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 24 組

企管三　106401551　王鈺豪

資管三B 106403038　涂育慈

資管三A 106403508　高語萱

資管三B 106403544　劉宇哲

資管三A 106403027　呂健穩

指導教授：許智誠　教授、陳以錚　教授

中 華 民 國 １０８ 年 １２ 月 ３１ 日

目錄

表目錄.....4
圖目錄.....5
版本修訂.....6
第 1 章　簡介.....7
1.1　文件目的.....7
1.2　系統範圍.....7
1.3　參考文件.....7
1.4　文件架構.....7
第 3 章　類別圖.....14
第 4 章　系統循序圖.....1
4.1　使用案例圖.....1
4.2　Use Case 實做之循序圖.....2
4.2.1　商業流程編號 4.0：結帳模組.....2
4.2.1.1　Sequence Diagram—Use Case 4.1 結帳（會員）.....3
4.2.1.2　Sequence Diagram—Use Case 4.4 新增客戶資訊（會員） 4
第 5 章　系統開發環境.....6
5.1　環境需求.....6
5.1.1　伺服器硬體.....6
5.1.2　伺服器軟體.....6
5.1.3　前端套件.....7
5.1.4　後端套件.....7
5.1.5　客戶端使用環境.....7
5.2　專案架構.....8
5.2.1　系統架構圖.....8
5.2.2　MVC 架構.....9
5.3　部署.....13
第 6 章　專案撰寫風格.....15
6.1　程式命名風格.....15
6.2　回傳訊息規範.....16
6.3　API 規範.....17
6.4　專案資料夾架構.....18
6.5　Route 列表.....20
6.6　程式碼版本控制（參考用）.....21
第 7 章　專案程式設計.....22
7.1　JSON.....22
7.1.1　JSON 格式介紹.....22

7.1.2 前端發送 AJAX Request 說明.....23
7.1.3 前端表格 Render 與欄位回填.....25
7.2 JsonReader、JSONObject 與 JSONArray 操作.....25
7.3 DBMgr 與 JDBC.....26

表目錄

Table 1 會員資料表（tblmember）之資料結構.....10
Table 2 房間資料表（tblroom）之資料結構.....11
Table 3 旅館資料表（tblhotel）之資料結構.....11
Table 4 國家資料表（tblcountry）之資料結構.....12
Table 5 城市資料表（tblcity）之資料結構.....12
Table 6 訂單資料表（tblorder）之資料結構.....12
Table 7 後台管理者會員管理模組關聯頁面.....2
Table 8 Route 表格.....20
Table 9 JSONObject 和 JSONArray 之操作範例.....26

圖目錄

Figure 1 實體關聯圖.....10
Figure 2 類別圖(1/3).....14
Figure 3 類別圖(2/3).....14
Figure 4 類別圖(3/3).....1
Figure 5 PaperPlane 線上訂房系統使用案例.....1
Figure 6 商業流程編號 4.1 結帳（會員）循序圖.....3
Figure 7 商業流程編號 4.4 新增客戶資訊（會員）循序圖.....4
Figure 8 系統架構圖.....8
Figure 9 Servlet 之 Controller 範例模板（以 MemberController 為例）.....11
Figure 10 MemberHelper 之檢查會員電子郵件是否重複之 Method12
Figure 11 DBMgr 類別內之資料庫參數.....13
Figure 12 專案部署圖.....13
Figure 13 MemberController 之 GET 取得回傳之資料格式範例...17
Figure 14 專案之資料夾架構.....19
Figure 15 本專案所必須 import.....22
Figure 16 常見之 JSON 格式範例.....23
Figure 17 註冊會員之程式碼.....24
Figure 18 更新會員欄位回填.....25
Figure 19 更新 SQL 查詢結果之表格.....25
Figure 20 JsonReader 操作之範例.....26
Figure 21 新增會員之 MemberHelper create() method（節錄）..28
Figure 22 檢索所有會員之 MemberHelper getAll() method（節錄）.....29
Figure 23 操作 JDBC 之模板.....30

版本修訂

版本	修訂者	修訂簡述	日期
V0.1.0	王鈺豪、 涂育慈、 高語萱、 劉宇哲	完成初稿	2019/12/19
V0.2.0	王鈺豪、 涂育慈、 高語萱、 劉宇哲	完成定稿	2019/12/26
V0.3.0	王鈺豪、 涂育慈、 高語萱、 劉宇哲	校稿	2020/01/05

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

本系統範圍用於線上訂房網站，其中主要包含會員、旅館房間資訊、追蹤清單、結帳、訂單管理、店家、房間管理與管理員等八個模組，並且能進行相關瀏覽、新增、修改、刪除、查詢與維護工作，藉由此系統支持完成線上訂房所需的管理流程。詳細各模組之功能與內容可參閱第一份文件系統軟體需求規格書。

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

本專案之線上訂房網站提供使用者整合式的線上訂房、訂單管理與會員資料維護等服務；也提供旅宿業者公布房型資訊以接受線上預訂、訂單管理與業者資料維護等服務，對於管理者來說，亦能以簡便方式進行資料維護等後台作業，希冀不僅能提供最新且詳盡之房型說明，更能提供良好之訂房服務。

以下分析階段之資料庫設計採用實體關係圖（Entity-Relation Diagram）表示，並根據管理者與使用者之需求進行歸納與整理初步之系統條件。

以下詳述系統之資料庫需求，並將其整理成下圖之實體關係圖共計包含 6 個實體（Entity）、6 個關係（Relationship）：

1. 一般訪客可以註冊成為會員/店家且必須以電子郵件作為登入帳號使用，同時系統會自動給予每位會員編號。
2. 會員可將欲訂購之房間加入追蹤清單，但伺服器不儲存該資料，而存於使用者之本地端。
3. 會員可對追蹤清單內之房間進行結帳，在輸入入住日期、退房日期及欲訂購之房間數並選擇付款方式後始成立訂單，爾後可隨時查詢訂單詳細資料。
4. 旅宿業者可以管理房間資訊之異動，但不可進行結帳。
5. 管理者可以管理會員/店家之資料及訂單之異動，並且進行維護作業，同時要記錄帳號建立時間與異動之動作。

