

國立中央大學資訊管理學系

軟體設計規格書 (SADD)

專案名稱： Exchange Platform (卡片交換平台)

版本： 1.0

報告日期： 2025/12/12

第八組

114423020 陳欣妤

114423037 蕭筑云

114423051 張譽麒

114423068 廖承偉

指導教授:許智誠 教授

目錄

第一章 簡介.....	5
1.1 文件目的.....	5
1.2 文件使用範圍.....	5
1.3 預期讀者.....	6
1.4 參考文件.....	6
第二章 系統架構與運行環境.....	7
2.1 系統定位與目標.....	7
2.2 系統主要功能.....	7
2.3 系統運作流程概述.....	9
2.4 系統架構風格.....	11
2.5 系統外部整合與依賴關係.....	14
2.6 部屬與運行環境.....	16
第三章 系統架構設計.....	19
3.1 架構視圖總覽.....	19
3.2 分層類別視圖（Layered Architecture Class View）.....	20
3.3 模組依賴關係（Modules Dependency Overview）.....	22
3.4 橫切關注點（Cross-cutting Concerns）.....	22
3.5 架構設計亮點整理.....	23
3.6 模組劃分.....	24
第四章 資料模型與資料庫設計（Data Model & Database Design）.....	27
4.1 資料建模原則.....	27
4.2 資料模型概述.....	28
4.3 核心實體與關聯（Core Entities & Relationships）.....	28

4.4 狀態欄位與狀態機.....	32
4.5 資料庫結構與正規化 (Database Schema & Normalization)	33
4.6 資料一致性與交易策略 (Consistency & Transaction Strategy)	33
第五章 Use Case 實作對應.....	34
5.1 UC-02 小卡刊登管理 (Use Case Diagram)	34
5.2 UC-02.1 建立刊登的泳道圖 (Swimlane Diagram)	38
5.3 UC-02.1 建立刊登的循序圖 (Sequence Diagram)	39
5.4 UC-04 交易與提案管理.....	39
5.5 UC-04.1 建立刊登的泳道圖 (Swimlane Diagram)	43
5.6 UC-04.1 建立刊登的循序圖 (Sequence Diagram)	44
第六章 非功能性需求與架構對應.....	44
6.1 效能.....	44
6.2 安全性.....	45
6.3 資料一致性.....	45
6.4 可用性.....	46
6.5 可靠性.....	46
6.6 可維護性.....	47
6.7 可擴充性.....	47
6.8 可測試性.....	48
6.9 操作性.....	48
6.10 系統限制.....	49
6.11 NFR 與設計對應表	49
第七章 風險與改善方向.....	50
第八章 結語與後續擴充建議.....	51

8.1 系統成果總結.....	51
8.2 系統限制.....	53
8.3 後續擴充建議.....	53
8.4 本專案之學習價值.....	55
8.5 結語.....	56

第一章 簡介

1.1 文件目的

本文件為「Exchange Platform」系統之軟體架構設計文件（Software Architecture Design Document, SADD）。本文件旨在自軟體架構與設計的角度，描述系統之整體結構、模組劃分與實作方式，作為開發、測試與維運人員在系統建置過程中的主要參考依據。

本文件著重於闡述系統「如何被建構與運作」，並作為軟體需求規格書（Software Requirements Specification, SRS）的下游文件。SRS 所描述之功能需求與驗收條件，將在本文件中轉化為具體的架構設計、流程規劃、資料模型與模組協作方式。本文件提供一致且可維護的設計基礎，以降低開發風險、提高系統可擴充性並支援後續重構活動。

1.2 文件使用範圍

「Exchange Platform」為一款以明星小卡交換為核心的以物易物平台系統，提供使用者建立刊登、瀏覽他人小卡、提出交換提案、透過站內訊息協商、安排寄送與追蹤物流，並於完成交易時收到系統通知。系統採用分層式單體架構（Layered Monolithic Architecture）與物件導向分析與設計（OOAD / BCE）方式建置。

本軟體架構設計文件之內容涵蓋系統在設計層級之完整架構，包括：

- 架構風格與系統分層說明
- 模組劃分與職責界定
- 系統流程設計與物件間協作（以 Use Case 為基礎）
- 資料模型（Class Diagram）、資料庫結構（ERD / Schema）
- 系統介面與 API 設計之概要
- 非功能性需求（NFR）相關架構考量

本 SADD 的撰寫範圍主要對應目前已完成或設計成熟之核心 Use Case，包括：

- UC-01 身分驗證與使用者管理（Authentication & User Management）
- UC-02 小卡刊登管理（Listing Management）
- UC-03 搜尋與篩選服務（Search & Filter Service）
- UC-04 交易與提案管理（Exchange Proposal & Transaction Management）
- UC-05 即時交流（In-App Messaging / Chat）
- UC-06 出貨整合（Shipping Integration）
- UC-07 貨品追蹤服務（Tracking Service / Manual Update）
- UC-08 Email 通知管理（Email Notification Service）

