 接著透過 EventHelper 物件的 updateEvent(ev: Event)更新活動。

6. 透過 EventSessionsHelper 物件的 updateSessoins(esl: ArrayList<EventSessions>)及 createByList(esl: ArrayList<EventSessions>)更新、創建活動場次。
7. 最終回傳更新成功之結果。
8. 透過 JavaScript 之 updateInfo()更新 SQL 表格之內容。

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 @ 3.6GHz3.6GHz

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 19

2 Application Server：Apache Tomcat 10.1101

3 Database：Oracle MySQL Community 8.0.17.0

4 IDE：Eclipse 2022-12

4.1 專案類型：Java Servlet 4.0

4.2 程式語言：Java

5.1.3 前端套件

1. JQuery-3.4.1
2. CSS
3. Bootstrap

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 11）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。

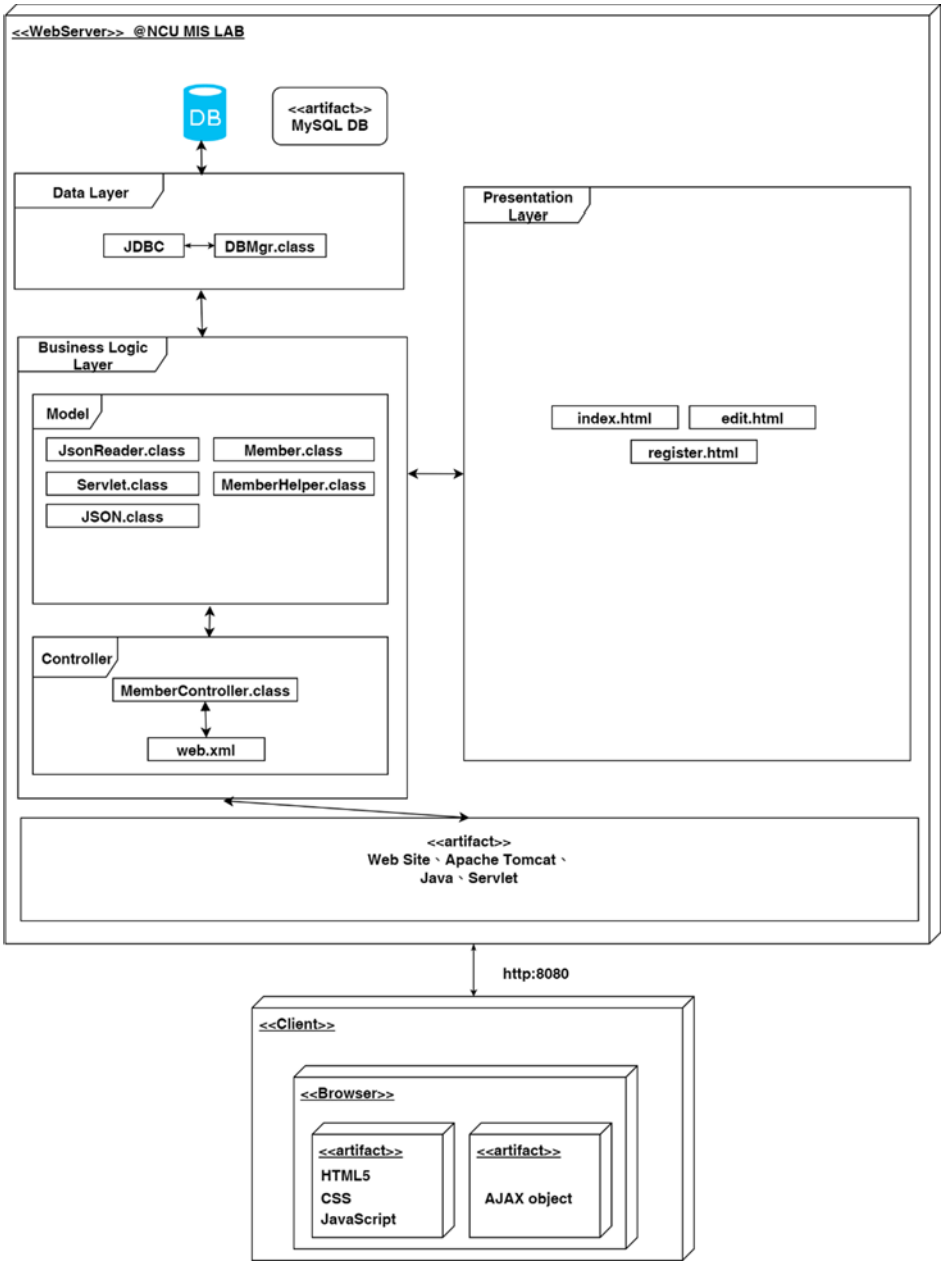

圖 10：系統架構圖

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 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 /NCU_MIS_SAGroup5/src/main/java/ 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「/NCU_MIS_SAGroup5/src/build/classes/」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「/NCU_MIS_SAGroup5/src/main/java/」資料夾當中，其中包含 DBMgr.java 等
- ✓ 同專案共用之工具則會放置到「/NCU_MIS_SAGroup5/src/main/java/」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 11）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 11：Servlet 之 Controller 範例模板（以 MemberController 為例）

