國　　立　　中　　央　　大　　學

資　訊　管　理　學　系

108（一）系　統　分　析　與　設　計

系統軟體設計規格書

第 14 組

資管三 A 106403518　呂文楷

資管三 A 106403015　華崧淇

資管三 A 106403201　陳威捷

企管四 B 105401530　余函倩

資管三 A 106403505　蔡苑萍

資管三 A 106403007　蔣書珊

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

<設計文件修改部分>

第二章：

演唱會資料表修正-----增加 session 欄位

第三章：

類別圖(4 張)修正

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	十四組	Draft	2019/12/29
V1.0.0	十四組	完成版	2019/1/4

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 6 個實體（Entity）、5 個關係（Relationship）、0 個複合性實體（Compound Entity），下圖（圖 1）為設計階段之 ER 圖，亦可使用資料庫綱要圖（Schema Diagram）進行取代：

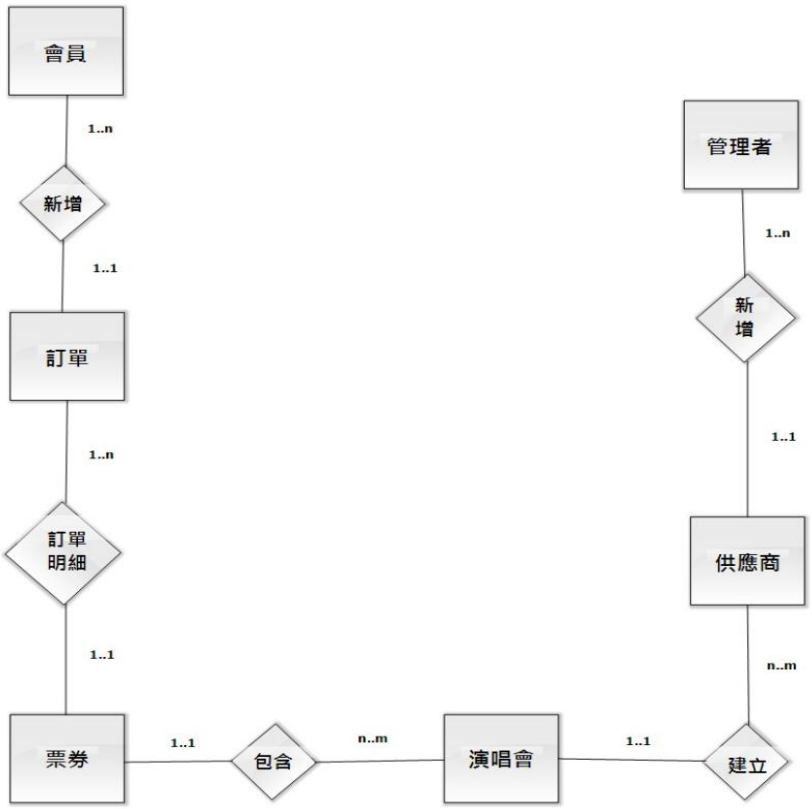


圖 1：設計階段之實體關係圖

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（members）：

表 1：會員資料表（members）之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K	idmember	Int	無	否	V	
	name	Varchar(255)	無	否		utf8mb4_0900_ai_ci
	email	Varchar(255)	無	否		utf8mb4_0900_ai_ci
	password	Varchar(255)	無	否		utf8mb4_0900_ai_ci
	dateofbirth	DATE	無	否		
	idnumber	Varchar(255)	無	否		
	phonenumber	Varchar(255)	無	否		
	address	Varchar(255)	無	否		

- ✓ idmember：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。
- ✓ name：用於紀錄會員姓名。
- ✓ email：用於紀錄該名會員電子郵件地址。
- ✓ password：用於記錄該名會員註冊時設定的密碼。
- ✓ dateofbirth：用於記錄該名會員出生年月日。
- ✓ idnumber：用於記錄該名會員身分證字號。
- ✓ phonenumber：用於記錄該名會員電話號碼。
- ✓ address：用於記錄該名會員住處地址。

2. 商品資料表（ticket）：

表 2：商品資料表（ticket）之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K	idticket	Int(11)	無	否	V	
F.K	concertid	Int(11)	無	否		utf8mb4_0900_ai_ci
F.K	orderid	Int(11)	無	否		

	seatarea	Varchar(255)	無	否		utf8mb4_0900_ai_ci
	seatid	Int(11)	無	否		
	isused	TINYINT(1)	0	否		
	name	Varchar(255)	無	否		
	email	Varchar(255)	無	否		
	phonenumber	Varchar(255)	無	否		

- ✓ idticket：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。
- ✓ concertid：用於紀錄該商品的演唱會編號。
- ✓ orderid：用於紀錄該商品後續所放入的訂單編號。
- ✓ seatarea：用來記錄商品的座位區域。
- ✓ seatid：用來記錄商品的座位編號。
- ✓ isused：用來記錄商品是否被使用。
- ✓ name：用來記錄商品購買者的姓名。
- ✓ email：用來記錄商品購買者的電子郵件。
- ✓ phonenumber：用來記錄商品購買者的電話號碼。

3. 訂單資料表（order）：

表 3：訂單資料表（order）之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K	idorder	Int(11)	無	否	V	
F.K	memberid	Int(11)	無	否		utf8mb4_0900_ai_ci
	payment	Varchar(255)	無	否		utf8mb4_0900_ai_ci
	paid	TINYINT(1)	0	否		utf8mb4_0900_ai_ci
	ticketamount	Int(11)	無	否		
	createtime	Datetime	0	否		
	totalprice	Int(45)	無	否		

- ✓ idorder：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：用於紀錄該訂單所屬的會員編號。
- ✓ payment：用於紀錄該訂單的付款方式。
- ✓ paid：用於紀錄該訂單的付款金額。

- ✓ ticketamount：用於紀錄該訂單購買的票券數量。
- ✓ createtime：用於紀錄該訂單成立的時間。
- ✓ totalprice：用來記錄該訂單的總金額。

4. 演唱會資料表（concert）：

表 4：演唱會資料表(concert)之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K	idconcert	Int(11)	無	否	V	
	name	Varchar(45)	無	否		utf8mb4_0900_ai_ci
F.K	supplierid	Int(11)	無	否		utf8mb4_0900_ai_ci
	location	Varchar(45)	無	否		utf8mb4_0900_ai_ci
	picture	Varchar(5000)	無	否		
	seatpicture	Varchar(5000)	無	否		
	endsellingtime	Datetime	無	否		
	content	Varchar(10000)	無	否		
	ticketstatus	JSON	無	否		
	concertstarttime	Datetime	無	否		
	concertendtime	Datetime	無	否		
	session	Varchar	無	否		

- ✓ idconcert：為自動增加作為演唱會編號，不可更動由資料庫系統自動產生。
- ✓ name：用於紀錄該演唱會的名稱。
- ✓ supplierid：用於紀錄該演唱會供應商的識別編號。
- ✓ location：用於紀錄該演唱會的地點。
- ✓ picture：用於紀錄演唱會封面圖檔之路徑位置。
- ✓ seatpicture：用於紀錄演唱會座位圖檔的路徑位置。
- ✓ endsellingtime：用於紀錄演唱會售票截止時間。
- ✓ content：用於紀錄進入購票頁面前的演唱會描述內容。
- ✓ ticketstatus：用於紀錄演唱會總共票券區域、價格、數量。
- ✓ concertstarttime：紀錄演唱會開始時間。
- ✓ concertendtime：紀錄演唱會結束時間。

- ✓ session：用來記錄演唱者同一期演唱會。

5. 管理者資料表 (manager)

表 5：管理者資料表(manager)資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K	idmanager	Int(11)	無	否	V	
	account	Varchar(255)	無	否		utf8mb4_0900_ai_ci
	password	Varchar(255)	無	否		utf8mb4_0900_ai_ci
	lastlogintime	Datetime	無	否		utf8mb4_0900_ai_ci

- ✓ idmanager：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。
- ✓ account：用來記錄管理者帳號。
- ✓ password：用來記錄管理者密碼。
- ✓ lastlogintime：用來記錄管理者最後登入的時間。

6. 供應商資料表 (supplier)

表 6：供應商資料表(supplier)資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K	idsupplier	Int(11)	無	否	V	
	account	Varchar(255)	無	否		utf8mb4_0900_ai_ci
	password	Varchar(255)	無	否		utf8mb4_0900_ai_ci
	name	Varchar(255)	無	否		utf8mb4_0900_ai_ci
	phonenumber	Varchar(255)	無	否		
F.K	addedbywhom	Int(11)	無	否		