本文件不包含程式碼層級細節，亦不涵蓋未於目前版本實作之預期功能。

1.3 預期讀者

本文件之讀者包含下列角色：

- 後端、前端與全端工程師：作為開發、重構與維護系統時的架構依據。
- 系統分析與設計人員：用以檢視架構設計是否符合 SRS 與設計原則。
- 測試與品質保證人員 (QA)：協助理解模組行為、流程與邏輯邊界，以規劃測試案例。
- 專案管理者、指導教師與助教：掌握系統架構全貌與主要技術決策，作為審查與追蹤依據。

預期讀者需具備基本軟體工程知識（架構分層、UML、物件導向、後端架構等）。

1.4 參考文件

本文件之內容與架構設計係參考以下資源進行制定：

- (1) Exchange Platform – Software Requirements Specification (SRS)：主要功能與非功能性需求來源。
- (2) Use Case 規格書與相關 UML 圖表
 - Use Case Diagram
 - Sequence Diagram
 - Activity Diagram
 - 初步 Class Diagram
- (3) 課程相關文件
 - 軟體工程設計課程作業規格
 - 指導教授與助教提供之範本、修改建議、設計準則
- (4) 相關標準與技術文件
 - ISO/IEC/IEEE 42010: Software and Systems Engineering – Architecture Description
 - Spring Framework / Spring Boot 官方文件
 - JPA / Hibernate 文件
 - SMTP 與 OAuth2 技術規格

讀者應搭配上述文件以獲得完整系統理解。

第二章 系統架構與運行環境

2.1 系統定位與目標

Exchange Platform 為一個專注於「明星小卡交換」流程的以物易物平台，目標在協助用戶以更透明、便利與安全的方式進行交換提案、協商與寄送。系統提供刊登管理、搜尋與篩選、提案機制、站內訊息交流、出貨安排、物流追蹤與通知機制，並以一致的架構原則確保系統具備可維護性與可擴充性。

平台採用 **Web-based 前後端整合式架構**，後端以 Spring Boot 實作；前端採用 Thymeleaf 與 HTML/CSS，並以 MySQL 作為主要資料庫。系統定位於：

- (1) 一般用戶：提供便捷且清楚的交換流程
- (2) 交易撮合平台：協助交換雙方進行協商與確認
- (3) 資訊服務平台：提供追蹤、通知、交易狀態管理等服務
- (4) 多 Use Case 模組化系統：將交易相關流程分為多個可維護之模組 (Listing、Proposal、Messaging、Shipping 等)