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 12）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 12：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 14）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 13：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 14）所示：

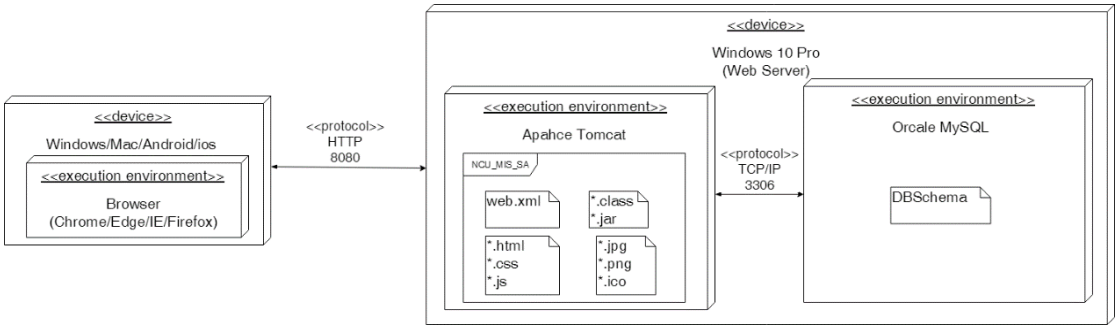

圖 14：專案部署圖

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（參考用）

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、pres、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 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。

   ✓ 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。

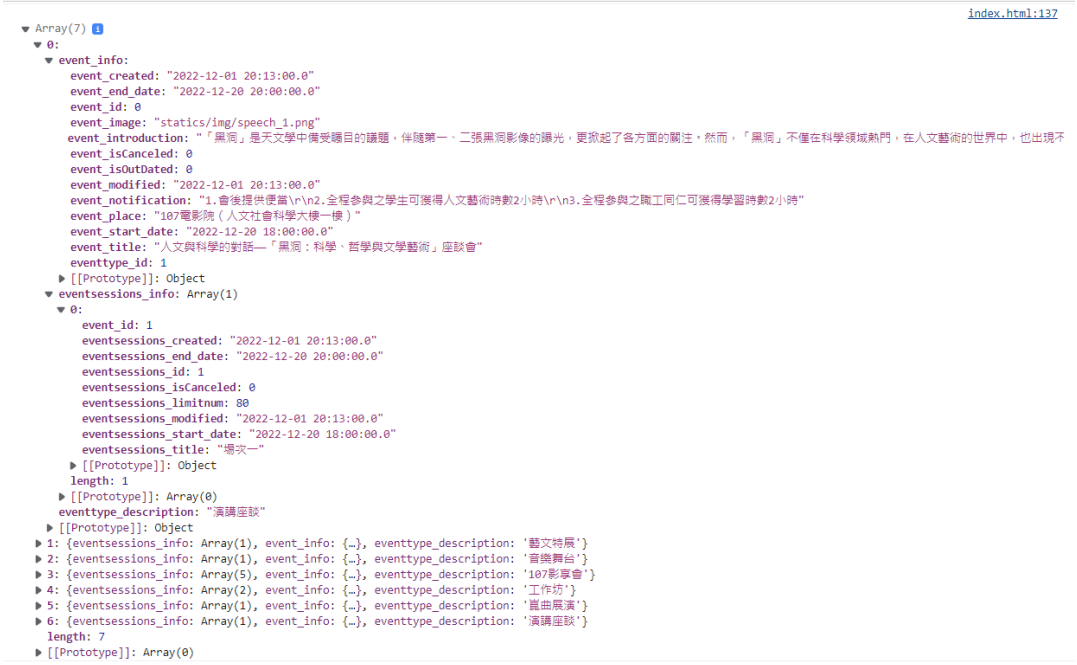

圖 15：EventController 之 GET 取得回傳之資料格式範例

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 16、圖 17、圖 18）為本系統之專案資料夾整體架構：
根目錄總共三個資料夾，分別如下:

1. .setting : 主要存放 Eclipse 相關設定檔案。
2. build : 其底下的 classes 資料用於存放.java 編譯後的.class 檔案
3. src>main : 存放專案運行的主要程式碼，在此之下再細分成兩個資料夾
    a. java
        - 存放所有 java 檔案，包含為 controller 、helper、tools (JSONReader、DBMGR 等)
    b. webapp
        - 根目錄放置包含 HTML 等 View 之相關文件
        - statics 資料夾當中則存放網站之靜態文件（CSS、image、doc、bootstrap 資源等）
        - WEB-INF 存放 Controller 與 Model 之後端檔案:web.xml 用於存放網站之路徑（route）資料；lib 則存放第三方套件

其他細節:

- 網站上線時，須將所有*.java 檔案編譯成*.class。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））。
- 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。

```
Folder PATH listing
Volume serial number is C0F2-FACC
C:\USERS\USER\DESKTOP\FINAL\NCU_MIS_SA_GROUP5\NCU_MIS_SA_GROUP5
|    .classpath
|    .project
|
|---.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
|
|---build
|   \---classes
|           Administrator.class
|           AdministratorController.class
|           AdministratorHelper.class
|           BCrypt.class
|           Collection.class
|           CollectionController.class
|           CollectionHelper.class
|           DBMgr.class
|           Event.class
|           EventController.class
|           EventHelper.class
|           EventSessions.class
|           EventSessionsHelper.class
|           JsonReader.class
|           Member.class
|           MemberController.class
|           MemberHelper.class
|           SessionMemberDetail.class
|           SessionMemberDetailController.class
|           SessionMemberDetailHelper.class
|
\---src
    \---main
        |---java
        |       Administrator.java
        |       AdministratorController.java
        |       AdministratorHelper.java
        |       BCrypt.java
        |       Collection.java
        |       CollectionController.java
        |       CollectionHelper.java
        |       DBMgr.java
        |       Event.java
        |       EventController.java
        |       EventHelper.java
        |       EventSessions.java
        |       EventSessionsHelper.java
        |       JsonReader.java
        |       Member.java
        |       MemberController.java
        |       MemberHelper.java
        |       SessionMemberDetail.java
        |       SessionMemberDetailController.java
        |       SessionMemberDetailHelper.java
        |
        \---webapp
            |   .tomcatplugin
```

圖 16：專案之資料夾架構(1/3)

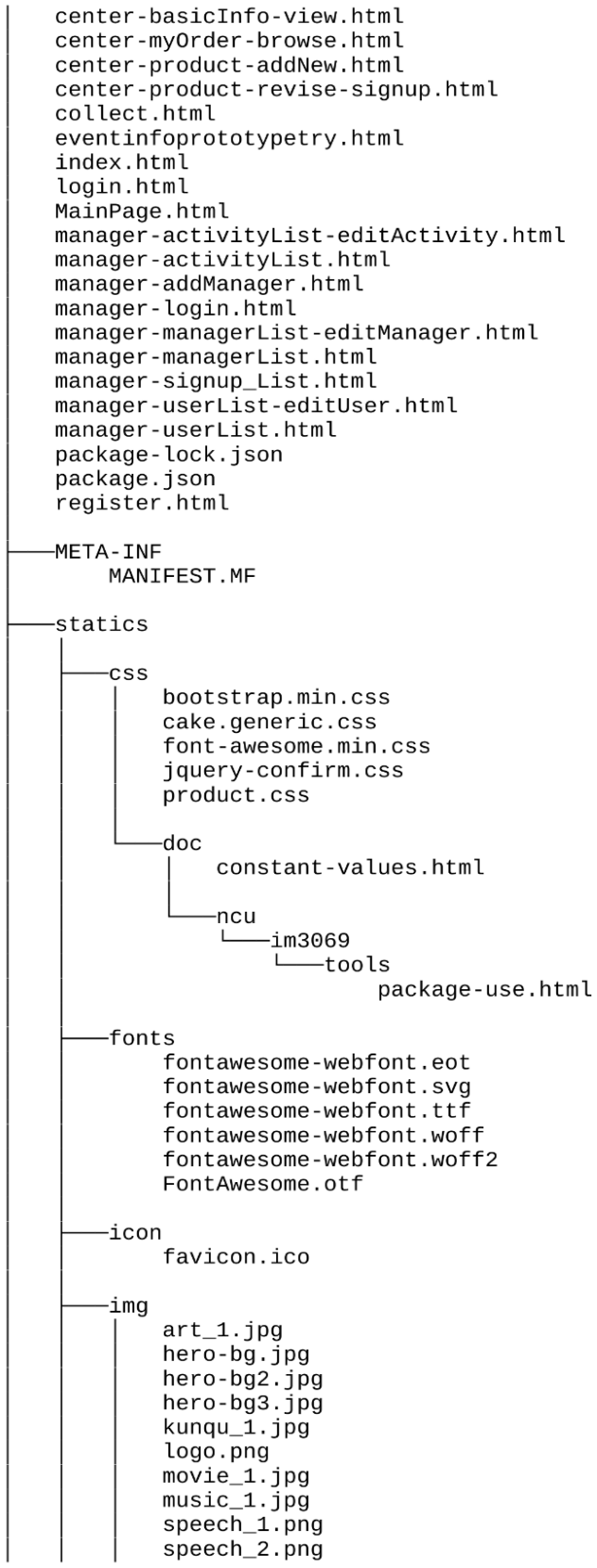

圖 17：專案之資料夾架構(2/3)