- ✓ idsupplier：為自動增加作為供應商編號，不可更動由資料庫系統自動產生。
- ✓ account：用來記錄供應商帳號。
- ✓ password：用來記錄供應商密碼。
- ✓ name：用來記錄供應商名稱。

- ✓ phonenumber：用來記錄供應商電話號碼。
- ✓ addedbywhom：用來記錄加入該供應商的管理者。

第 3 章　類別圖

下圖（圖 2、　　圖 3、圖 4、圖 5）係依據搶票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）。

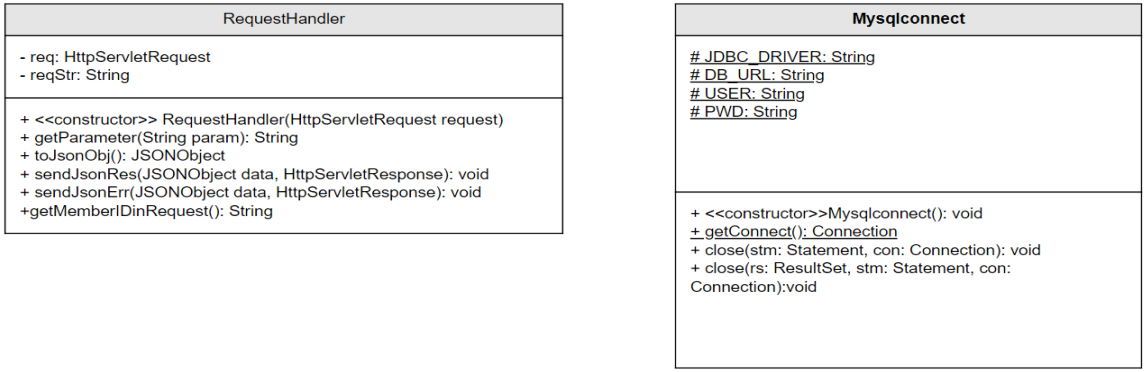

RequestHandler
- req: HttpServletRequest
- reqStr: String
+ <<constructor>> RequestHandler(HttpServletRequest request)
+ getParameter(String param): String
+ toJsonObj(): JSONObject
+ sendJsonRes(JSONObject data, HttpServletResponse): void
+ sendJsonErr(JSONObject data, HttpServletResponse): void
+getMemberIDinRequest(): String
Mysqlconnect
JDBC_DRIVER: String
DB_URL: String
USER: String
PWD: String
+ <<constructor>>Mysqlconnect(): void
+ getConnect(): Connection
+ close(stm: Statement, con: Connection): void
+ close(rs: ResultSet, stm: Statement, con:
Connection):void

圖 2：類別圖（1/4）

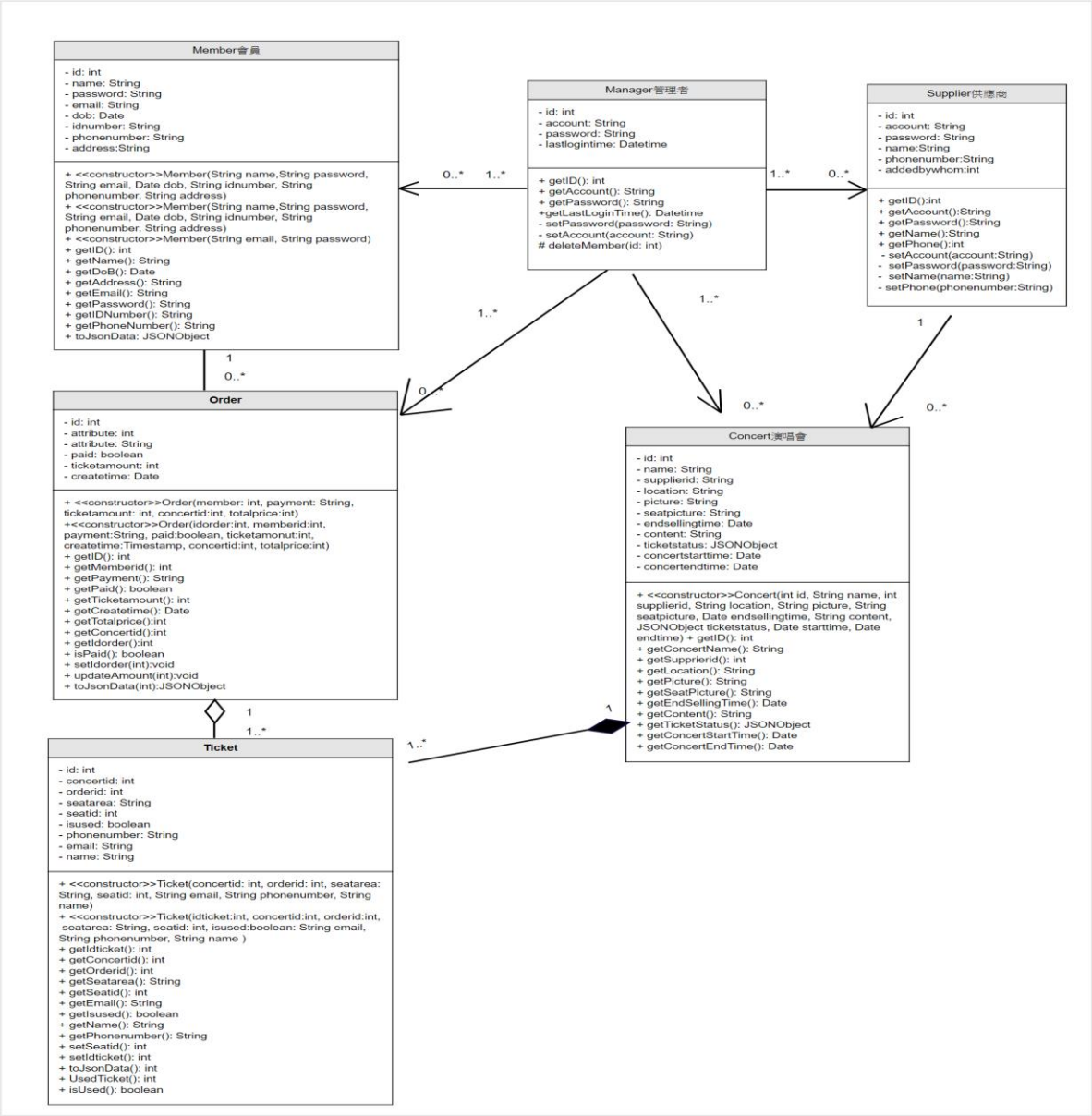

Member會員
- id: int
- name: String
- password: String
- email: String
- dob: Date
- idnumber: String
- phonenumber: String
- address:String
+ <<constructor>>Member(String name,String password, String email, Date dob, String idnumber, String phonenumber, String address)
+ <<constructor>>Member(String name,String password, String email, Date dob, String idnumber, String phonenumber, String address)
+ <<constructor>>Member(String email, String password)
+ getID(): int
+ getName(): String
+ getDoB(): Date
+ getAddress(): String
+ getEmail(): String
+ getPassword(): String
+ getIDNumber(): String
+ getPhoneNumber(): String
+ toJsonData: JSONObject
Manager管理者
- id: int
- account: String
- password: String
- lastlogintime: Datetime
+ getID(): int
+ getAccount(): String
+ getPassword(): String
+getLastLoginTime(): Datetime
- setPassword(password: String)
- setAccount(account: String)
deleteMember(id: int)
Supplier供應商
- id: int
- account: String
- password: String
- name:String
- phonenumber:String
- addedbywhom:int
+ getID():int
+ getAccount():String
+ getPassword():String
+ getName():String
+ getPhone():int
- setAccount(account:String)
- setPassword(password:String)
- setName(name:String)
- setPhone(phonenumber:String)
Order
- id: int
- attribute: int
- attribute: String
- paid: boolean
- ticketamount: int
- createtime: Date
+ <<constructor>>Order(member: int, payment: String, ticketamount: int, concertid:int, totalprice:int)
+<<constructor>>Order(idorder:int, memberid:int, payment:String, paid:boolean, ticketamonut:int, createtime:Timestamp, concertid:int, totalprice:int)
+ getID(): int
+ getMemberid(): int
+ getPayment(): String
+ getPaid(): boolean
+ getTicketamount(): int
+ getCreatetime(): Date
+ getTotalprice():int
+ getConcertid():int
+ getIdorder():int
+ isPaid(): boolean
+ setIdorder(int):void
+ updateAmount(int):void
+ toJsonData(int):JSONObject
Concert演唱會
- id: int
- name: String
- supplierid: String
- location: String
- picture: String
- seatpicture: String
- endsellingtime: Date
- content: String
- ticketstatus: JSONObject
- concertstarttime: Date
- concertendtime: Date
+ <<constructor>>Concert(int id, String name, int supplierid, String location, String picture, String seatpicture, Date endsellingtime, String content, JSONObject ticketstatus, Date starttime, Date endtime) + getID(): int
+ getConcertName(): String
+ getSupplierid(): int
+ getLocation(): String
+ getPicture(): String
+ getSeatPicture(): String
+ getEndSellingTime(): Date
+ getContent(): String
+ getTicketStatus(): JSONObject
+ getConcertStartTime(): Date
+ getConcertEndTime(): Date
Ticket
- id: int
- concertid: int
- orderid: int
- seatarea: String
- seatid: int
- isused: boolean
- phonenumber: String
- email: String
- name: String
+ <<constructor>>Ticket(concertid: int, orderid: int, seatarea: String, seatid: int, String email, String phonenumber, String name)
+ <<constructor>>Ticket(idticket:int, concertid:int, orderid:int, seatarea: String, seatid: int, isused:boolean: String email, String phonenumber, String name)
+ getIdticket(): int
+ getConcertid(): int
+ getOrderid(): int
+ getSeatarea(): String
+ getSeatid(): int
+ getEmail(): String
+ getIsused(): boolean
+ getName(): String
+ getPhonenumber(): String
+ setSeatid(): int
+ setIdticket(): int
+ toJsonData(): int
+ UsedTicket(): int
+ isUsed(): boolean
0..*
1..*
1
0..*
1..*
0..*
1..*
1..*
0..*
1
0..*
1
1..*
1..*