2.2 系統主要功能

本平台核心功能可分為下列模組與服務：

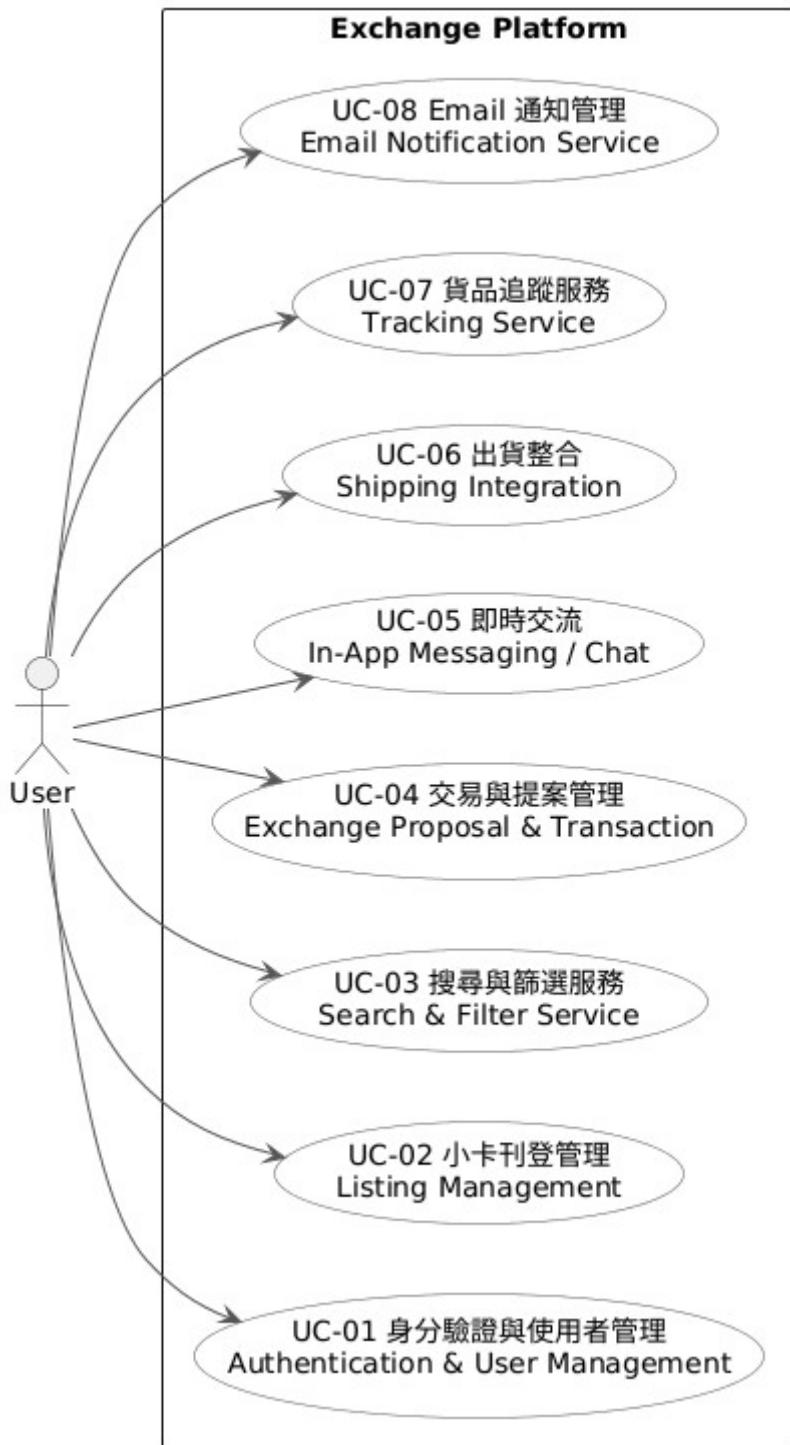


圖 1 系統 Use Case Diagram (Exchange Platform)

(1) 身分驗證與使用者管理 (UC-01)

- Google OAuth2 或平台帳號登入
- 使用者資料建立、基本資料管理
- 權限控制（一般用戶）

(2) 小卡刊登管理 (UC-02)

- 建立與編輯刊登
- 上傳小卡圖片與設定需求標籤
- 刊登狀態管理（上架、下架）

(3) 搜尋與篩選服務 (UC-03)

- 條件搜尋（名稱、類型、稀有度等）
- 多條件篩選
- 分頁檢索與排序

(4) 交易與提案管理 (UC-04)

- 建立提案（選擇交換物品 + 訊息）
- 提案驗證
- 提案方與刊登方之提案狀態管理

(5) 即時交流 (In-App Messaging / Chat) (UC-05)

- 建立私人聊天室
- 提案雙方訊息協商
- 自動建立對應的聊天房間與訊息記錄

(6) 出貨整合 (UC-06)

- 填寫寄件與收件資訊
- 出貨資料交換
- 狀態更新（待寄出 → 已寄出）

(7) 貨品追蹤服務 (UC-07)

- 手動輸入物流編號
- 交易雙方可更新目前物流狀態
- 以狀態機方式管理：待寄出 → 運送中 → 已送達 → 已確認收貨

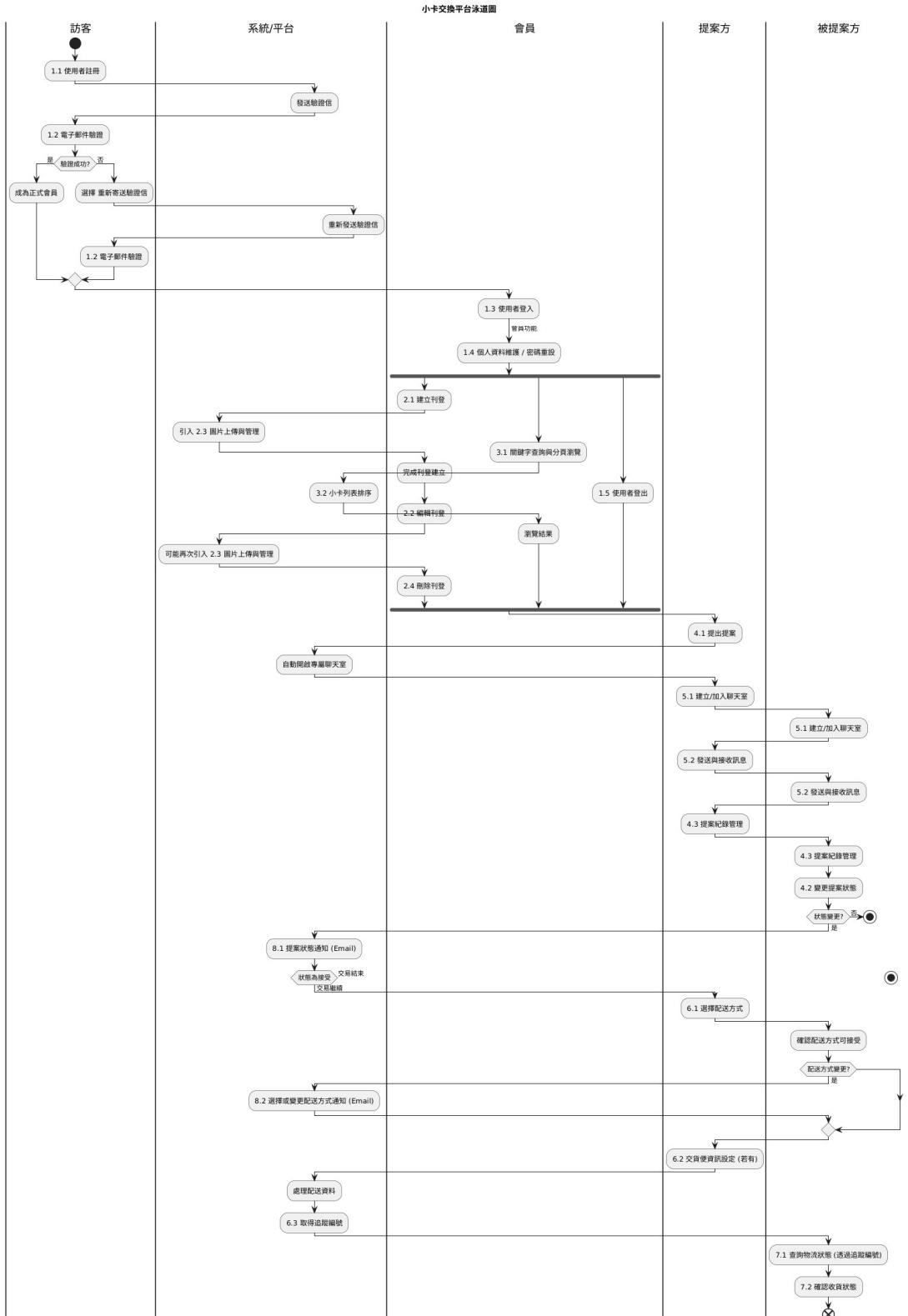
(8) Email 通知管理 (UC-08)