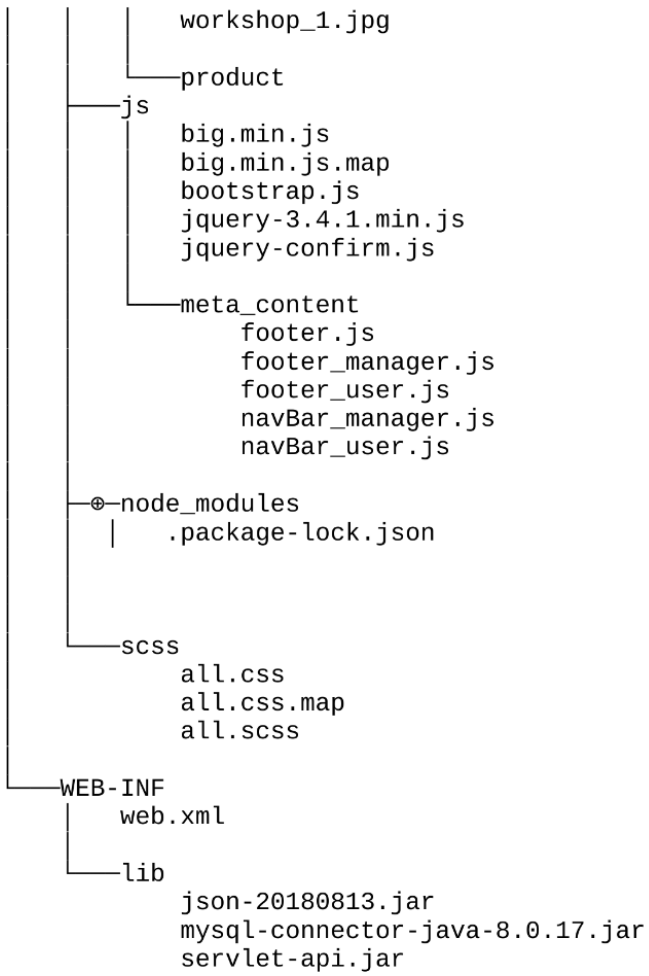

圖 18：專案之資料夾架構(3/3)

6.5 Route 列表

以下表格（表 13）為所有頁面之 Route 列表並依照 Index 順序逐項進行功能說明，本專案之範例實作會員及管理員模組，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 13：Route 表格

Index	Route	Action	網址參數	功能描述/作法
1	/login.html	POST	name,account,password	會員登入
2	/register.html	POST	name,account,password	會員註冊
3	/manager-login.html	POST	account,password	管理員登入
4	/manager-addManager.html	POST	account,name,password,email	新增管理者
5	/manager-signup-List.html	GET/DELETE	None,user_id	管理員取得報名人資料/管理員刪除已報名
6	/manager-userList-edit-User.html	PUT/GET	user_id,user_name,email,password	取得會員資料/管理者編輯會員資料頁面
7	/manager-managerList-edit-Manager.html	PUT/GET	administrator_id,account,user_name,password,isRoot	取得管理員資料/管理者修改管理者資料
8	/manager-userList.html	GET/DELETE	None/user_id	管理員取得會員資料/管理員刪除會員
9	/manager-managerList.html	GET/DELETE	None/administrator_id	取得所有管理員資料，並提供更改功能
10	/index.html	GET	eventtype_id	活動資料頁面
11	/eventinfoprototypetry	GET/POST	event_id/ eventsessions_id, member_id	活動詳細資料與報名頁面
12	/center-myOrder-browse.html	GET/ DELETE	member_id/ eventsessions_id	已報名活動資料編輯頁面

13	/collect.html	GET/ DELETE	member_id/ collection_id	會員收藏活動編輯頁面
14	/manager-activityList.html	GET	eventtype_id/	管理員國棟資料頁面
15	/manager-activityList-editActivity.html	GET/PUT /DELETE	event_id/event_id, eventsessions_id/ event_id	活動編輯頁面
16	/center-product-addNew.html	POST	None	活動新增頁面
17	/api/member.do	GET	None	取得會員資料
18	/api/member.do	POST	None	新增會員、判斷登入密碼是否相同
19	/api/member.do	DELETE	None	刪除會員
20	/api/member.do	PUT	None	更新會員
21	/api/administrator.do	GET	None	取得管理員資料
22	/api/administrator.do	POST	None	新增管理員、判斷管理員登入密碼是否相同
23	/api/administrator.do	DELETE	None	刪除管理員
24	/api/administrator.do	PUT	None	更新管理員
25	/api/event.do	GET	None	取得活動資料
26	/api/event.do	POST	None	新增活動
27	/api/event.do	PUT	None	更新活動資料
28	/api/event.do	DELETE	None	刪除活動
29	/api/collection.do	GET	None	取得收藏
30	/api/collection.do	POST	None	加入收藏
31	/api/collection.do	DELETE	None	刪除收藏
32	/api/sessoinmemberdetail.do	GET	None	取得已報名的場次及活動資料

33	/api/sessoinmember detail.do	POST	None	報名活動場次
34	/api/sessoinmember detail.do	DELETE	None	取消報名活動場次

下列將根據上表進行更詳細之說明與其運行步驟：

1. /login.html：取得登入頁面（render page），運行步驟如下：
   A. 取得網址所傳入的姓名、帳號、密碼參數(name, account, password)
   B. 進入頁面，透過 AJAX 發送帶參數之 POST 請求。
      若 name 值為空則為登入
   C. 將取回之資料庫會員資料 Render 至表格當中。
   D. 將取回之所下 SQL 指令與花費時間更新至表格當中。
   E. 依傳入的參數(密碼)與資料庫資料(hashed 後的密碼)進行比對
   F. 回傳結果(是否能夠登入)