圖 3：類別圖（2/4）

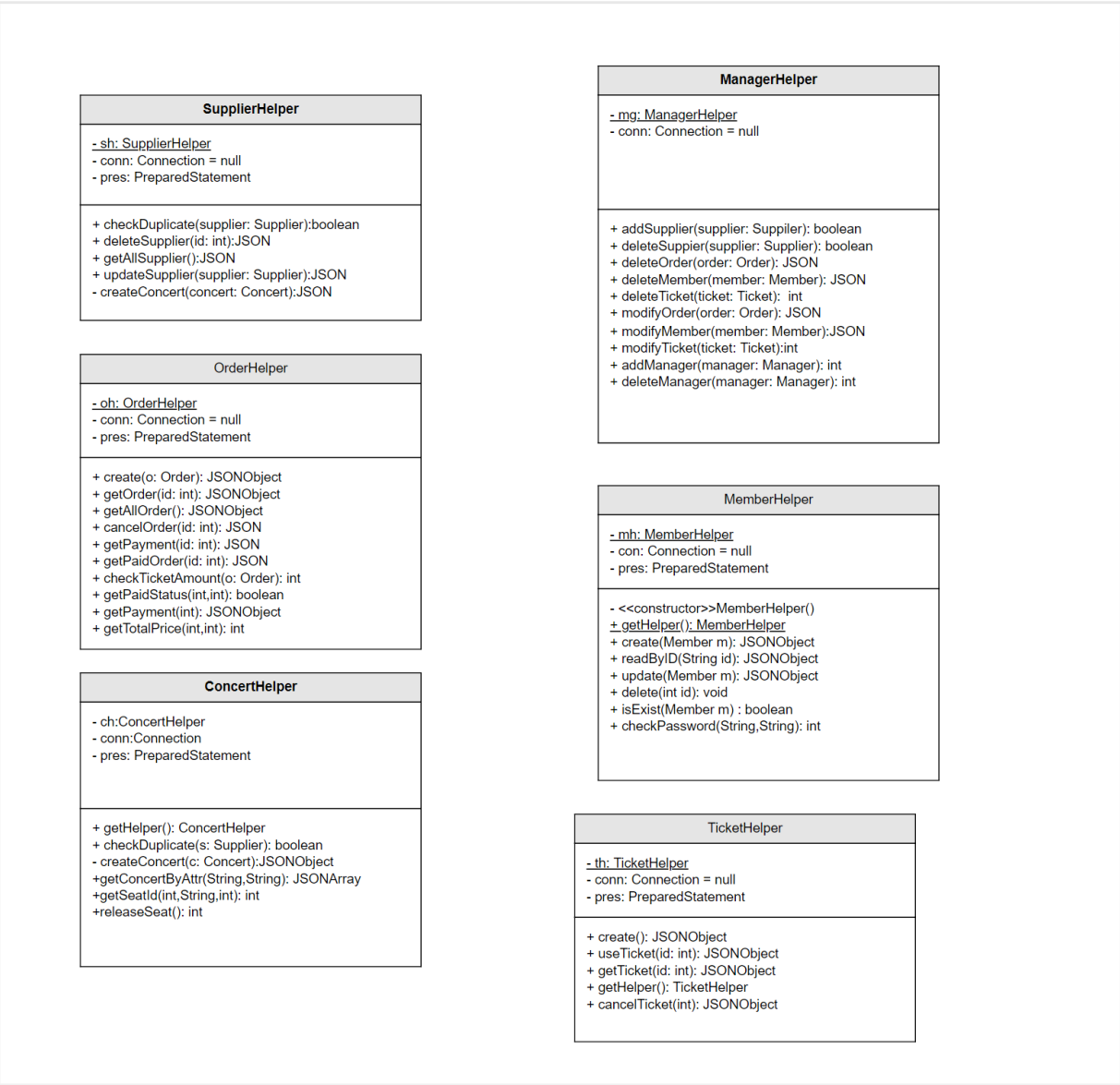

SupplierHelper
- sh: SupplierHelper
- conn: Connection = null
- pres: PreparedStatement
+ checkDuplicate(supplier: Supplier):boolean
+ deleteSupplier(id: int):JSON
+ getAllSupplier():JSON
+ updateSupplier(supplier: Supplier):JSON
- createConcert(concert: Concert):JSON
OrderHelper
- oh: OrderHelper
- conn: Connection = null
- pres: PreparedStatement
+ create(o: Order): JSONObject
+ getOrder(id: int): JSONObject
+ getAllOrder(): JSONObject
+ cancelOrder(id: int): JSON
+ getPayment(id: int): JSON
+ getPaidOrder(id: int): JSON
+ checkTicketAmount(o: Order): int
+ getPaidStatus(int,int): boolean
+ getPayment(int): JSONObject
+ getTotalPrice(int,int): int
ConcertHelper
- ch:ConcertHelper
- conn:Connection
- pres: PreparedStatement
+ getHelper(): ConcertHelper
+ checkDuplicate(s: Supplier): boolean
- createConcert(c: Concert):JSONObject
+getConcertByAttr(String,String): JSONArray
+getSeatId(int,String,int): int
+releaseSeat(): int
ManagerHelper
- mg: ManagerHelper
- conn: Connection = null
+ addSupplier(supplier: Suppiler): boolean
+ deleteSuppier(supplier: Supplier): boolean
+ deleteOrder(order: Order): JSON
+ deleteMember(member: Member): JSON
+ deleteTicket(ticket: Ticket): int
+ modifyOrder(order: Order): JSON
+ modifyMember(member: Member):JSON
+ modifyTicket(ticket: Ticket):int
+ addManager(manager: Manager): int
+ deleteManager(manager: Manager): int
MemberHelper
- mh: MemberHelper
- con: Connection = null
- pres: PreparedStatement
- <<constructor>>MemberHelper()
+ getHelper(): MemberHelper
+ create(Member m): JSONObject
+ readByID(String id): JSONObject
+ update(Member m): JSONObject
+ delete(int id): void
+ isExist(Member m) : boolean
+ checkPassword(String,String): int
TicketHelper
- th: TicketHelper
- conn: Connection = null
- pres: PreparedStatement
+ create(): JSONObject
+ useTicket(id: int): JSONObject
+ getTicket(id: int): JSONObject
+ getHelper(): TicketHelper
+ cancelTicket(int): JSONObject