- 寄送提案通知
- 寄送聊天室提醒
- 寄送/取件相關提醒
- SMTP 整合

2.3 系統運作流程概述

以下為 Exchange Platform 的高層級運作流程（非 Use Case 流程）：

- (1) 使用者登入平台。
- (2) 建立個人小卡刊登。
- (3) 搜尋並瀏覽其他使用者的刊登內容。
- (4) 提案方對刊登小卡申請交換提案，系統建立聊天室協助雙方進行協商，並由系統驗證提案內容與小卡狀態。
- (5) 若刊登方同意提案，則交換提案正式被建立，雙方可選擇配送方式。
- (6) 雙方填寫交換寄件資料並各自寄出。
- (7) 系統紀錄物流更新，直到雙方皆確認收貨。
- (8) 系統寄送通知、更新交易狀態並完成交易。
- (9) 系統泳道圖如下：



2.4 系統架構風格

Exchange Platform 採用分層式單體架構 (Layered Monolithic Architecture)，並依照 BCE (Boundary–Control–Entity) 概念進行模組與職責劃分。架構風格特點如

下：

(1) Presentation Layer (Boundary)

- 技術組成：Spring MVC、Thymeleaf Server-Side Rendering (SSR)。
- 主要類別：
 - UI Controllers : UiAuthController、UiListingController、UiProposalController、UiSwapController 等(對應頁面導向與表單提交)。
 - REST API Controller : ListingController、ProposalController、ShipmentController 等(提供 JSON API，支援前端非同步操作或未來前後端分離擴充)。
- 職責：
 - 接收來自瀏覽器的 HTTP 請求與表單資料，進行基本參數綁定與驗證錯誤顯示。
 - 呼叫對應的 Service 層方法，委派商業邏輯處理。
 - 回傳 HTML 頁面 (Thymeleaf Template) 或 JSON 回應，負責基本導頁 (redirect / forward) 與錯誤訊息展示。
 - 嚴格避免直接操作 Repository，維持「Controller 薄、Service 厚」的分層風格。

(2) Application / Domain Layer (Control)

- 技術組成：Spring @Service, Spring Transaction Management (@Transactional)。
- 主要類別：AuthService、ListingService、ProposalService、SwapService、ShipmentService、TrackingService、ChatService、EmailNotificationService 等。
- 職責：
 - 以 Use Case 為單位實作商業邏輯，作為 Boundary 與 Entity/Repository 之間的主要協調者。
 - 定義並維護重要 Domain Invariants (不變式)，對應 B-Machine 規格中的條件，例如：
 - Listing 建立 / 修改時的欄位與狀態規則 (例如：ACTIVE、IN_NEGOTIATION、COMPLETED 等)。
 - Proposal 多卡提案 + Listing 鎖定機制：在建立提案時，檢查目標 Listing 與提案方所有 Offered Listings 均為 ACTIVE 且未被其他

Proposal 鎖定，並透過 `locked_by_proposal_id` 欄位實作 row-level locking，避免同一張小卡被多個提案重複承諾。

- Swap 成立時，同步更新本次交換所涉及之雙方所有 Listing 狀態，從「可交換」轉為「交易中 / 已完成」，與 Proposal / Swap 的狀態機制一致。
- Shipment 與 ShipmentEvent 的事件流模型：依據 Swap 建立雙方的出貨紀錄，並透過事件歷程管理物流狀態。

➤ 管理「提案 → 聊天室 → 通知」的一條龍流程：

- 提案建立成功後，由 ChatService 建立對應 ChatRoom，寫入系統訊息。
- 同時由 EmailNotificationService 建立 EmailNotification，寄送 Email 通知被提案方。