2. /register.html：取得會員資料更新頁面（render page），運行步驟如下：
   A. 取得網址所傳入的姓名、帳號、密碼參數(name, account, password)
   B. 進入頁面，透過 AJAX 發送帶參數之 POST 請求。若 name 不為空則為註冊
   C. 利用 checkDuplicate()確認是否有重複的會員資料，若無則將取回之資料庫會員資料 Render 至表格當中。
   D. 將取回之所下 SQL 指令與花費時間更新至表格當中。

3. /manager_login.html：
   A. 取得網址所傳入的帳號、密碼參數(name, account, password)
   B. 進入頁面，透過 AJAX 發送帶參數之 POST 請求。
      若 name 值為空則為登入
   C. 將取回之資料庫會員資料 Render 至表格當中。
   D. 將取回之所下 SQL 指令與花費時間更新至表格當中。
   E. 依傳入的參數與資料庫資料進行比對
   F. 回傳結果(是否能夠登入)

4. /manager-addManager.html:
   A. 取得網址所傳入的帳號、姓名、密碼、電子郵件參數 account,name,password,email
   B. 進入頁面，透過 AJAX 發送帶參數之 POST 請求。
   C. 將取回之所下 SQL 指令與花費時間更新至表格當中

5. /manager-signup-List.html:
   A. 透過 AJAX 發布 GET 請求
   B. 取得所有報名活動的會員資料
   C. 進入頁面，透過 AJAX 發送帶參數(user_id)之 DELETE 請求。
   D. 刪除在資料庫中對應到 user_id 的 member_id 會員

6. /manager-userList-edit-User.html
   A. 透過 AJAX 發布 GET 請求
   B. 將取得資料 Render 在表格中
   C. 若按下修改會員資料按鈕則取得 user_id,user_name,email,password
   D. 透過 AJAX 發布 PUT 請求
   E. 將取得的會員資料更新至資料中

7. /manager-managerList-edit-Manager.html
   A. 透過 AJAX 發布 GET 請求
   B. 將取得資料 Render 在表格中
   C. 若按下修改管理員資料按鈕
   D. 取得網址傳入參數(administrator_id,account,user_name,password,isRoot)
   E. 透過 AJAX 發布 PUT 請求
   F. 將取得資料更新在資料庫中