圖 4：類別圖（3/4）

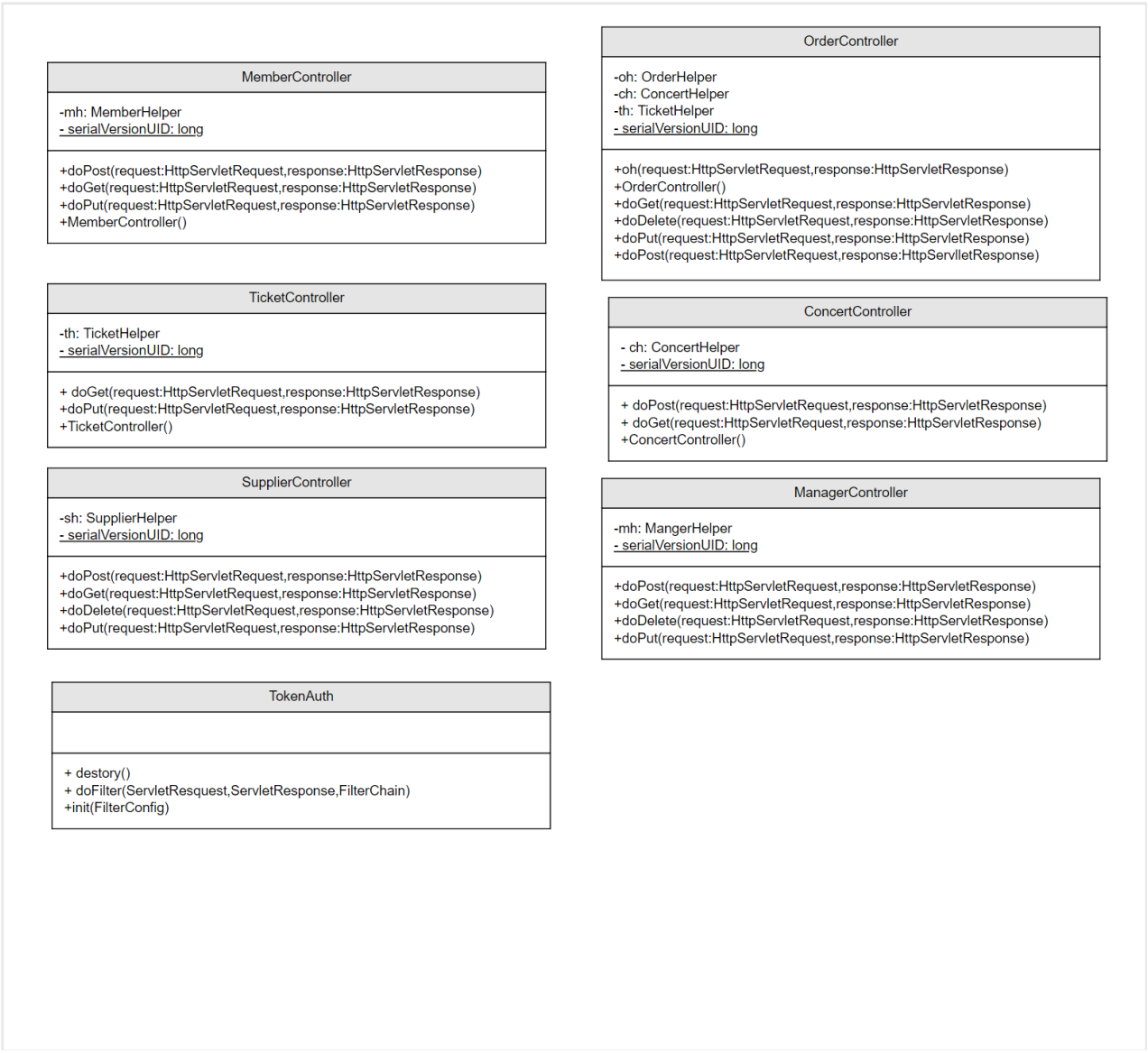

MemberController
-mh: MemberHelper
- serialVersionUID: long
+doPost(request:HttpServletRequest,response:HttpServletResponse)
+doGet(request:HttpServletRequest,response:HttpServletResponse)
+doPut(request:HttpServletRequest,response:HttpServletResponse)
+MemberController()
TicketController
-th: TicketHelper
- serialVersionUID: long
+ doGet(request:HttpServletRequest,response:HttpServletResponse)
+doPut(request:HttpServletRequest,response:HttpServletResponse)
+TicketController()
SupplierController
-sh: SupplierHelper
- serialVersionUID: long
+doPost(request:HttpServletRequest,response:HttpServletResponse)
+doGet(request:HttpServletRequest,response:HttpServletResponse)
+doDelete(request:HttpServletRequest,response:HttpServletResponse)
+doPut(request:HttpServletRequest,response:HttpServletResponse)
TokenAuth
+ destory()
+ doFilter(ServletResquest,ServletResponse,FilterChain)
+init(FilterConfig)
OrderController
-oh: OrderHelper
-ch: ConcertHelper
-th: TicketHelper
- serialVersionUID: long
+oh(request:HttpServletRequest,response:HttpServletResponse)
+OrderController()
+doGet(request:HttpServletRequest,response:HttpServletResponse)
+doDelete(request:HttpServletRequest,response:HttpServletResponse)
+doPut(request:HttpServletRequest,response:HttpServletResponse)
+doPost(request:HttpServletRequest,response:HttpServlletResponse)
ConcertController
- ch: ConcertHelper
- serialVersionUID: long
+ doPost(request:HttpServletRequest,response:HttpServletResponse)
+ doGet(request:HttpServletRequest,response:HttpServletResponse)
+ConcertController()
ManagerController
-mh: MangerHelper
- serialVersionUID: long
+doPost(request:HttpServletRequest,response:HttpServletResponse)
+doGet(request:HttpServletRequest,response:HttpServletResponse)
+doDelete(request:HttpServletRequest,response:HttpServletResponse)
+doPut(request:HttpServletRequest,response:HttpServletResponse)

圖 5：類別圖（4/4）

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，本搶票系統預計共有 5 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六個子系統，其中包含以下：①會員子系統、②訂單子系統、③訂票子系統、④票券子系統、⑤供應商子系統、⑥管理者子系統，如下