➤ 落實「無金流平台」的設計：

- 在表單驗證與服務邏輯中，避免價格欄位、金額計算與外部金流整合。
- 可在文字內容檢查中偵測明顯價格或聯絡資訊(例如電話、Line ID)並予以提示或限制。

(3) Persistence Layer (Entity / Repository / DB)

● 技術組成：Spring Data JPA、Hibernate、MySQL 8。

● 主要類別：

- JPA Entities : User、Listing、Proposal、ProposalItem、Swap、Shipment、ShipmentEvent、ChatRoom、ChatMessage、EmailNotification 等，並包含針對明星小卡領域客製化的欄位，如：`idol_group`、`member_name`、`album`、`era`、`version`、`card_code`、`is_official`、`condition` (S/A/B/C)、`has_protection` 等。
- Repository : UserRepository、ListingRepository、ProposalRepository、SwapRepository、ShipmentRepository、ShipmentEventRepository 等。

● 職責：

- 封裝資料存取細節，提供 CRUD、條件查詢、排序與分頁等操作。
- 配合 Service 層的 `@Transactional` 設定，確保多表更新與狀態轉換在單一交易中完成，維護 domain 一致性。

➤ 與 Domain Invariants 對應：例如 Listing 狀態、Proposal 狀態、Swap 狀態與 Shipment 狀態的欄位組合，皆在資料模型中一致呈現。

(4) Infra / Integration Layer

- Email(SMTP)：由 EmailNotificationService 封裝與 SMTP Server 的互動，提供寄送註冊驗證信、提案通知、聊天提醒與出貨提醒等功能。
- Google OAuth2 (第三方登入)：透過 Spring Security OAuth2 Client 與 Google Provider 整合，使用者可使用 Google 帳號登入，簡化註冊流程並降低密碼管理風險。
- 檔案儲存：小卡照片與交易證據圖片上傳至 Web Server 本機 /uploads 目錄，由 Service 層控管檔案命名與對應路徑，並於資料庫中記錄檔案資訊。
- 7-11 交貨便整合（爬蟲輔助查詢）：透過後端爬蟲模組存取 7-11 交貨便公開查詢頁，使用者輸入交貨便貨物編號後，系統可協助取得物流狀態資訊，並搭配 ShipmentEvent 事件流模型紀錄於系統中。

(5) 架構設計亮點

- 圍繞「明星小卡交換」客製化之 Domain Model，而非一般汎用二手平台。
- 使用 Proposal + ProposallItem + Listing Locking 的組合，處理「多卡換一張」及競價/撞單場景，降低併發爭議。
- 以 Swap 為交易核心，搭配雙向 Shipment 與 ShipmentEvent 實作「事件流式」物流追蹤，兼顧課程實作範圍與實務可追蹤性。
- 從畫面設計、資料模型到 Service 邏輯多層次實作「無金流平台」，避免價格與金流責任。
- 將 Proposal 視為溝通起點，與 ChatRoom、EmailNotification 串成一條完整協商與通知鍊。

2.5 系統外部整合與依賴關係

Exchange Platform 與多個外部系統進行整合，以支援身分驗證、郵件通知與物流查詢等功能；同時在執行上亦仰賴資料庫、檔案儲存與瀏覽器環境等基礎元件。本節說明主要整合與依賴之系統。

(1) Google OAuth2 登入整合

系統整合 **Google OAuth2** 作為第三方登入機制，使用者可透過 Google 帳號快速登入平台。整合方式如下：

- 使用 Spring Security OAuth2 Client 設定 Google Provider 相關參數 (Client Id / Client Secret 、 Redirect URI 等)。
- 使用者於登入頁選擇「以 Google 登入」後，瀏覽器被導向 Google 授權頁面完成授權。

- 授權完成後，Google 透過 Redirect 將使用者導回本系統，後端接收 Authorization Code 並交換 Access Token。
- 系統依據回傳之使用者資訊（如 email）建立或更新對應的 User 資料，並建立應用程式內的登入 Session。

此機制簡化註冊流程，並減少本系統直接管理密碼的風險。

(2) SMTP Email Server 整合

系統透過 SMTP Email Server 寄送各類通知郵件，包括：

- 註冊驗證信
- 提案建立通知
- 提案回覆／接受／拒絕通知
- 聊天訊息提醒
- 交易與出貨相關通知

EmailNotificationService 封裝所有郵件寄送邏輯，負責產生信件內容（收件者、標題、信件模板套用）並透過 SMTP Server 寄出。此設計讓應用程式其他模組不需直接操作 SMTP，未來若更換郵件服務供應者，只需調整基礎設施層（Infra Layer）設定即可。

(3) 7-11 交貨便物流查詢整合

系統支援 7-11 交貨便之物流狀態查詢，設計重點如下：

- 使用者在 Swap / Shipment 相關畫面中輸入交貨便貨物編號。
- 系統產生對應之 7-11 交貨便查詢連結，並可透過後端爬蟲模組對公開查詢頁進行查詢，以取得當前物流狀態。
- 查詢結果可配合 ShipmentEvent 模型，以事件形式紀錄於系統中（例如：「已交寄」、「配送中」、「門市已到貨」、「已取件」），並更新 Shipment 的 last_status 欄位。