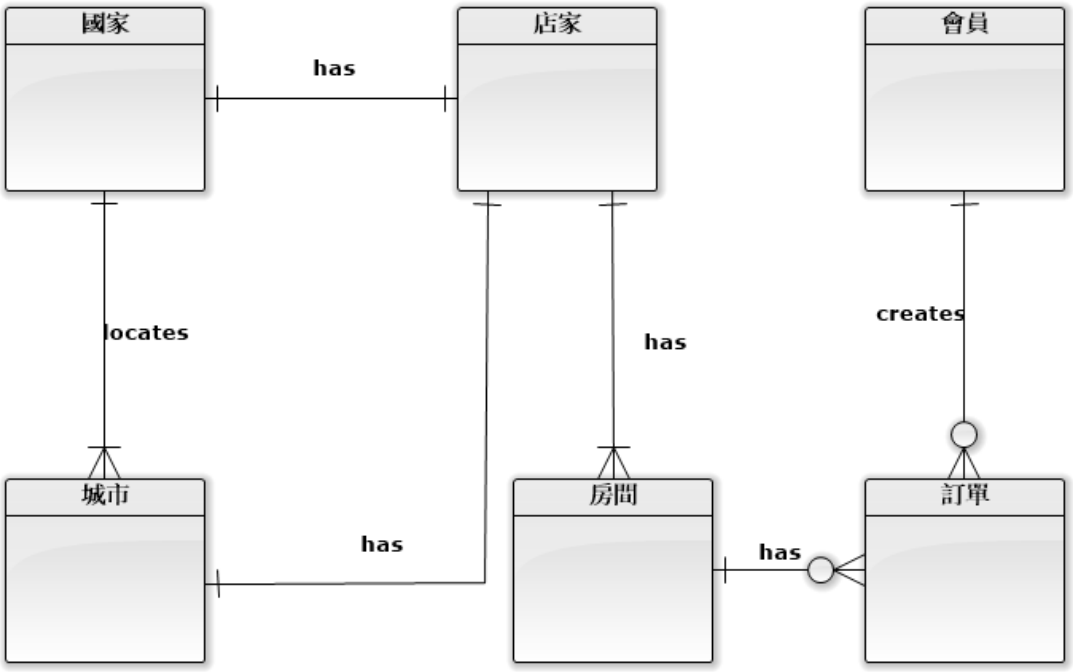


Figure 1　實體關聯圖

根據實體關係圖分析本專案所需之資料庫架構，以下將針對每張資料表進行描述，由於本範例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之分析呈現於此：

1. 會員資料表 (tblmember) :

Table 1 會員資料表 (tblmember) 之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K.	member_id	int(11)	無	否	✓	utf8
	member_firstname	varchar(30)	無	否		utf8
	member_lastname	varchar(30)	無	否		utf8
	member_email	varchar(50)	無	否		utf8
	member_password	varchar(30)	無	否		utf8
	member_modified	datetime	無	否		utf8
	member_created	date	無	否		utf8
	member_dob	date	無	否		utf8

- member_id：為自動增加，作為會員編號，由資料庫自動產生，不可更動。
- member_modified：記錄會員資料最後一次的更新時間。
- member_created：記錄會員註冊的時間點。

2. 房間資料表 (tblroom)：

Table 2 房間資料表 (tblroom) 之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K.	room_id	int(11)	無	否	✓	utf8
F.K.	hotel_id	int(11)	無	否		utf8
	room_name	varchar(30)	無	否		utf8
	room_price	varchar(30)	無	否		utf8
	room_image	varchar(255)	無	否		utf8

- room_id：為自動增加，作為房間編號，由資料庫系統自動產生，不可更動。
- room_image：記錄房間圖片檔案之路徑。
- hotel_id：為外來鍵，記錄該房間是由哪間旅館提供

3. 旅館資料表 (tblhotel)：

Table 3 旅館資料表 (tblhotel) 之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K.	hotel_id	int(11)	無	否	✓	utf8
	hotel_name	varchar(30)	無	否		utf8
	hotel_email	varchar(50)	無	否		utf8
	hotel_password	varchar(30)	無	否		utf8
	hotel_modified	datetime	無	否		utf8
	hotel_created	date	無	否		utf8
	hotel_image	varchar(255)	無	否		utf8
	hotel_description	varchar(255)	無	否		utf8
	hotel_phone	varchar(15)	無	否		utf8
F.K. F.K.	hotel_country_id hotel_city_id hotel_address	int(11) int(11) varchar(50)	無 無 無	否 否 否		utf8 utf8 utf8

- hotel_id：為自動增加，作為旅館編號，由資料庫系統自動產生，不可更動。
- hotel_modified：記錄旅館資料最後一次的更新時間。
- hotel_created：記錄旅館註冊的時間點。
- hotel_image：記錄旅館圖片檔案之路徑位置。
- hotel_country_id：為外來鍵，記錄該旅館位於哪個國家。
- hotel_city_id：為外來鍵，記錄該旅館位於哪個城市。

4. 國家資料表 (tblcountry)：

Table 4 國家資料表 (tblcountry) 之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K.	country_id	int(11)	無	否	✓	utf8
	country_name	varchar(30)	無	否		utf8

- country_id：為自動增加，作為國家編號，由資料庫系統自動產生，不可更動。

5. 城市資料表 (tblcity)：

Table 5 城市資料表 (tblcity) 之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K.	city_id	int(11)	無	否	✓	utf8
	country_id	int(11)	無	否		utf8
	city_name	varchar(30)	無	否		utf8

- city_id：為自動增加，作為城市編號，由資料庫系統自動產生，不可更動。
- country_id：為外來鍵，記錄該城市是位於哪個國家。

6. 訂單資料表 (tblorder)：

Table 6 訂單資料表 (tblorder) 之資料結構

Key	名稱	類型	預設值	空值	自動增加	編碼
P.K.	order_id	int(11)	無	否	✓	utf8

	member_id	int(11)	無	否		utf8
F.K.	room_id	int(11)	無	否		utf8
	room_amount	int(11)	無	否		utf8
	total_price	int(11)	無	否		utf8
	checkin_date	date	無	否		utf8
	checkout_date	date	無	否		utf8
	customer	varchar(30)	無	否		utf8
	paid_status	tinyint(4)	無	否		utf8
	created	date	無	否		utf8
	hotel_id	int(11)	無	否		utf8
	room_name	varchar(30)	無	否		utf8
	room_price	varchar(30)	無	否		utf8
	room_image	varchar(255)	無	否		utf8

- order_id：為自動產生，作為訂單編號，由資料庫系統自動產生，不可更動。
- member_id：記錄該訂單是由哪一位會員成立。
- room_id：為外來鍵，記錄該訂單訂購哪個房間。
- checkin_date：記錄會員/住客即將入住之日期。
- checkout_date：記錄會員/住客退房之日期。
- customer：記錄該訂單之實行人。
- paid_status：記錄訂單之付款狀態，預設值為 0 表示該筆訂單尚未付款。
- created：記錄訂單成立之時間。
- hotel_id：記錄該房間之所屬旅館。
- room_name：記錄房間名稱。之所以於此處再次記錄房間詳細資訊，是確保即使日後已無本次訂購之房間可以訂購，會員仍可知道自己先前訂購之房間紀錄