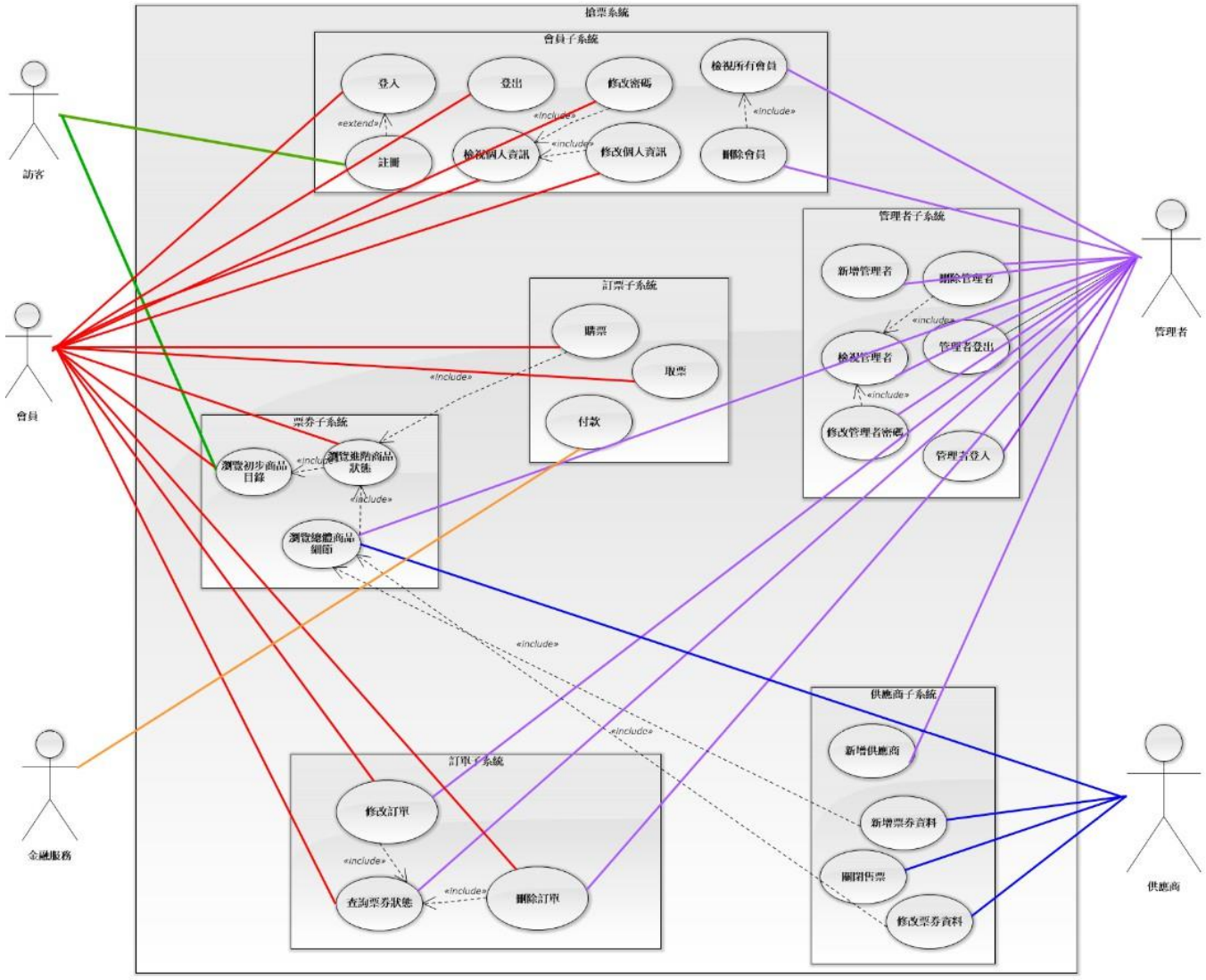


圖 6：搶票系統使用案例圖

4.2 Use Case 實做之循序圖

4.2.1 訂票子系統

此使用案例限一般訪客完成使用案例 1.2 會員登入，才得以執行。會員間接進行購票，需要藉由使用者輸入需要的演唱會時間、票券數量、票券區域、購票人個人資訊、付款方式，其中訂單編號和訂單成立時間由系統自行帶入，非使用者手動計算來完成此項功能。購票動作成功的話，會同時減少演場會的剩餘票券數量。

與該模組相關之頁面於下表進行說明：

表 7：後台管理者會員管理模組關聯頁面

HTML	關聯 Use Case	說明
buyticket.html	Use Case 3.1	會員購票頁面

4.2.1.1 Sequence Diagram—Use Case 3.1 購買票券(建立訂單)

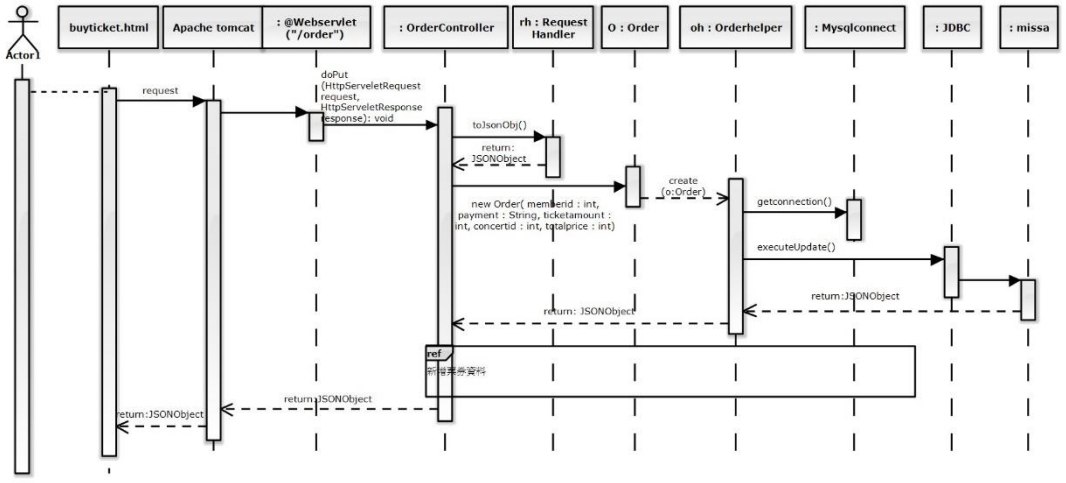


圖 7：商業流程編號 3.1 購買票券循序圖(建立訂單)

1. 購票者進入購票頁面，選填演唱會資料後，前端驗證完會透過 javascript 的 ajax 送出 post request。
2. 後端以 MemberController 之 doPost()進行處理，以 RequestHandler 取回

request 之參數，並 create 一個 order 物件。

3. 使用 orderhelper 物件的 create()方法來建立一個新訂單。
4. 呼叫 Mysqlconnect 的 getconnection 來連結資料庫。
5. 呼叫 Mysqlconnect 的 executeupdate()來執行寫好的 sql，以新增一個 new request.
6. 成功新增訂單後，會新增票券資料

表 8：商業流程編號 3.1 購買票券（建立訂單）Business Exception List

http status code/message	發生之 method	說明
400/新增票券失敗，超過上限！	submit()	單場演唱會購票數量超過上限

4.2.1.2 Sequence Diagram—Use Case 3.1 購買票券 (新增票券)

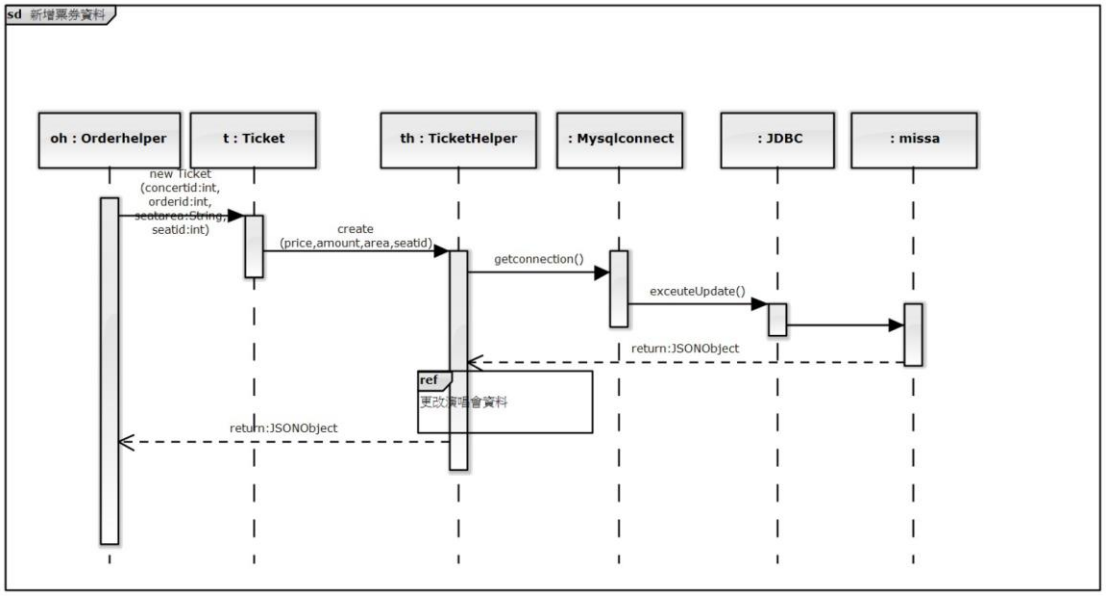


圖 8：商業流程編號 3.1 購買票券循序圖片段（新增票券）

1. 當 missa 資料庫回傳結果-訂單成功新增，OrderHelper 會根據數量新增票券(ticket)物件，並新增 tickethelper 來建立票券資訊，包含演唱會資料、區域、價格這些票券資訊。
2. TicketHelper 會透過 mysqlconnect 和資料庫取得連結，並使用 executeUpdate()來執行 sysql 指令，新增票券。
3. 成功新增票券後，會更改演場會資料。

4.2.1.3 Sequence Diagram—Use Case 3.1 購買票券 (更改演唱會資料)

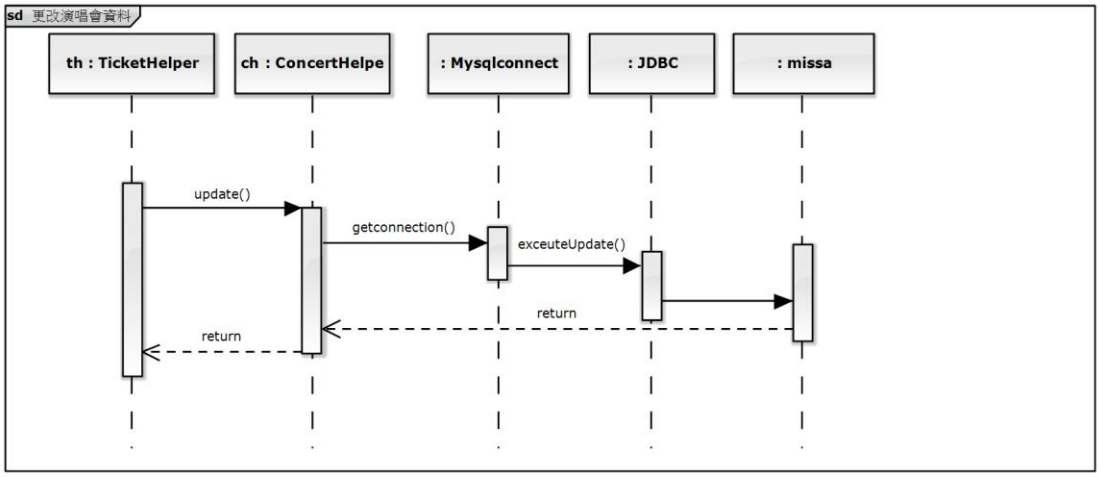


圖 9：商業流程編號 1.3 會員更改資訊（管理者）循序圖（更新資料）

1. TicketHelper 會使用 concerthelper 的 update()來更新票券內容
1. Concerthelper()會透過 getconnection()來連結資料庫，用 executeUpdate 執行寫好的 sql 並取得結果。
2. 成功更新資料庫後，回傳更新成功之結果。