本設計刻意不直接整合正式物流 API，而是採用「手動事件 + 外部查詢頁 + 爬蟲輔助」的方式，在課程實作範圍內兼顧實作可行性、可追蹤性與資料透明度，同時保留將來改接正式 API 的擴充空間。

(4) 其他關鍵依賴元件

除了上述外部服務外，本系統尚仰賴下列關鍵依賴：

- MySQL 8：作為平台核心業務資料的儲存與一致性來源，所有交換、提案、出貨與追蹤狀態皆以資料庫為準。
- 檔案儲存（本機 /uploads 目錄）：用於存放 Listing 上架時的小卡圖片以及交換證據檔案，實際檔案路徑由 Service 層管理並記錄於資料庫欄位。

- Web Browser(瀏覽器): 作為主要使用介面環境，前端採標準 HTML/CSS / JavaScript 與 Thymeleaf SSR，不依賴特定前端框架，以提高相容性並降低部署複雜度。

透過上述依賴與整合設計，Exchange Platform 在不引入過度複雜基礎建設的前提下，仍能提供完整的登入、通知與物流支援能力。

2.6 部屬與運行環境

本系統採用三層式(Client–Server–Database)架構，前端瀏覽器作為 Client，Spring Boot Monolith 作為 Application Server，MySQL 作為資料庫伺服器，並輔以本機檔案儲存與外部 Email / OAuth2 服務。系統架構如圖 1 所示。

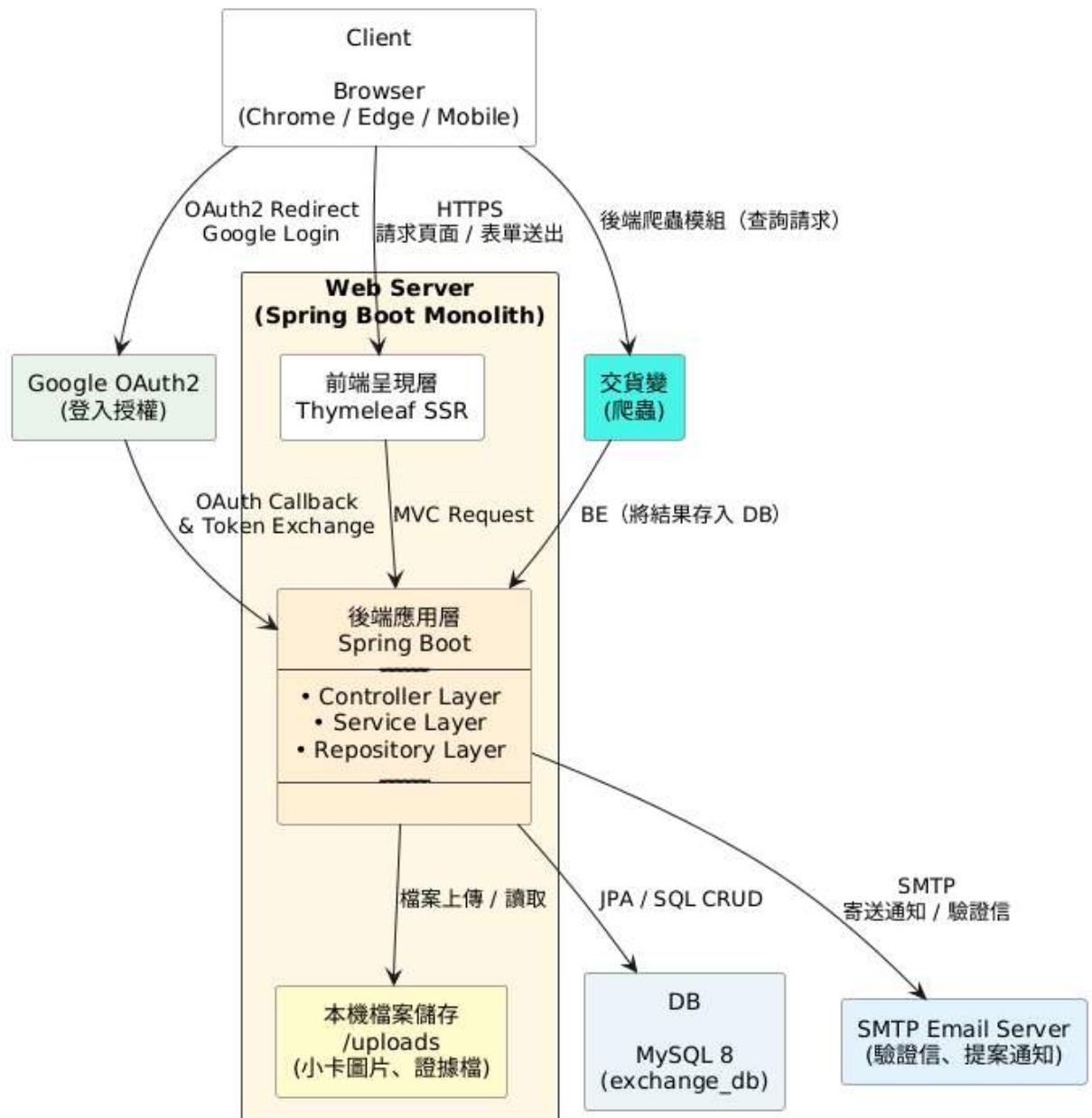


圖 2 系統架構圖 (Exchange Platform)