第 3 章　類別圖

下圖(Figure 2、Figure 3、Figure 4)是依據電子商務線上訂購系統的分析模型和建立的互動圖，以及實體關係圖（Entity-Relation Diagram）所繪製之設計階段之類別圖（Class Diagram），用於描述系統的類別集合，包含其中之屬性，與類別之間的關係。

本階段之類別圖屬於細部（detail）之設計圖，與上一份文件分析階段之類別圖需要有詳細之變數型態、所擁有之方法，依據這些設計原則，本類別圖之說明如下所列：類別圖除包含與資料庫相對應之物件外，亦包含相關之控制物件（controller）、DBMgr 與各功能相對應資料庫操作類別（例如：MemberHelper）和相對應之類別工具（JsonReader）。

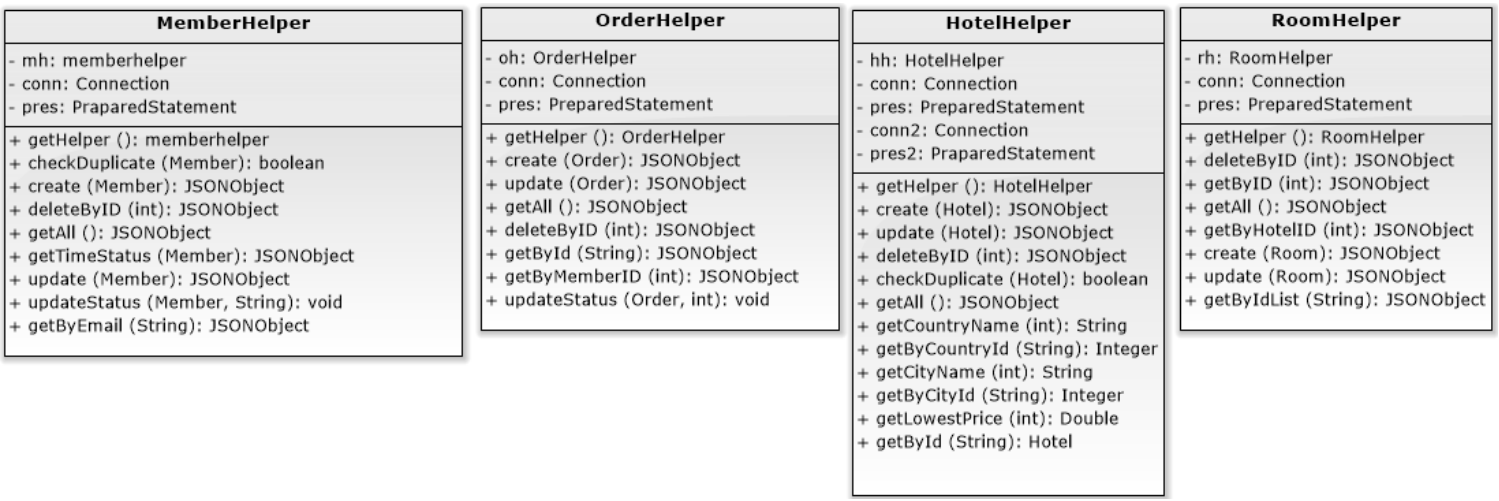


Figure 2 類別圖(1/3)

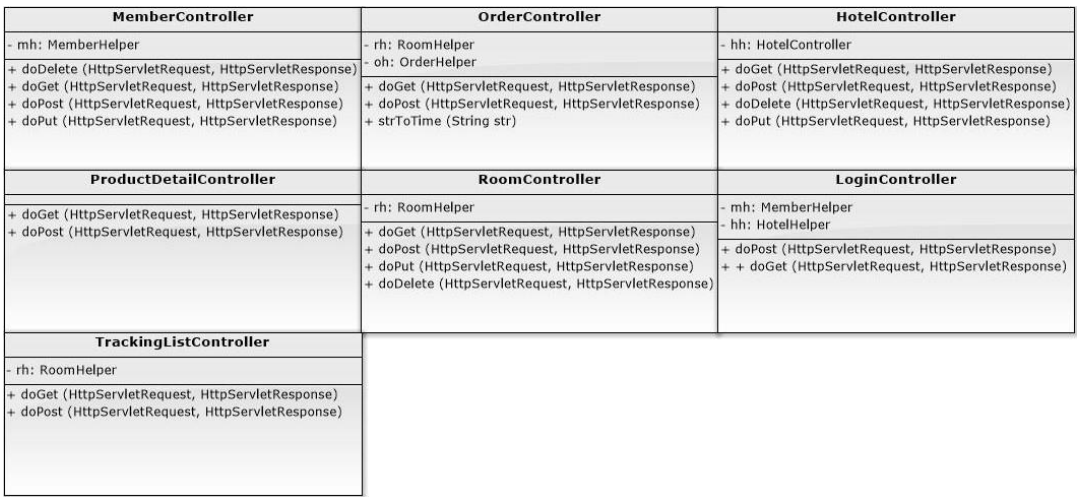


Figure 3 類別圖(2/3)

Figure 4 類別圖(3/3)

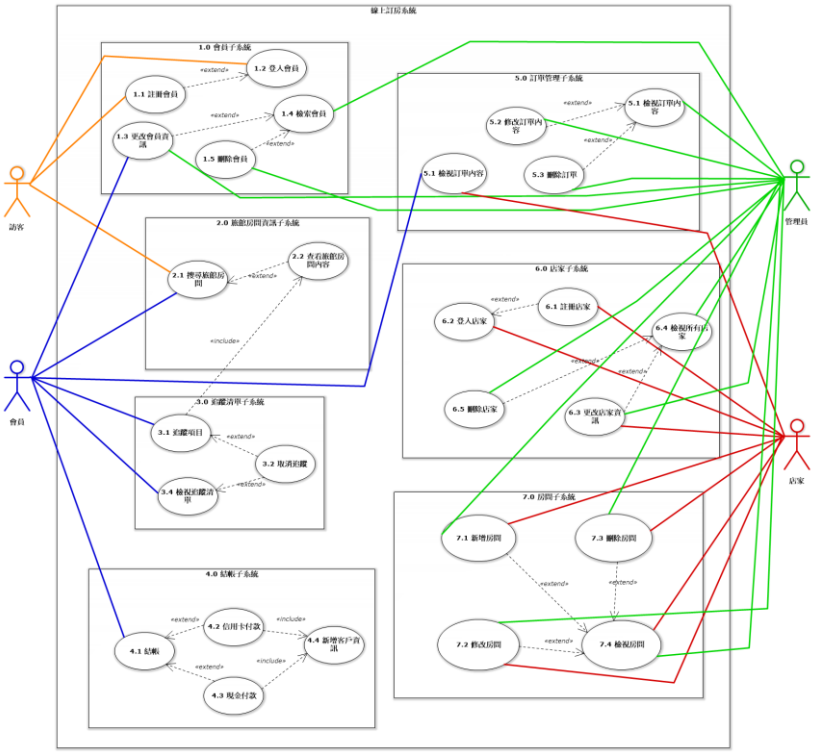
第 4 章　系統循序圖

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—系統軟體需求規格書（Software Requirement Specification），本線上訂房網站系統預計共有 4 位動作者與 32 個使用案例，並依照不同之模組區分成不同子系統共計八個子系統，其中包含以下：①會員子系統、②旅館房間資訊子系統、③追蹤清單子系統、④結帳子系統、⑤訂單管理子系統、⑥店家子系統、⑦房間管理子系統，如下圖所示：