4.2.2 供應商子系統

1. Use Case 7.1 新增票券資料：

   此使用案例限一般訪客完成使用案例 1.2 會員輸入供應商資料，成為供應商後才得以執行。供應商輸入演唱會時間、演唱會地點、演唱會描述內容、演唱會名稱、演唱會開始結束時間、演唱會票價、座位圖、演唱會封面圖檔、演唱會截止售票時間、販賣總票券數量、販賣票券區域、販賣票券剩餘數量，其中演唱會編號和訂單成立時間由系統自行帶入，非使用者手動計算來完成此項功能。

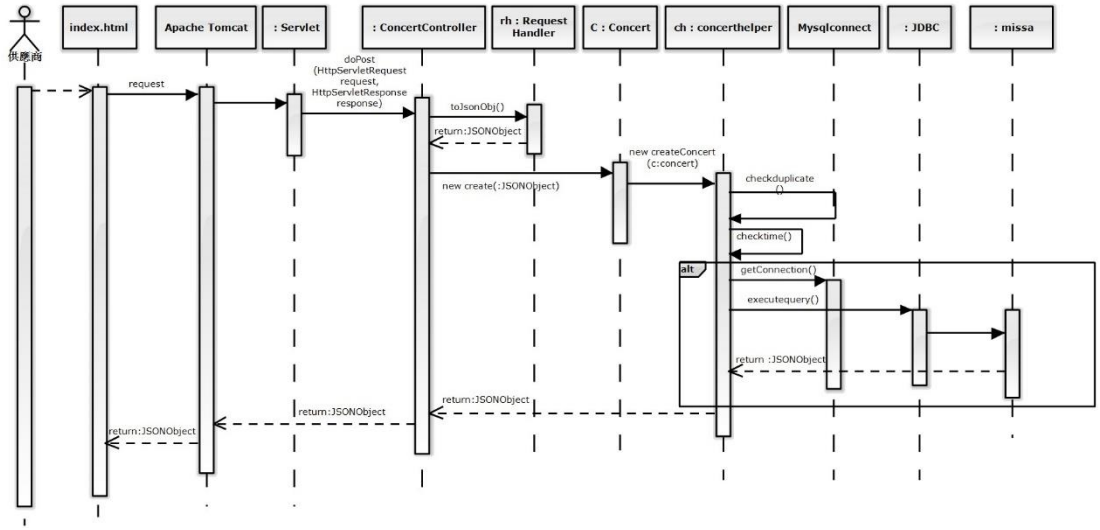
與該模組相關之頁面於下表（**錯誤! 找不到參照來源。**）進行說明：

表 9：後台管理者會員管理模組關聯頁面

HTML	關聯 Use Case	說明
addconcert.html	Use Case 7.1	供應商新增票券資料頁面

4.2.2.1 Sequence Diagram—Use Case7.1 新增票券資料（供應商）

圖 10 :商業流程編號 7.1 新增票券資訊(供應商)循序圖



2. 供應商進入新增演唱會頁面
3. 填寫演唱會資料後送出透過 javascript 的 ajax 送出 post request.
4. 送到後端後，交由 ConcertController 的 doPost 來處理。
5. 進到 doPost 以後會經由 RequestHandler 的 toJsonObj 把 request 轉換成 JSONObject，接著會 new 一個 concert 物件，並將這個 concert 物件交由 ConcertHelper 來做驗證，例如:checkduplicate()跟 checktime，若驗證通過後，呼叫 Mysqlconnect 來存入資料庫。
6. 呼叫 Mysqlconnect 的 getconnection 後執行寫好的 sql，以新增一個 new request.

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 家用版

2　CPU：Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz

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
2. 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. Mozilla Firefox 69 以上
2. Microsoft Edge 44 以上
3. Microsoft Internet Explorer 11 以上
4. Google Chrome 76 以上(需要取消禁止第三方資訊)

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（**錯誤! 找不到參照來源。**）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 8090 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。

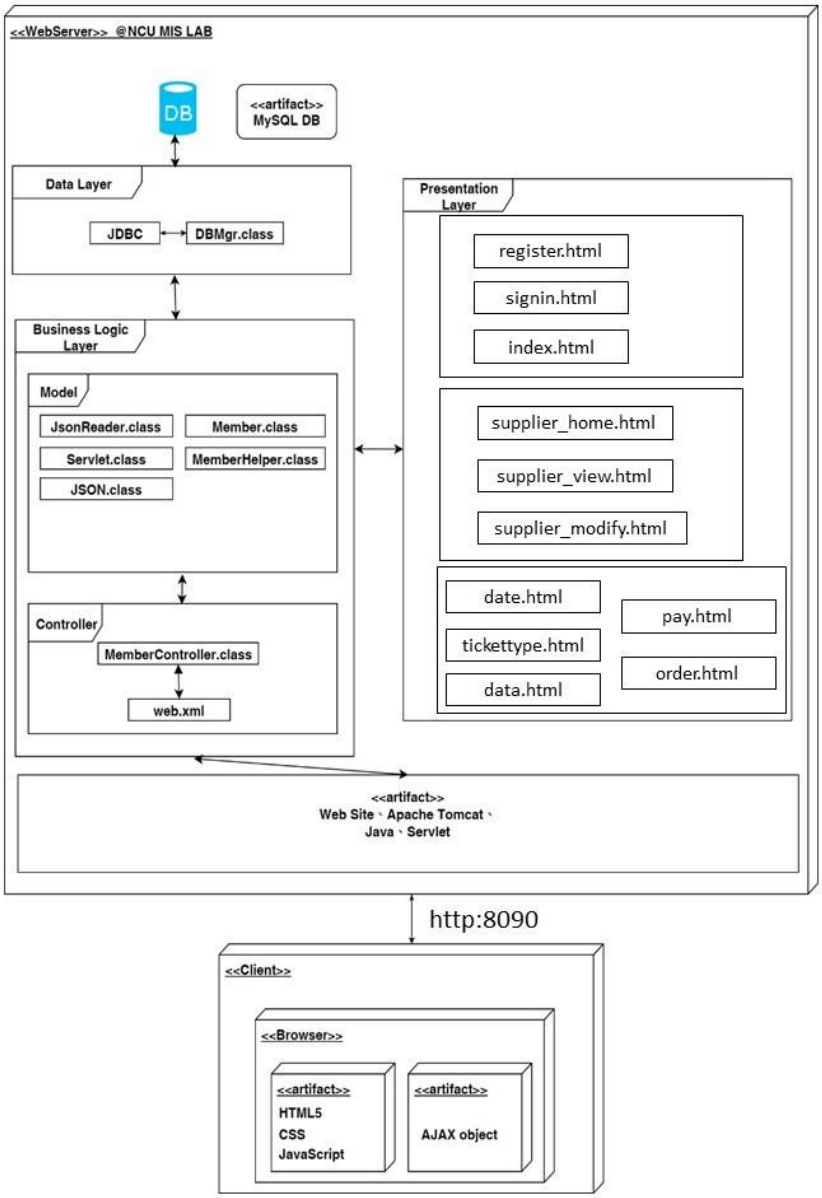


圖 11：系統架構圖

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/group14/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/group14/util/*」資料夾當中，其中包含 Mysqlconnect.java 等
   - ✓ 不同專案共用之工具則會放置到「WEB-INF/classes/ncu/im3069/group14/tools/*」資料夾當中，如；RequestHandler.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/group14/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 12）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("/Auth/order")），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。

```
1  package servletclass;
2
3  import java.io.*;
4  import javax.servlet.*;
5  import javax.servlet.http.*;
6
7  public class MemberController extends HttpServlet {
8
9      public void init() throws ServletException {
10         // Do required initialization
11     }
12
13     public void doPost(HttpServletRequest request, HttpServletResponse response)
14         throws ServletException, IOException {
15       // Do POST Request
16     }
17
18     public void doGet(HttpServletRequest request, HttpServletResponse response)
19         throws ServletException, IOException {
20       // Do GET Request
21     }
22
23     public void doDelete(HttpServletRequest request, HttpServletResponse response)
24         throws ServletException, IOException {
25       // Do DELETE Request
26     }
27
28     public void doPut(HttpServletRequest request, HttpServletResponse response)
29         throws ServletException, IOException {
30       // Do PUT Request
31     }
32 }
```