8. /manager-userList.html
   A. 透過 AJAX 發布 GET 請求
   B. 將取回的資料庫會員 Render 至網頁欄位中

C. 若按下確認刪除會員

D. 取得網頁傳入參數(user_id)

E. 透過 AJAX 發布 DELETE 請求

F. 刪除該名會員在資料庫的會員資料

9. /manager-managerList.html

A. 取回網頁傳入參數(administrator_id)

B. 透過 AJAX 發布 GET 請求

C. 將取回管理員資料至表格中

D. 透過 AJAX 發布 DELETE 請求

E. 透過傳入參數(administrator_id)刪除在資料庫中對應的管理員資料

10. /index.html: 取得活動資料列表頁面

A. 透過 AJAX 發送 GET 請求，若參數不為空則取回特定類別的資料，若為空則取回所有活動資料

B. 將取回之資料庫活動資料 Render 至表格中

11. /eventinfo.html: 取得某一活動的詳細資料頁面

A. 取得網頁所傳入的活動編號(event_id)參數

B. 透過 AJAX 發送帶參數之 GET 請求

C. 將取回之資料庫活動資料 Render 至表格中

D. 若報名按鈕被按下，則取得網頁中的活動場次編號(eventsessions_id)以及會員編號(member_id)

E. 透過 AJAX 發送帶參數之 POST 請求

F. 使用 member_id 與 eventsessions_id 於 sessionmemberdetail 資料表中新增一筆資料，紀錄會員報名了哪一場活動的場次

12. /center-myOrder-browse.html: 取得某一會員已報名的活動場次頁面

A. 取得網頁所傳入的會員編號(member_id)參數

B. 透過 AJAX 發送帶參數之 GET 請求

C. 將取回之資料庫活動資料 Render 至表格中

D. 將取回之資料庫活動資料 Render 至表格中

E. 透過 AJAX 發送帶參數之 DELETE 請求

F. 使用場次編號刪除 sessionmemberdetail 已報名場次資料表中的資料

13. /collect.html: 取得某一會員已收藏的活動頁面

A. 取得網頁所傳入的會員編號(member_id)參數

B. 透過 AJAX 發送帶參數之 GET 請求

C. 將取回之資料庫活動資料 Render 至表格中

D. 若列表中某一場次之專屬刪除按鈕被按下，則取得該活動之收藏編號(collection_id)

E. 透過 AJAX 發送帶參數之 DELETE 請求

F. 使用收藏編號刪除 collection 收藏資料表中的資料

14. /manager-activityList.html: 為管理員取得活動資料列表頁面

A. 透過 AJAX 發送 GET 請求，若參數不為空則取回特定類別的資料，若為空則取回所有活動資料

B. 將取回之資料庫活動資料 Render 至表格中

15. /manager-activityList-editActivity.html: 活動與場次資料編輯頁面

A. 取得網頁所傳入的活動類別編號(event_id)參數

B. 透過 AJAX 發送 GET 請求

C. 將取回資料填入先前的欄位中

D. 若儲存更動按鈕被按下，則取得活動之活動編號與場次編號

E. 透過 AJAX 發送帶參數的 POST 請求

F. 將更新過後的活動資料與場次資料填入資料庫中

G. 若刪除活動按鈕被按下，則取得活動之活動編號

H. 透過 AJAX 發送帶參數的 DELETE 請求

I. 使用活動編號刪除資料庫中相關的活動、活動場次、收藏以及已報名場次資料

16. /center-product-addNew.html: 活動與場次資料新增頁面

A. 進入頁面

B. 若新增活動按鈕被按下，則收集個欄位之資料

C. 透過 AJAX 發送帶參數的 POST 請求

D. 將新增的活動與場次資料加入資料庫中料

17. /api/member.do：取得會員資料之 API，運行步驟如下：

A. 前端傳入會員編號參數（id）

B. 請求資料庫中該名會員資料。

C. 回傳取得結果

18. /api/member.do：登入、註冊會員之 API，運行步驟如下：

A. 前端傳入 account、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

19. /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(將 isDeleted 改為 1)

C. 回傳刪除成功訊息

20. /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 account、password 之 JSON 物件。

B. 更新 member 資料。

C. 回傳更新成功訊息。

21. /api/administrator.do: 取得管理員資料之 API ，運行步驟如下

A. 前端可選擇是否傳入管理員編號(administrator_id)

B. 若參數不為空則取回特定 id 的資料，若為空則取回所有管理員資料

C. 回傳取得的結果

22. /api/Administrator.do：註冊、登入管理員之 API，運行步驟如下：

A. 前端取得帳號(account)、密碼(password)

B. 註冊則建立 Administrator 物件

C. 登陸則判斷密碼是否和資料庫中 hashed 過的密碼相符

D. 回傳狀態消息

23. /api/Administrator.do：刪除會員之 API，運行步驟如下：

A. 前端傳入管理員編號之 JSON 物件。

B. 刪除管理員資料

C. 回傳刪除成功訊息

24. /api/Administrator.do：更新會員資料之 API，運行步驟如下：

A. 前端傳入更新後之 administrator_id,account、user_name、isRoot,password 之 JSON 物件。

B. 更新 Administrator 資料。

C. 回傳更新成功訊息。

25. /api/event.do: 取得活動資料之 API

A. 前端可選擇是否傳入活動類別編號(eventtype_id)之 JSON 物件

B. 若參數不為空則取回特定類別的資料，若為空則取回所有活動資料

C. 回傳取得的結果

26. /api/event.do: 新增活動資料之 API

A. 前端傳入 event_title, eventtype_id, event_introduction, evnet_place, event_start_date, event_end_date, event_notification, event_image, event_isCanceled, event_modified 以及 eventsessions_title, eventsessions_litmitnum, eventsessions_start_date, eventsessions_end_date, eventsessions_isCanceled, eventsessions_modified, eventsessions_created 兩者結合之 JSON 物件

B. 建立 Event 與 EventSessions

C. 回傳新增成功的資訊

27. /api/event.do: 更新活動資料之 API

A. 前端傳入 event_id,event_title, eventtype_id, event_introduction, evnet_place, event_start_date, event_end_date, event_notification, event_image 以及 eventsessions_id, eventsessions_title, eventsessions_limitnum, eventsessions_start_date, eventsessions_end_date, eventsessions_isCanceled, eventsessions_modified 兩者結合之 JSON 物件

B. 更新 Event 與 EventSessions

C. 回傳更新成功的資訊

28. /api/event.do: 刪除活動資料之 API

A. 前端傳入活動編號之 JSON 物件

B. 刪除與編號相關的活動、活動場次、已報名場次、收藏資料

C. 回傳刪除成功的資訊

29. /api/collection.do: 取得收藏資料之 API

A. 前端傳入會員編號之 JSON 物件

B. 取得該會員的收藏資料

C. 回傳取得的結果

30. /api/collection.do: 新增收藏資料之 API

A. 前端傳入會員編號與活動編號之 JSON 物件

B. 於資料庫中新增收藏資料

C. 回傳新增成功的訊息

31. /api/collection.do: 刪除收藏資料之 API

A. 前端傳入收藏編號之 JSON 物件

B. 於資料庫中刪除收藏資料

C. 回傳刪除成功的訊息

32. /api/sessoinmemberdetail.do: 取得已報名活動場次之 API

A. 前端傳入會員編號之 JSON 物件

B. 於資料庫中取得已報名的場次及活動資料

C. 回傳取得的結果

33. /api/sessoinmemberdetail.do: 報名活動場次之 API

A. 前端傳入會員編號與場次編號之 JSON 物件

B. 於資料庫中新增已報名活動場次資料

C. 回傳報名成功的訊息

34. /api/sessoinmemberdetail.do: 取消報名活動場次之 API

A. 前端傳入已報名活動場次編號之 JSON 物件

B. 於資料庫中刪除已報名活動場次資料

C. 回傳取消報名成功的訊息

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：GitHub - xup6sophia/MIS_SA_Group5。