Figure 5　PaperPlane 線上訂房系統使用案例



4.2 Use Case 實做之循序圖

4.2.1 商業流程編號 4.0：結帳模組

在 Use Case 4.0 中所使用到的功能（包含：結帳、信用卡付款、現金付款、新增客戶資訊），以下簡要說明兩個循序圖的功能：「使用案例 4.1：結帳」、「使用案例 4.4：新增客戶資訊」：

1. Use Case 4.1：結帳

   此使用案例用於說明會員如何選擇預訂房間進行結帳，會員需選擇入退住日期和客房數，以傳送訂購資料到後端進行資料交換。

2. Use Case 4.4：新增客戶資訊

   此使用案例用於說明會員如何輸入預訂者及住客的資料（姓名、電話與電郵），以建立訂單完成訂單交易。

與該模組相關之頁面於下表（Figure 6、7）進行說明：

Table 7 後台管理者會員管理模組關聯頁面

HTML	關聯 Use Case	說明
productDetail.html	Use Case 4.1	旅館房間詳細頁面
Checkout_3.html	Use Case 4.4	新增客戶資訊頁面

4.2.1.1 Sequence Diagram—Use Case 4.1 結帳（會員）

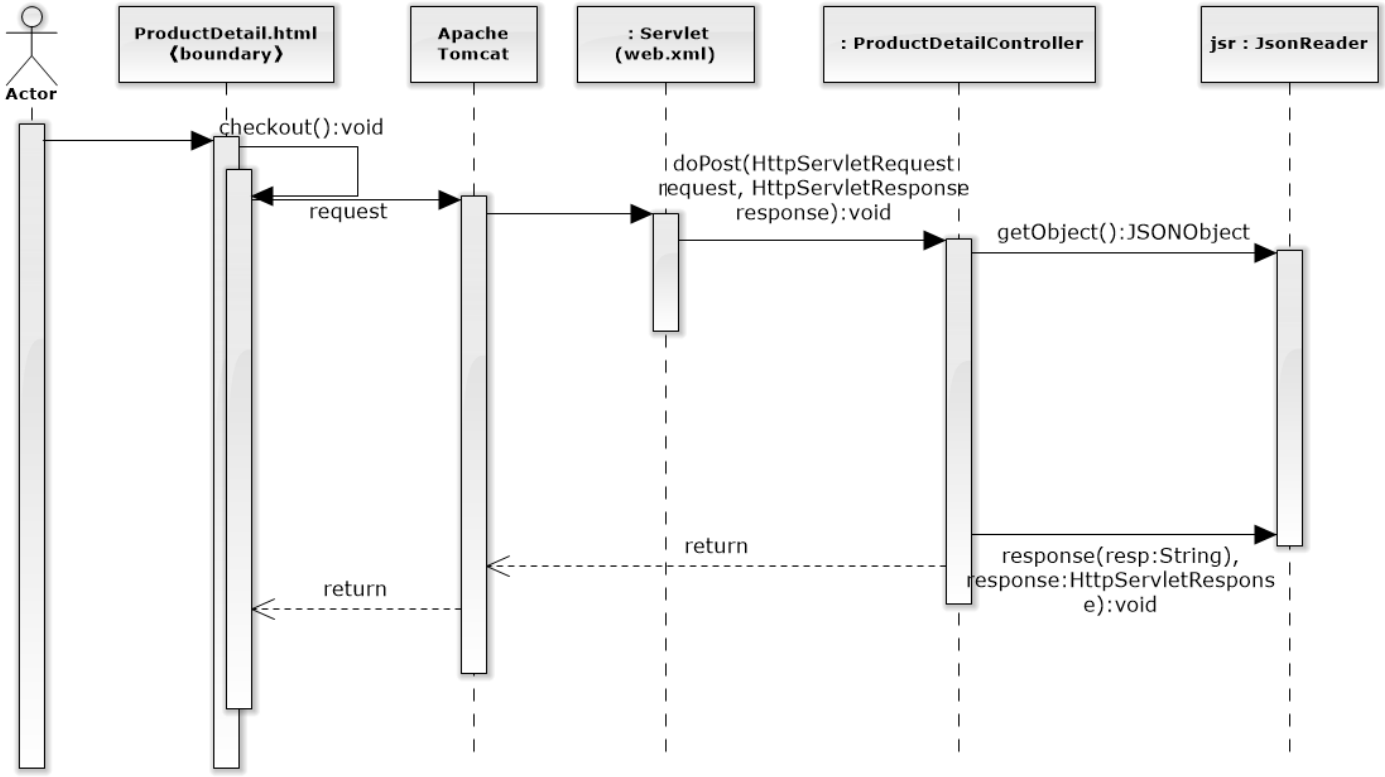


Figure 6　商業流程編號 4.1 結帳（會員）循序圖

1. 會員完成商業流程編號「 1.2 會員登入」，以及完成商業流程編號「 2.1 搜尋旅館房間」、「 2.3 查看旅館房間內容」，點擊進入該旅館房間詳細頁面(productDetail.html)
2. 會員完成介面所需資料之填寫，點擊「預訂房間」按鈕並通過前端之資料驗證後，透過 JavaScript 之 checkout()送出 POST 請求。
3. 將訂購的房間資訊、入退住日期及客房數，從後端以 ProductDetailController 之 doPost()進行處理，以 JsonReader 取回 request 之參數。之後**不經過資料庫，只以 api 進行資料的交換。**
4. POST 請求成功，則進入確認預定房間頁面(checkout_1.html)。