系統架構圖如圖一所示，主要元件與互動關係說明如下：

(1)Client (使用者端)

- 使用者透過桌機或手機瀏覽器（Chrome / Edge / Safari / 行動版瀏覽器）存取系統。
- 所有畫面由後端以 Thymeleaf Server-Side Rendering (SSR) 產生 HTML 頁面，瀏覽器負責顯示頁面、上傳圖片、送出表單與發送 Ajax 請求。
- 部分功能（例如建立 Listing、送出 Proposal）除了傳統表單送出外，亦提供 REST API 端點，方便未來改為 SPA 或行動 App。
- Client 與伺服器之間以 HTTP/HTTPS Request/Response 溝通，確保資料在網路傳輸過程中的機密性與完整性（正式環境預期採用 HTTPS）。

(2)Application Server (Spring Boot Monolith)

應用伺服器為本系統的核心容器，以 Spring Boot 可執行 Jar 方式部署於單一主機。內部依職責區分為多個邏輯子層：

- Controller Layer (Boundary)
 - UI Controller：處理 Thymeleaf 頁面相關的 GET/POST 請求（如 UiAuthController 、 UiListingController 、 UiProposalController 、 UiSwapController ）。
 - REST API Controller：處理 JSON API 請求（如 ListingController 、 ProposalController 、 ShipmentController ），提供前端非同步操作與未來前後端分離擴充的基礎。
 - 負責路由、輸入驗證與回傳 View 或 JSON ，不直接操作資料庫。
- Service Layer (Control)
 - 包含 AuthService 、 ListingService 、 ProposalService 、 SwapService 、 ShipmentService 、 TrackingService 、 ChatService 、 EmailNotificationService 等核心元件。
 - 實作交換提案、 Listing 管理、提案鎖定、多卡交換、出貨與收貨確認等業務邏輯，並透過 @Transactional 管理交易邊界。
 - 負責維護 Domain Invariants （例如 Listing 鎖定邏輯、 Proposal/Swap/Shipment 狀態流轉），確保系統狀態與 B-Machine 規格一致。
- Repository Layer (Entity / DB Mapping)
 - 使用 Spring Data JPA 與 Hibernate 對應資料表，提供 UserRepository 、 ListingRepository 、 ProposalRepository 、 SwapRepository 、

- ShipmentRepository、ShipmentEventRepository 等介面。
- 負責執行 CRUD 與查詢，所有資料存取均在 Service 的交易範圍內完成，以維持資料一致性。
 - 檔案儲存模組（Local Disk Storage）
 - 系統將上架圖片、交易證據檔案等存放於伺服器本機 /uploads 目錄。
 - 應用程式透過 Service 層封裝檔案上傳、命名與對應 URL 產生，前端僅透過公開 URL 存取檔案，避免直接暴露實體路徑。
 - 外部整合封裝
 - Email 通知整合：由 EmailNotificationService 統一呼叫 SMTP Email Server 寄送帳號驗證信、提案通知、出貨提醒與收貨提醒。
 - OAuth2 登入整合：透過 Spring Security OAuth2 Client 與 Google OAuth2 Provider 溝通，提供第三方登入功能。
 - 交貨便查詢整合：後端爬蟲模組負責對 7-11 公開查詢頁發出請求並解析結果，將物流狀態寫入 ShipmentEvent 與相關欄位。

(3)Database (MySQL)

- 採用 MySQL 8 作為關聯式資料庫，儲存系統所有業務資料，包括 users、listings、proposals、proposal_items、swaps、shipments、shipment_events、chat_rooms、chat_messages、email_notifications 等資料表。
- Application Server 透過 Spring Data JPA / Hibernate 與 MySQL 連線，並由 Service 層以交易（Transaction）為單位進行操作，以符合 B-Machine 所定義之狀態不變條件。
- 資料庫可與應用伺服器部署於同一台主機，或視未來需求獨立部署至專用 DB 主機或雲端 RDS 服務。

(4)外部服務容器

- SMTP Email Server：作為獨立外部服務容器，負責實際發送 Email。應用程式僅透過設定好的 SMTP 端點與認證資料進行連線。開發／測試階段可使用如 Mailtrap 等測試 SMTP 服務。
- Google OAuth2 Provider：作為外部身分認證提供者，負責處理使用者以 Google 帳號登入之授權流程，Application Server 則負責將授權結果轉換為本系統的 User 資料與 Session。

(5)運行環境架構 (Runtime Architecture)

- 開發與 Demo 環境
 - Java 17 以上版本
 - Spring Boot 3.x
 - 本機 MySQL 資料庫
 - SMTP 測試服務（例如 Mailtrap 或自架 Mail Server）