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 19）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。

```
//* 操作 Controller 與 import Servlet 所需
import java.io*;
import java.util*;
import jakarta.servlet*;
import jakarta.servlet.http*;
import JsonReader;

//* 操作 JSONObject 相關物件
import org.json.*;

//* 操作 JDBC 相關物件
import java.sql*;
```

圖 19：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 20：常見之 JSON 格式範例

上圖（圖 20）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 21）為例：

```
function submit() {

  // 將前端表單填寫值存進變數
  var student_id = $('#input_student_id').val();
  var name = $('#input_name').val();
  var email = $('#input_email').val();
  var password = $('#input_password').val();
  var phone_number = $('#input_phone_number').val();

  // 格式設定(未做)
  // 格式設定(未做)
    var email_rule = /^\w+((-\w+)|(\.\w+))*\@[A-Za-z0-9]+((\.|-)[A-Za-z0-9]+)*\.[A-Za-z]+$/;
    var stu_id_rule = /^(?=.*[0-9])[0-9\d]{9,9}$/;
    var user_phone_rule = /^(?=.*[0-9])[0-9\d]{10,10}$/;

    if (!email_rule.test(email)) {
        alert("Email格式不符!");
    }
    else if (!stu_id_rule.test(student_id)) {
        alert("學號由9位數字組成!");
    }
    else if (!user_phone_rule.test(phone_number)) {
        alert("請輸入10位電話");
    }
    else {
      // 將資料組成JSON格式(格式名稱參照文件)
      var data_object = {
          "user_name": name,
          "stu_id": student_id,
          "email": email,
          "password": password,
          "user_phone": phone_number
      };

                // 將JSON格式轉換成字串
                var data_string = JSON.stringify(data_object);

                // 發出POST的AJAX請求(未做)
                $.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) {
                           alert("建立成功!")
                    document.location.href = "login.html";
                         }
                    },
                    error: function () {
                        alert("無法連線到伺服器！");
                    }
                });

            }


        }
    });
</script>
```

圖 21：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 22）顯示修改管理員之方法，根據修改後的值進行回填方式：

```
function submitEdit() {
        // 將前端表單填寫值存進變數
        var account = $('#input_account').val();
        var user_name = $('#input_user_name').val();
        var password = $('#input_password').val();
        var isRoot = $('#input_isRoot').val();


        var data_object = {
                "administrator_id" : administrator_id,
                "account" : account,
                "user_name" : user_name,
                "password" : password,
                "isRoot": isRoot
        };


        // 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 : 'PUT',
                url : "api/administrator.do",
                data : data_string,
                crossDomain : true,
                cache : false,
                dataType : 'json',
                timeout : 5000,
                success : function(response) {
                        if (response.status == 200) {
                                alert("成功修改資料");
                        } else if (response.status == 400) {
                                alert("修改失敗");
                        }
                },
                error : function() {
                        alert("無法連線到伺服器");
                }
        });
}
```

```
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 22：修改管理員之欄位回填

若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 23）以更新管理員之表格為例：

```
// 更新管理員列表表格
function updateTable(data) {
        // 清空前端表格rows

        $("#member_list_table > tbody").empty();

        var table_html = '';

        $.each(data, function(index, value) {
                table_html += '<tr>';
                table_html += '<th scope="row">' + value['administrator_id'] + '</th>';
                table_html += '<td>' + value['email'] + '</td>';
                table_html += '<td>' + value['name'] + '</td>';
                table_html += '<td>';
                table_html += '<a href="manager-managerList-editManager.html?administrator_id=' + value['administrator_id'] +'" class="btn btn-outline-
primary">修改</a>';
                table_html += '<a href="javascript: deleteMember('+ value['administrator_id']+ ')" class="btn btn-outline-danger">刪除</a>';
                table_html += '</td></tr>';
            })

        // 重新append所有會員row
        $("#member_list_table > tbody").append(table_html);
    }
```

圖 23：更新管理員之表格

7.2 JsonReader、JSONObject 與 JSONArray 操作

為簡化取回前端所傳入之 JSON 資料，本專案使用了 JsonReader class 來協助處理。為此專案中需包含 JsonReader.java 的檔案，並讓整個專案中的其他 java 檔都能讀取。另外，若要操作 JSONObject，則必須另外 import（import org.json.*;），程式碼範例如下圖（圖 24）所示：

```
import JsonReader;
import org.json.*;


//* 使用 JsonReader 解析 Servlet 中 HttpServletRequest 之 request
JsonReader jsr = new JsonReader( request );
JSONObject jso = jsr.getObject();


//* 取出 request 中的參數
String name = jso.getString( "name" );
String email = jso. getString( "email" );
int age = jso.getInt( "age" );