4.2.1.2 Sequence Diagram—Use Case 4.4 新增客戶資訊（會員）

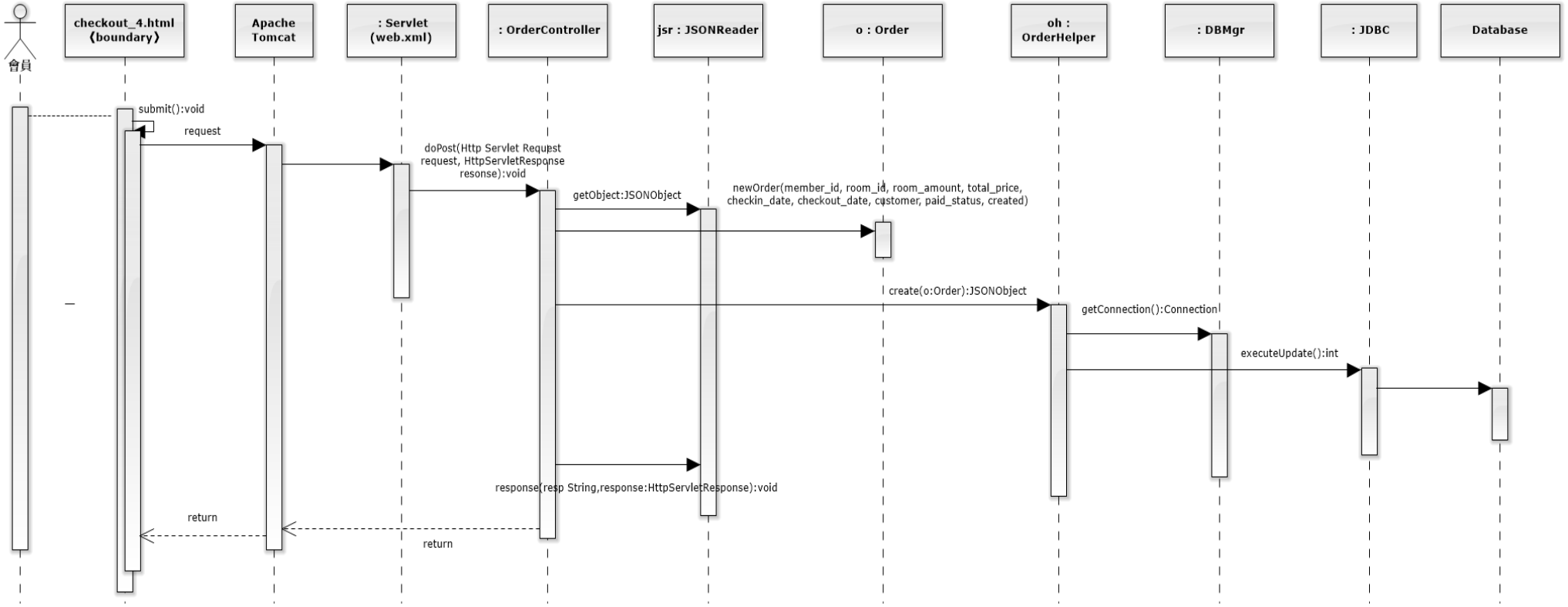


Figure 7　商業流程編號 4.4 新增客戶資訊（會員）循序圖

1. 會員完成「4.2 信用卡付款」或「4.3 現金付款」後，進入新增客戶資訊頁面(checkout_3.html)。
2. 當會員完成填寫並通過前端之資料驗證後，透過 JavaScript 之 submit()送出 POST 請求。
3. 後端以 OrderController 之 doPost()進行處理，以 JsonReader 取回 request 之參數，再以 OrderHelper()之 create()新增該訂單資料至資料庫當中。
4. 訂單新增成功，系統將回傳新增成功之結果並進入確認訂單頁面(checkout_4.html)。