圖 12：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 13）所示：

A. 使用 try-catch 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()

```
294    public JSONObject cancelOrder(int idorder) {
295        int row = 0;
296        String query = "";
297        String execute_sql = "";
298        JSONObject response = new JSONObject();
299
300        try {
301            conn = Mysqlconnect.getConnect();
302            query = "delete from `missa`.`order` where `idorder` = ? ";
303            pres = conn.prepareStatement(query);
304            pres.setInt(1, idorder);
305
306            row = pres.executeUpdate();
307            execute_sql = pres.toString();
308            System.out.println(execute_sql);
309
310        } catch (SQLException e) {
311            /** 印出JDBC SQL指令錯誤 **/
312            System.err.format("SQL State: %s\n%s\n%s\n", e.getErrorCode(), e.getSQLState(), e.getMessage());
313            e.printStackTrace();
314        } catch (Exception e) {
315            /** 若錯誤則印出錯誤訊息 */
316            e.printStackTrace();
317        } finally {
318            /** 關閉連線並釋放所有資料庫相關之資源 **/
319            Mysqlconnect.close( pres, conn);
320        }
321        response.put("result", "delete order success");
322        response.put("row", row);
323        response.put("orderid", idorder);
324        return response;
325
326    }
```

圖 13：OrderHelper 之取消訂單之 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。

```
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 14：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 15）所示：

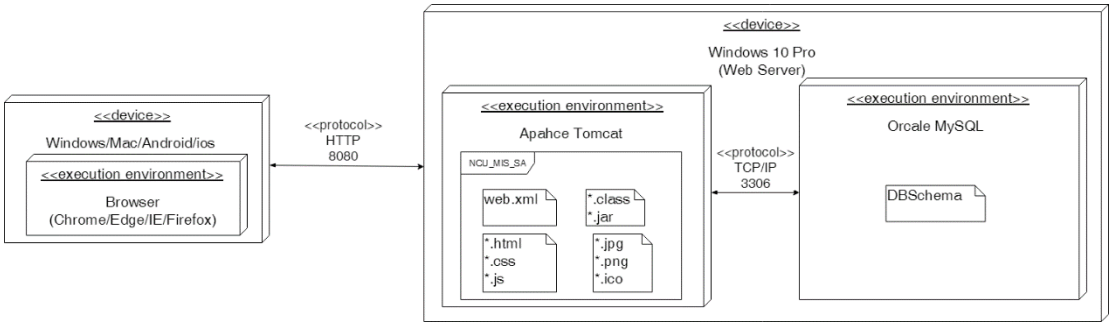


圖 15：專案部署圖

1. 本專案之部署方式，其硬體與軟體規格如前揭（5.1　環境需求）所述。
2. 本專案之網頁伺服器軟體與資料庫同屬相同之伺服器且所有流量皆導向相同之伺服器（Server）。
3. 最終整個專案之檔案將透過匯出封裝成 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 Mysqlconnect 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 8090、資料庫埠號採用 Oracle MySQL 預設之 3306。

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. 使用 tab。

2. 單字組成方式

2.1. 動作：get/set/update/read/is/use/delete/check 等。

2.2. 附加欄位：主要與該欄位之涵蓋範圍有關（例如：paid/all/exist 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： MysqlConnect。

3.2. 主要依照 ER Diagram 進行建立，同時包含所需之 Controller，並加入另外不同四個部分：

- ✓ Controller：命名以*Controller 為主（例如：MemberController）。
- ✓ MysqlConnect.java：管理所有與資料庫相關聯之函式。
- ✓ RequestHandler.java：讀取 Request 之資料與回傳 JSON 格式之函式。
- ✓ Token.java：可以新產生一個 Token，破譯 Token 等等
- ✓ TokenAuth.java：是一個 Filter 跟，用來檢查是否為登入狀態

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，後面單字字首皆大寫」之命名方式。

5.2. 例如：jsonObj，clmBody, jwtCookie，等。

6.2 回傳訊息規範

1. 透過 RequestHandler 類別之 sendJsonRes()、sendJsonErr()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 傳入物件允許傳入 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（**錯誤! 找不到參照來源。**）所示。

   2.1. API 回傳資料之組成包含五個 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。
   - ✓ ContentType
     - 以 application/json 回傳
   - ✓ 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。
   - ✓ CharacterEncoding
     - 設為 UTF-8
   - ✓ response
     - 儲存另一個 JSON 格式物件，可跟隨所需資料擴充裡面的值。

6.3 API 規範

1. 路徑皆採用「/Auth/*」。
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. 傳入之資料可以使用 JSON.stringify ()將物件序列化成 JSON 字串。
5. 回傳之資料透過 RequestHandler 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（**錯誤! 找不到參照來源。**）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。
2. WebContent 存放所有靜態的文件，html 存放在目錄
   A. css Directory 存放 css 檔
   B. js Directory 存放 js 檔
3. 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 RequestHandler.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））。
4. 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。

6.4 專案資料夾架構

下圖（**錯誤! 找不到參照來源。**）為本系統之專案資料夾整體架構：

5. 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。

6. WEB-INF 則存放 Controller 與 Model 之後端檔案：

   E. web.xml：存放網站之路徑（route）資料

   F. 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

   G. 網站上線時，須將所有*.java 檔案編譯成*.class。

   H. 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））。

7. 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。

```
C:.
|  .classpath
|  .project
|  Clist.txt
|
├─.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
|  |  .gitignore
|  |
|  └─classes
|      └─ncu
|          └─im3069
|              └─group14
|                  ├─app
|                  |      Concert.class
|                  |      ConcertHelper.class
|                  |      Member.class
|                  |      MemberHelper.class
|                  |      Order.class
|                  |      OrderHelper.class
|                  |      Ticket.class
|                  |      TicketHelper.class
|                  |
|                  ├─controller
|                  |      ConcertController.class
|                  |      IndexController.class
|                  |      LoginController.class
|                  |      MemberController.class
|                  |      OrderController.class
|                  |      TicketController.class
|                  |      TokenAuth.class
|                  |
|                  ├─tools
|                  |      RequestHandler.class
|                  |
|                  └─util
|                         Mysqlconnect.class
|                         Token.class
|
├─src
|  └─ncu
|      └─im3069
|          └─group14
|              ├─app
|              |      Concert.java
|              |      ConcertHelper.java
|              |      Member.java
|              |      MemberHelper.java
|              |      Order.java
|              |      OrderHelper.java
|              |      Ticket.java
|              |      TicketHelper.java
|              |
|              ├─controller
|              |      ConcertController.java
|              |      IndexController.java
|              |      LoginController.java
|              |      MemberController.java
|              |      OrderController.java
|              |      TicketController.java
|              |      TokenAuth.java
|              |
|              ├─tools
|              |      RequestHandler.java
|              |
|              └─util
|                     MysqlConnect.java
|                     Token.java
|
└─WebContent
   |  concert.html
   |  data.html
   |  date.html
   |  index-signin.html
   |  index.html
   |  member.html
   |  navbar.html
   |  pay.html
   |  register.html
   |  signin.html
   |  spotlights.png
   |  style.css
   |  tickettype.html
   |  www.homeppt.html
   |
   ├─css
   |      register.css
   |
   ├─js
   |      jquery-3.4.1.min.js
   |
   ├─META-INF
   |      MANIFEST.MF
   |
   └─WEB-INF
      |  web.xml
      |
      └─lib
             jackson-annotations-2.9.0.jar
             jackson-core-2.9.6.jar
             jackson-databind-2.9.6.jar
             jjwt-0.9.1.jar
             json-20180813.jar
             mysql-connector-java-8.0.17.jar
             servlet-api.jar
```

圖 16：專案之資料夾架構

6.5 Route 列表

以下表格（**錯誤! 找不到參照來源。**）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 10：Route 表格

Index	Route	Action	網址參數	功能描述/作法
1	/	GET	None	首頁(檢視所有演唱會,分成已登入/未登入)
2	/edit.html	GET	None	會員更改其個人資料
3	/register.html	GET	None	訪客註冊會員頁面
4	/Auth/member	GET	None	取得會員資料
5	/Auth/member	POST	None	新增會員
6	/Auth/member	DELETE	None	刪除會員
7	/Auth/member	PUT	None	更新會員
8	/Auth/order	POST	None	提交訂單資訊
9	/Auth/order	GET	None	顯示我的訂單頁面資訊
10	/Auth/ticket	GET	None	顯示訂單詳細資訊
11	/Auth/concert.do	GET	None	購票頁面更新演唱會/票券資訊
12	/Auth/concert.do	POST	None	供應商新增演唱會資料

下列將根據上表進行更詳細之說明與其運行步驟：

1. /：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. /Auth/member：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. /Auth/member：註冊會員之 API，運行步驟如下：

A. 前端傳入 email、name、password 等 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. /Auth/member：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. /Auth/member：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 等 JSON 物件。

B. 更新 member 資料。

C. 回傳更新成功訊息。

8. /Auth/order : 提交訂單資訊

A. 前端傳入 concertid、seatarea、ticketamount 等 JSON 物件

B. 建立 order

C. 回傳訂票成功資訊

9. /Auth/order : 顯示我的訂單頁面資訊

A. 前端輸入 memberid 判斷其 order

B. 顯示此 memberid 的 order 資訊

C. 回傳成功資訊

10. /Auth/ticket : 顯示訂單詳細資訊

A. 前端輸入 orderrid 判斷其包含的 ticket

B. 顯示此 orderid 的 ticket 資訊

C. 回傳成功資訊

11. /Auth/concert.do : 購票頁面更新演唱會/票券資訊

A. 前端可選擇是否輸入要的 concertid，若無則全部顯示

B. 回傳 name、location、picture、seatpicture、content 等演唱會資訊

C. 回傳成功資訊

12. /Auth/concert.do : 供應商新增演唱會資料

A. 前端傳入 name、location、picture、seatpicture、content 等 JSON 物件。

B. 確認演唱會是否重複，若重複則回傳錯誤資訊

C. 建立 concert

D. 回傳新增成功資訊

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 17）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 17：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 18：常見之 JSON 格式範例

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

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 19）為例：