//* 新建一個 JSONObject 將回傳之資料進行封裝
JSONObject resp = new JSONObject();
resp.put( "status" , 200 );
resp.put( "message" , "回傳成功!" );
resp.put( "response" , data );     //* data 可為 數值、JSONObjecct 或 JSONArray
jsr.response( resp , response );
```

圖 24：JsonReader 操作之範例

下表（表 14）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 14：JSONObject 和 JSONArray 之操作範例

<table>
<tr><td>

```
//* 要取回的 JSONObject
{
    "name": "小明",
    "email": "demo123@gmail.com"
    "age": 21,
    "major": [ "MIS" , "CS" ],
    "data":
    {
        "id": 1594826,
        "class": "3B"
    }
}
```

</td><td>

```
JSONObject jso = new JSONObject( JSONObject_name );
String name = jso.getString( "name" );
String email = jso.getString( "email" );
int age = jso.getIng( "age" );
JSONArray major = jso.getJSONArray( "major" );
JSONObject data = jso.getJSONObject( "data" );
```

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

```
//* 新增物件
JSONObject jso = new JSONObject();

//* 放入 key-value
jso.put( "name" , "小明" );

//* 放入 Array
jso.element( "major" , new JSONArray() );
```

</td><td>

```
//* 新增 Array
JSONArray jsa = new JSONArray();

//* 放入 key-value
jsa.add( 0 , "MIS" );
```

</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

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 26）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 25 第 456 行透過 DBMgr.getConnection()建立連線。

B. 圖 25 第 460 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 24）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 25 第 463 行若為檢索則是 executeUpdate()，回傳為 int 整數，即影響之行數；其餘指令如圖 25 第 107 行 executeQuery()回傳為 ResultSet。

D. 圖 26 第 107 行檢索後透過 ResultSet 將結果儲存，圖 26 第 113 行並以 while 迴圈移動指標，將每筆查詢結果依序進行操作。

```
446         int row = 0;
447
448         String execute_sql = "";
449
450         long start_time = System.nanoTime();
451
452         JSONObject deleted_eventsessions = ESH.deleteByEventID( eventID );
453
454         try
455         {
456             conn = DBMgr.getConnection();
457
458             String sql = "DELETE FROM group5_final.event WHERE event_id = ?";
459
460             pres = conn.prepareStatement(sql);
461             pres.setString( 1 , eventID );
462
463             row = pres.executeUpdate();
464
465             execute_sql = pres.toString();
466             System.out.println(execute_sql);
467
468
469         }
470         catch( SQLException sqlex )
471         {
472             /** 印出JDBC SQL指令錯誤 **/
473             System.err.format("SQL State: %s\n%s\n%s", sqlex.getErrorCode(), sqlex.getSQLState(), sqlex.getMessage());
474         }
475         catch( Exception ex )
476         {
477             /** 若錯誤則印出錯誤訊息 */
478             ex.printStackTrace();
479         }
480         finally
481         {
482             /** 關閉連線並釋放所有資料庫相關之資源 **/
483             DBMgr.close(pres, conn);
484         }
```

圖 25：依照活動編號刪除活動之 EventHelper deleteByEventID() method（節錄）

```
95      try
96      {
97          /** 取得資料庫之連線 */
98          conn = DBMgr.getConnection();
99          /** SQL指令 */
100         String sql = "SELECT * FROM `group5_final`.`eventsessions` WHERE `eventsessions`.`event_id` = ?";
101
102         /** 將參數回填至SQL指令當中 */
103         pres = conn.prepareStatement(sql);
104         pres.setInt(1, event_id);
105
106         /** 執行新增之SQL指令並記錄影響之行數 */
107         rs = pres.executeQuery();
108
109         /** 紀錄真實執行的SQL指令，並印出 **/
110         String exexcute_sql = pres.toString();
111         System.out.println(exexcute_sql);
112
113         while(rs.next())
114         {
115             /** 將 ResultSet 之資料取出 */
116             int eventsessions_id = rs.getInt("eventsessions_id");
117             String title = rs.getString("eventsessions_title");
118             int limitnum = rs.getInt("eventsessions_limitnum");
119             Timestamp start_date = rs.getTimestamp("eventsessions_start_date");
120             Timestamp end_date = rs.getTimestamp("eventsessions_end_date");
121             byte isCanceled = rs.getByte("eventsessions_isCanceled");
122             Timestamp modified = rs.getTimestamp("eventsessions_modified");
123             Timestamp created = rs.getTimestamp("eventsessions_created");
124
125             /** 將每一筆會員資料產生一名新Member物件 */
126             es = new EventSessions( eventsessions_id , event_id , title , limitnum , start_date , end_date , isCanceled , modified , created );
127             /** 取出該名會員之資料並封裝至 JSONsonArray 內 */
128             result.add(es);
129         }
130     }
```

圖 26：以活動編號取得所有活動場次之 EventSessionsHelper getEventSessionByEventID() method（節錄）

SQL 資料庫之操作，大致上以下圖（圖 27）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 = "";
/** 記錄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 27：操作 JDBC 之模板