第 5 章　系統開發環境

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

5.1　環境需求

5.1.1　伺服器硬體

本專案預計部署之設備如下：

1　OS：Microsoft Windows 10 Pro 1903 x64

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

3　RAM：16.0 GB

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

5.1.2　伺服器軟體

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

1　Java JDK Version：Oracle JDK 1.8.0_202

2　Application Server：Apache Tomcat 9.0.24

3　Database：Oracle MySQL Community 8.0.17.0

4　Workbench：Oracle MySQL Workbench Community Edition 8.0.17

5　IDE：Microsoft Visual Studio Code 1.37.1

5.1　專案類型：Java Servlet 4.0

5.2　程式語言：Java

5.1.3 前端套件

1. JQuery-3.4.1
2. CakePHP CSS

5.1.4 後端套件

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

5.1.5 客戶端使用環境

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

1. Google Chrome 76 以上
2. Mozilla Firefox 69 以上
3. Microsoft Edge 44 以上
4. Microsoft Internet Explorer 11 以上

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

5.2 專案架構

5.2.1 系統架構圖

本專案系統整體架構如下圖（Figure 8）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。

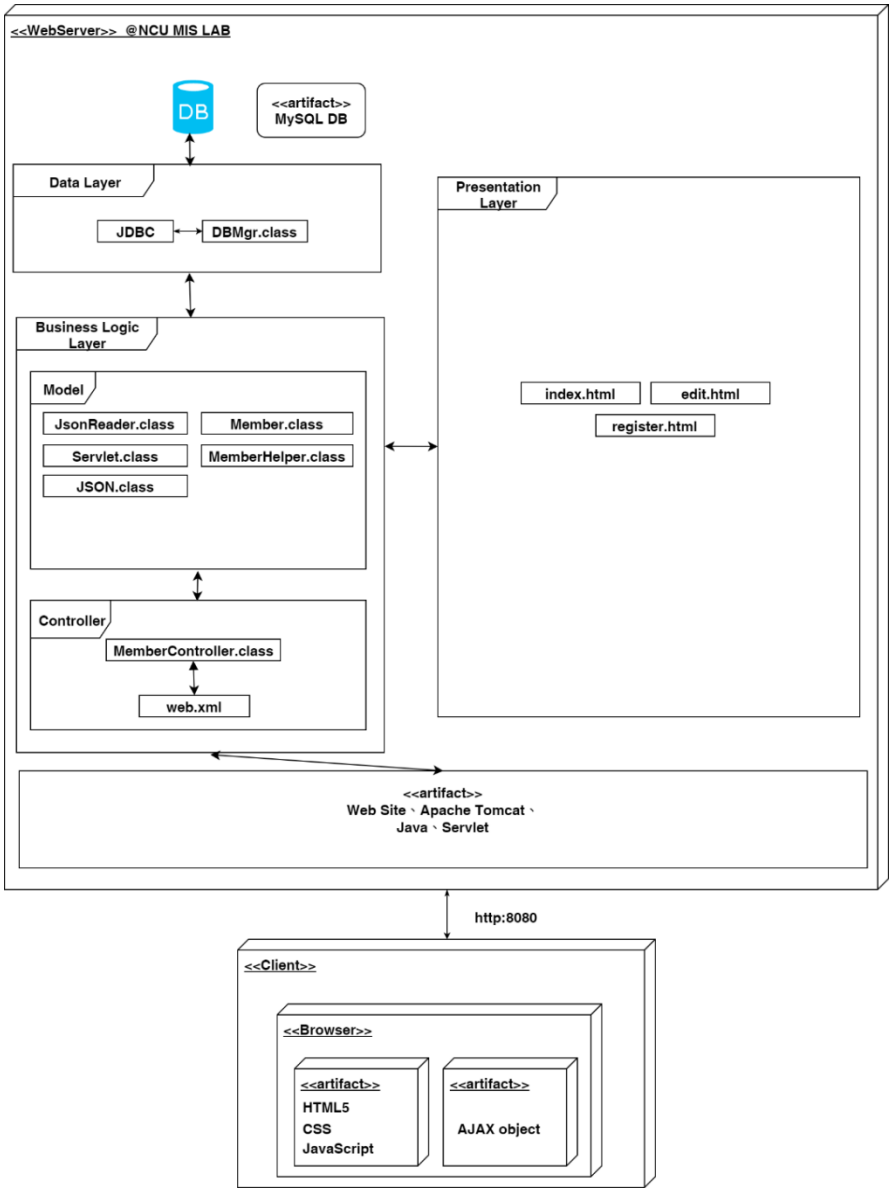


Figure 8 系統架構圖

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 WEB-INF 目錄下
   - ✓ 編譯指令：javac -encoding UTF-8 -classpath lib/* classes/*/*.java
2. Business Model：MVC-Model
   - ✓ 主要放置於「WEB-INF/classes/ncu/im3069/*/app/*」資料夾當中，主要有許多類別（class）於其中，如：Member.java 等。
   - ✓ 主要用於處理邏輯判斷資料查找與溝通，並與資料層（DB）藉由 JDBC 進行存取，例如：SELECT、UPDATE、INSERT、DELETE。

- ✓ 其他功用與雜項之項目則統一會放置到「WEB-INF/classes/ncu/im3069/*/util/*」資料夾當中，其中包含 DBMgr.java 等
- ✓ 不同專案共用之工具則會放置到「WEB-INF/classes/ncu/im3069/tools/*」資料夾當中，如；JsonReader.java 其類別可以被其他工具共用。
- ✓ 不同專案可以使用不同資料夾（package）進行區分，以增加分類之明確度。

3. View Controls：MVC-Controller
   - ✓ 主要放置於「WEB-INF/classes/ncu/im3069/*/controller/*」資料夾當中。
   - ✓ Controller 主要為 View 和 Model 之溝通橋梁，當前端 View 發送 AJAX Request 透過 Servlet 提供之 WEB-INF 內的 web.xml 檔案指定處理的 Controller，並由後端之 Java 進行承接。
   - ✓ 主要用於控制各頁面路徑（Route）與功能流程，規劃之 use case 於此處實作，主要將 model 所查找之數據進行組合，最終仍以 JSON 格式回傳給使用者。
   - ✓ 實做 Controller 應依照以下之類別（class）之範例進行撰寫。
   - ✓ 命名會以*Controller 進行命名，同時本類別必須將 HttpServlet 進行 extends，下圖（Figure 9）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
    }
}
```

Figure 9　Servlet 之 Controller 範例模板（以 MemberController 為例）

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

5.2.2.3　資料層（Data Layer）

1. 作為資料庫取得資料的基本方法之用，藉由第三方模組 JDBC 進行實現。
2. 透過編寫 DBMgr 之 class 進行客製化本專案所需之資料庫連線內容。
3. 透過 import 此 DBMgr class 能套用到不同的功能當中，以 MemberHelper.class 為例，該 class 管理所有有關會員之方法。
   - ✓ 以下以檢查會員帳號是否重複為例，如下圖（Figure 10）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;
}
```