```
            "seatarea": localStorage.area1,
            "ticketamount": localStorage.area1_q,
            "memberid": localStorage.memberid,
            "name": localStorage.temp_name,
            "phonenumber": localStorage.temp_phone,
            "email": localStorage.temp_mail,
            "payment": paytype,
        }
        var data_string = JSON.stringify(data);

        $.ajax({
            type: "POST",
            url: "order",
            data: data_string,
            crossDomain: true,
            cache: false,
            dataType: 'json',
            timeout: 5000,
            success: function (response) {
                if (response.status == 200) {
                    alert("成功!");
                }
            },
            error: function () {
                alert("無法連線到伺服器！");
            }
        });
    }
```

圖 19：提交購票訂單之程式碼

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 20）顯示會訂票資訊之欄位，根據檢所之結果進行回填方式：

```
success: function (data) {


    //演唱會名字
    $("#concert_name").html(data.name);

    //逐一顯示
    $.each(data.ticketstatus, function (i, item) {
        $("#quantitynow" + i).html("剩下" + item.quantity + "張");

        $("#concert" + i).html(item.content);
    })
```

圖 20：更新訂票資訊欄位回填

若查詢之資料要以表格方式呈現，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 21）以更新 SQL 查詢結果之表格為例：

```
        success: function (data) {
            $.each(data, function (i, item) {

                var temp = "";
                temp += '<tr><th scope="row">' + item.concertname + '</th>';
                temp += '<td>' + item.seatarea + '</td>';
                temp += '<td>' + item.seatid + '</td>';
                temp += '<td>' + item.name + '</td>';
                temp += '<td>' + item.email + '</td>';
                temp += '<td>' + item.phonenumber + '</td></tr>'

                $(tbody).append(temp);
            });
```

圖 21：更新 SQL 查詢結果之表格

7.2 Requesthandler、JSONObject 與 JSONArray 操作

為簡化取回前端所傳入之 JSON 資料，本範例提供 Requesthandler class 用於協助處理，為使用此 class 需要將該 Requesthandler.java 檔案放置，並且將整個 java 專案資料夾 import（import ncu.im3069.group14.tools. Requesthandler;），若要操作 JSONObject 則必須 import（import org.json.*;），程式碼範例如下圖（圖 22）所示：

```
public void sendJsonRes(JSONObject data,HttpServletResponse response) throws IOException {
    PrintWriter out = response.getWriter();
    response.setContentType("application/json; charset=UTF-8");
    response.setStatus(200);
    out.print(data);
    out.flush();
}

public void sendJsonErr(JSONObject data, HttpServletResponse response) throws IOException {
    PrintWriter out = response.getWriter();
    response.setContentType("applicatiion/json");
    response.setCharacterEncoding("UTF-8");
    response.setStatus(400);
    out.print(data);
    out.flush();
}
public String getMemberIDinRequest() {

    Cookie[] cookies = this.req.getCookies();

    String token = null;

    for (Cookie c:cookies) {
        System.out.println("Name:"+c.getName());
        if(c.getName().equals("Token")) {
            token = c.getValue();
            System.out.println("Incoming token: "+token);
            break;
```

圖 22：Requesthandler 操作之範例

下表（表 11）將呈現 JSONObejct 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 11：JSONObject 之操作範例

var data = {
"concertid": localStorage.concertid,
"seatarea": localStorage.areal,
"ticketamount": localStorage.areal_q,
"memberid": localStorage.memberid,
"name": localStorage.temp_name,
"phonenumber": localStorage.temp_phone,
"email": localStorage.temp_mail,
"payment": paytype,
"seatid": 100,
"totalprice":200
}
	public Concert(JSONObject obj) {
	this.name = obj.getString("concertName");
	this.supplierId = obj.getInt("supplierId");
	this.location = obj.getString("location");
	this.picture = obj.getString("picture");
	this.seatpicture = obj.getString("seatpicture");
	this.endsellingtime = obj.getString("endsellingtime");
	this.content = obj.getString("content");
	this.ticketstatus = obj.getJSONObject("ticketstatus");
	this.concertstarttime = obj.getString("concertstarttime");
	this.concertendtime = obj.getString("concertendtime");
	}

範例 JSON 格式	範例取出 JSONObject 之內容

JSONObject data = ch.createConcert(c);
JSONObject resp = new JSONObject();
resp.put("status", "200");
resp.put("message", "成功新增演唱會");
resp.put("row-effect", data);
 	
/** 透過JsonReader物件回傳到前端（以JSONObject方式） */
jsr.sendJsonRes(resp, response);
	無

JSONObject 之操作方式	

7.3 Mysqlconnect 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 Mysqlconnect 之類別進行封裝，以下節錄說明 Mysqlconnect 之實作方式，詳細之內容可參閱 Mysqlconnect 類別。

以下圖（22）新增演唱會為例，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 22 第 45 行透過 Mysqlconnect.getConnect()建立連線。

B. 圖 22 第 48 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 23）所示，並以 setString()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 22 第 61 行若為檢索則是 executeQuery()，回傳為 ResultSet、其餘指令如圖 23 第 61 行為 executeUpdate()回傳為 int 整數，影響之行數。

```
43          try {
44              /** 取得資料庫之連線 */
45              conn = Mysqlconnect.getConnect();
46              /** sql指令 */
47              String sql = "INSERT INTO testconcert(name,supplierid,location,picture,seatpicture,endsellingtime,content,t
48              pres = conn.prepareStatement(sql);
49              pres.setString(1, c.getConcertName());
50              pres.setString(2, c.getSupplierId().toString());
51              pres.setString(3, c.getLocation());
52              pres.setString(4, c.getPicture());
53              pres.setString(5, c.getSeatPicture());
54              pres.setString(6, c.getEndSellingTime());
55              pres.setString(7, c.getContent());
56              pres.setString(8, c.getTicketStatus().toString());
57              pres.setString(9, c.getConcertStartTime());
58              pres.setString(10, c.getConcertEndTime());
59              System.out.println(pres.toString());
60              /** 執行新增之SQL指令並記錄影響之行數 */
61              row = pres.executeUpdate();
62
63          }catch(SQLException e) {
64              /** 印出JDBC SQL指令錯誤 **/
65              System.err.format("SQL State: %s\n%s\n%s", e.getErrorCode(), e.getSQLState(), e.getMessage());
66          } catch (Exception e) {
67              /** 發錯誤則印出錯誤訊息 */
68              e.printStackTrace();
69          } finally {
70              /** 關閉連線並釋放所有資料庫相關之資源 **/
```

圖 23：新增演唱會 concertHelper createConcert() method（節錄）