Figure 10　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（Figure 11）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;
}
```

Figure 11　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（Figure 12）所示：

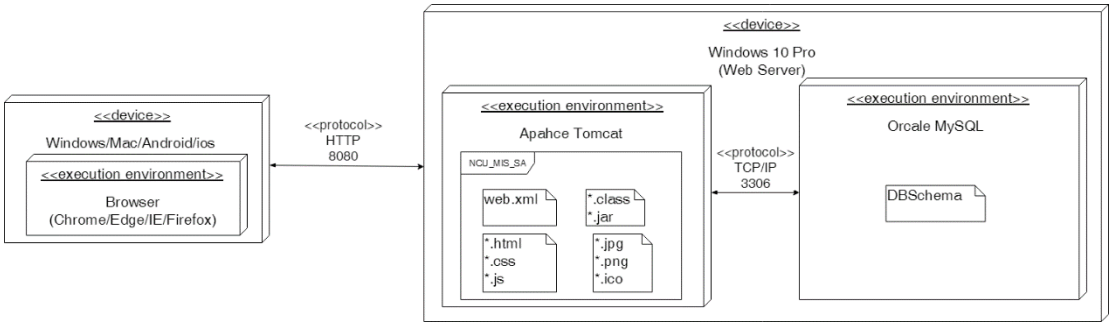


Figure 12　專案部署圖

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。

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. Tab 和空格不限。

1.2. code 一行超過 80 個字元即折行。

2. 單字組成方式

2.1. 動作：get/do/delete/check 等。

2.2. 附加欄位：主要與該欄位之涵蓋範圍有關（例如：duplicate/all 等）。

2.3. 主要關聯之資料庫資料表。

3. 類別（Class）

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

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

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

4. 函式（Method）

4.1. 主要採用「小駝峰式命名法（lower camel case）」，首字皆為小寫，第二單字開始首字為大寫。

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

5. **變數（Variable）**

5.1. 採用「首字皆小寫以底線區隔」之命名方式。

5.2. 例如：exexcute_sql、pre_stmt、start_time 等。

5.3. 區域變數縮寫組字不受此限制。

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（Figure 13）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。

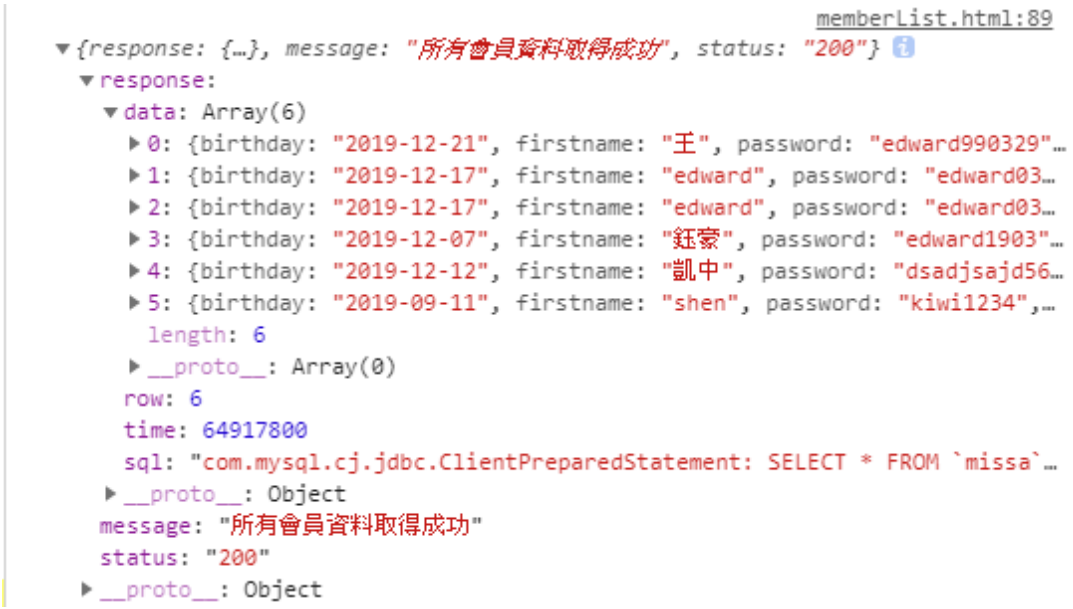


Figure 13　MemberController 之 GET 取得回傳之資料格式範例

6.3　API 規範

1. 路徑皆採用「api/*」。
2. API 採用 AJAX 傳送 JSON 物件。
3. 透過實作 Servlet 之方法，其對應如下：
   3.1. GET：用於取得資料庫查詢後之資料。
   3.2. POST：用於新增資料與登入。
   3.3. DELETE：用於刪除資料。
   3.4. PUT：用於將資料進行更新作業。
4. 傳入之資料需要使用 JSON.stringify ()將物件序列化成 JSON 字串。
5. 回傳之資料透過 JsonReader()內之 method 封裝成 JSONObject 物件（同為 JSON 格式）進行回傳，同時回傳之物件帶有狀態碼之資料。

6.4 專案資料夾架構

下圖（Figure 14）為本系統之專案資料夾整體架構：

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

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

Figure 14 專案之資料夾架構

6.5 Route 列表

以下表格（Table 9）為會員相關模組頁面之 Route 列表，並以 memberList.html 為例，依順序逐項進行功能說明，此 Route 之規劃可依照所實作之功能複雜度與結果進行描述。

Table 8 Route 表格

Index	Route	Action	網址參數	功能描述/作法
1	/login.html	GET	None	會員登入頁面
2	/registerMemberForUser.html	GET	None	會員註冊頁面
3	/MemberList.html	GET	None	管理者檢索會員資料頁面(會員清單)
4	/editMember.html	GET	id	會員更新資料頁面
5	/api/member.do	GET	None	取得會員資料
6	/api/member.do	POST	None	新增會員
7	/api/member.do	DELETE	None	刪除會員
8	/api/member.do	PUT	None	更新會員

下列將根據上表進行更詳細之說明與其運行步驟：

1. /login.html：取得登入頁面（render page）
2. /registerMemberForUser.html：取得會員註冊頁面（render page）
3. /MemberList.html：取得會員清單頁面(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。
4. /editMember.html：取得會員更新資料頁面(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。
5. /api/member.do：取得會員資料之 API，運行步驟如下：

A. 前端可選擇是否傳入會員編號參數（id）。

B. 若不帶參數則表示為請求資料庫所有會員資料。

C. 帶參數則表示為請求資料庫中該名會員資料。

D. 回傳取得結果

6. /api/member.do：新增會員之 API，運行步驟如下：

A. 前端傳入 firstname、lastname、email、password、modified、created、dob 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。

7. /api/member.do：刪除會員之 API，運行步驟如下：

A. 前端傳入會員編號之 JSON 物件。

B. 刪除會員資料。

C. 回傳刪除成功訊息。

8. /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 firstname、lastname、password、modified 之 JSON 物件。

B. 更新 member 資料。

C. 回傳更新成功訊息。

6.6 程式碼版本控制（參考用）

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

本專案於 GitHub 上採用公開程式碼倉庫（public repositories）進行軟體開發，GitHub 允許註冊與非註冊用戶進行瀏覽，並可隨時隨地將專案進行 fork 或是直接 clone 專案進行維護作業，同時本專案僅限執行人員可以進行發送 push 之請求，最後說明文件 readme 檔案採用 markdown 格式進行編輯，本專案之 Github 網址為：https://github.com/jason40418/ncu_mis_sa。

第 7 章　專案程式設計

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

本專案所需要 import 之項目為下圖（Figure 15）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
```

Figure 15 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」。

Figure 16 常見之 JSON 格式範例

上圖（Figure 16）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（Figure 17）為例：

```
function submit() {
                    var firstname = $('#member_firstname').val();
                    var lastname = $('#member_lastname').val();
                    var email = $('#member_email').val();
                    var password = $('#member_password').val();
                    var birthday =  $('#member_dob').val();
                    $("#test1").text = birthday;
                    var email_rule = /^\w+((-\w+)|(\.\w+))*\@[A-Za-z0-9]+((\.|-)[A-Za-z0-9]+)*\.[A-Za-z]+$/;
                    var password_rule = /^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{8,}$/;

                    if (!email_rule.test(email)) {
                        alert("Email格式不符！");
                    }
                    else if(!password_rule.test(password)) {
                        alert("密碼格式不符，長度至少8，且至少包含一個數字和英文字母！");
                    }
                    else {
                        // 將資料組成JSON格式
                        var data_object = {
                            "firstname": firstname,
                            "lastname": lastname,
                            "email": email,
                            "password": password,
                            "birthday" : birthday
                        };

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

                        // 發出POST的AJAX請求
                        $.ajax({
                                type: "POST",
                                url: "api/member.do",
                                data: data_string,
                                crossDomain: true,
                                cache: false,
                                dataType: 'json',
                                timeout: 5000,
                                success: function (response) {
                                    $('#flashMessage').html(response.message);
                                    $('#flashMessage').show();
                                    if(response.status == 200){
                                        updateSQLTable(response.response);
                                    }
                                },
                                error: function () {
                                    alert("無法連線到伺服器！");
                                }
                        });
                    }
                }
```

Figure 17 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（Figure 18）顯示會員更新之欄位，根據檢所之結果進行回填方式：

```
if(response.status == 200){
                            updateSQLTable(response.response);
                            document.getElementById('member_firstname').value = response['response']['data'][0]['firstname'];
                            document.getElementById('member_lastname').value = response['response']['data'][0]['lastname'];
                            document.getElementById('member_email').value = response['response']['data'][0]['email'];
                            document.getElementById('member_password').value = response['response']['data'][0]['password'];
                            document.getElementById('member_modified_times').value = response['response']['data'][0]['modifiedTimes'];
                            document.getElementById('member_created_times').value = response['response']['data'][0]['createdTimes'];
                            document.getElementById('member_dob').setAttribute("value",response['response']['data'][0]['birthday']);
                        }
                        console.log(response);
```

Figure 18 更新會員欄位回填

若查詢之資料要以表格方式呈現，可先將表格之 tbody 部分進行清空，爾後使用字串方式組成 table 之元素，最終使用 append()之 method 將元素回填，下圖（Figure 19）以更新 SQL 查詢結果之表格為例：

```
function updateSQLTable(data) {
                    var time = (data.time / 1000000).toFixed(2);
                    var table_html = "";

                    sql_num += 1

                    table_html += '<tr>';
                    table_html += '<td>' + sql_num + '</td>';
                    table_html += '<td>' + data.sql + '</td>';
                    table_html += '<td style="text-align: right">' + '0' + '</td>';
                    table_html += '<td style="text-align: right">' + data.row + '</td>';
                    table_html += '<td style="text-align: right">' + data.row + '</td>';
                    table_html += '<td style="text-align: right">' + time + '</td>';
                    table_html += '</tr>';
                    $("#sql_log > tbody").append(table_html);
                    $("#sql_summary").html("(default) " + data.row + " queries took " + time + " ms");
                }
```

Figure 19 更新 SQL 查詢結果之表格

7.2 JsonReader、JSONObject 與 JSONArray 操作

為簡化取回前端所傳入之 JSON 資料，本範例提供 JsonReader class 用於協助處理，為使用此 class 需要將該 JsonReader.java 檔案放置，並且將整個 java 專案資料夾 import（import ncu.im3069.tools.JsonReader;），若要操作 JSONObject 則必須 import（import org.json.*;），程式碼範例如下圖（Figure 20）所示：

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

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

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

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


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

Figure 20 JsonReader 操作之範例

下表（Table 10）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。

Table 9 JSONObject 和 JSONArray 之操作範例

``// 取回 { "name" : "test", "age" : 20, "major" : ["資訊管理", "企業管理"], "other" : { "birthday" : "1998-01-01", "like" : "dance" } }```	```JSONObject o = new JSONObject(str); System.out.println(o.getString("name")); System.out.println(o.getInt("age")); System.out.println(o.getJSONArray("major")); System.out.println(o.getJSONObject("other"));```
範例 JSON 格式	範例取出 JSONObject 之內容
``// JSONObject操作 // 新增物件 JSONObject jsonObject = new JSONObject(); // 放入key-value jsonObject.put("UserName", "ZHULI"); // 放入Array jsonObject.element("Array", arrayList);```	```// JSONArray操作 // 新增物件 JSONArray jsonArray = new JSONArray(); // 放入key-value jsonArray.add(0, "ZHULI"); // 印出 print：System.out.println("jsonArray1：" jsonArray);```
JSONObject 之操作方式	JSONArray 之操作方式

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（Figure 21）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. Figure 22 透過 DBMgr.getConnection()建立連線。

B. Figure 22 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（Figure 23）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. Figure 24 若為檢索則是 executeQuery()，回傳為 ResultSet、其餘指令如 Figure 25 第 33 行為 executeUpdate()回傳為 int 整數，影響之行數。

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

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

        try {
            /** 取得資料庫之連線 */
            conn = DBMgr.getConnection();
            /** SQL指令 */
            String sql = "INSERT INTO `missa`.`tblmember`(`member_firstname`,`member_lastname`, "
                    + "`member_email`, `member_password`, `member_modified`, `member_created`,"
                    + " `member_dob`) VALUES(?, ?, ?, ?, ?, ?, ?);";

            /** 取得所需之參數 */
            String fname = m.getFirstName();
            String lname = m.getLastName();
            String email = m.getEmail();
            String password = m.getPassword();


            /** 把birthday從date型態轉為date型態 */
            SimpleDateFormat birthdaysdf = new SimpleDateFormat("yyyy/MM/dd");
            String birthday = birthdaysdf.format(m.getBirthday());

            /** 將參數回填至SQL指令當中 */
            pres = conn.prepareStatement(sql);
            pres.setString(1, fname);
            pres.setString(2, lname);
            pres.setString(3, email);
            pres.setString(4, password);
            pres.setString(5, LocalDateTime.now().plusHours(8).toString());
            pres.setString(6, LocalDateTime.now().plusHours(8).toString());
            pres.setString(7, birthday);

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

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

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

        /** 紀錄程式結束執行時間 */
        long end_time = System.nanoTime();
        /** 紀錄程式執行時間 */
        long duration = (end_time - start_time);

        /** 將SQL指令、花費時間與影響行數，封裝成JSONObject回傳 */
        JSONObject response = new JSONObject();
        response.put("sql", exexcute_sql);
        response.put("time", duration);
        response.put("row", row);

        return response;
    }
```

Figure 27 新增會員之 MemberHelper create() method（節錄）

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

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

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

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

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

                /** 將 ResultSet 之資料取出 */
                int id = rs.getInt("member_id");
                String fname = rs.getString("member_firstname");
                String lname = rs.getString("member_lastname");
                String email = rs.getString("member_email");
                String password = rs.getString("member_password");
                /** 將每一筆會員資料產生一名新Member物件 */
                m = new Member(id, email, password, fname, lname);
                /** 取出該名會員之資料並封裝至 JSONsonArray 內 */
                jsa.put(m.getData());
            }

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

        /** 紀錄程式結束執行時間 */
        long end_time = System.nanoTime();
        /** 紀錄程式執行時間 */
        long duration = (end_time - start_time);

        /** 將SQL指令、花費時間、影響行數與所有會員資料之JSONArray，封裝成JSONObject回傳 */
        JSONObject response = new JSONObject();
        response.put("sql", exexcute_sql);
        response.put("row", row);
        response.put("time", duration);
        response.put("data", jsa);

        return response;
    }
```

Figure 28 檢索所有會員之 MemberHelper getAll() method（節錄）

SQL 資料庫之操作，大致上以下圖（Figure 29）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。

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

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

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

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

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

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

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

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

Figure 29 操作 JDBC 之模板