- 檔案存於開發機的 /uploads 目錄
- 目前部署方式
 - 以單一 Spring Boot Jar 部署於同學之個人電腦。
 - MySQL 可與應用程式部署於同一主機，以簡化課堂 demo 與除錯流程。
 - 尚未採用 Docker 或 Kubernetes，刻意避免引入過多 DevOps 複雜度。
- 未來擴充方向
 - 將 Application Server 與 Database 分離主機，減少體機負載。
 - 導入 Docker 容器與 CI/CD(例如 GitHub Actions)以自動化部署。
 - 若實際使用者數物加，可：在 Nginx 後方部署多個 Spring Boot 節點進行水平擴充；導入外部物件儲存(如 S3)儲存圖片；加入基本監控與告警(Prometheus / Grafana)。

第三章 系統架構設計

本章在第 2 章「系統架構與運行環境」的基礎上，進一步從不同架構視圖（Architecture Views）的角度，說明 Exchange Platform 的分層設計、模組劃分與依賴關係。本章不再重複描述部署環境，而是聚焦於「程式碼層級的架構設計」。

3.1 架構視圖總覽

本系統採用 Layered Monolithic Architecture，並落實 BCE (Boundary - Control - Entity) 概念。為了讓架構清晰可溝通，本章採用以下幾種視圖呈現系統：

- 分層類別視圖 (Layered Architecture Class View)：說明 Controller / Service / Repository / Entity / DTO 等類別如何對應至各架構層。
- 模組依賴視圖 (Modules Dependency Overview)：以「模組」為單位呈現 Authentication、Listing、Proposal、Swap、Shipment、Chat 等模組之間的依賴關係。
- 模組內部結構視圖 (Module Internal Structure)：以某幾個代表性模組（如 Listing Module、Proposal Module）為例，展示其 Controller - Service - Repository 之間的責任分工。
- 橫切關注點 (Cross-cutting Concerns)：說明交易管理 (Transaction)、驗證 (Validation)、錯誤處理與日誌等橫切關注如何在架構中實作。

這些視圖共同構成本系統的整體架構描述，有助於開發、維護與課堂評分時快

速理解系統設計。

3.2 分層類別視圖 (Layered Architecture Class View)

本視圖將 Exchange Platform 的主要類別依 BCE 與分層架構原則加以歸類，可視為第 2 章所述「三層式架構」在程式碼層級的具體映射。

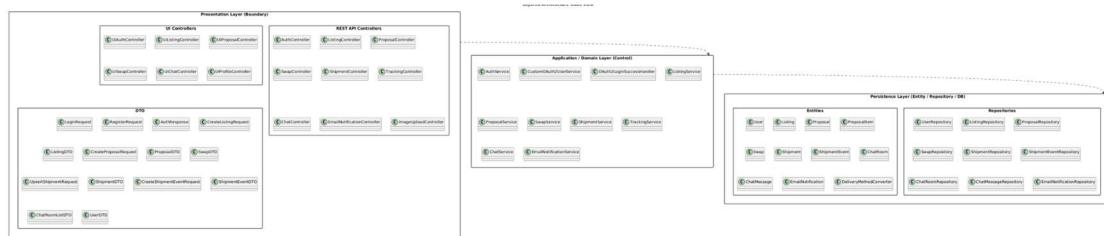


圖 2 Layered Architecture Class View (分層架構類別視圖)

本圖將 Exchange Platform 之主要元件依 BCE 與分層架構原則加以歸類，分為 Presentation (Controller)、Application/Domain (Service)、Persistence (Repository + Entity) 三層。上層僅能依賴下層：Controller 呼叫 Service 執行 Use Case ， Service 再透過 Repository 操作 Entity 與資料庫；任何 Controller 不得直接呼叫 Repository ， Entity 亦不包含商業流程邏輯。此圖呈現出程式碼目錄結構 (controller 、 service 、 repository 、 entity) 與架構理念的一致性，有助於後續維護與重構。

整體對應關係如下：

(1) Presentation Layer (Boundary)

- 對應：controller package (UI Controllers + REST API Controllers)
- 主要內容：

➤ UI Controllers (Thymeleaf)

UiAuthController 、 UiListingController 、 UiProposalController 、
UiSwapController 、 UiChatController 、 UiProfileController
→ 處理頁面導向、表單送出與 View Model 準備。

➤ REST API Controllers

AuthController 、 ListingController 、 ProposalController 、
SwapController 、 ShipmentController 、 TrackingController 、
ChatController 、 EmailNotificationController 、
ImageUploadController 等
→ 提供 JSON API ，作為未來前後端分離或 App 的介面基礎。

➤ DTO (Data Transfer Objects)

LoginRequest 、 RegisterRequest 、 AuthResponse 、
CreateListingRequest 、 ListingDTO 、 CreateProposalRequest 、
ProposalDTO 、 SwapDTO 、 UpsertShipmentRequest 、

ShipmentDTO、CreateShipmentEventRequest、
ShipmentEventDTO、ChatRoomListDTO、UserDTO 等
→ 將 HTTP 請求與回應格式與內部 Domain Model 解耦。

(2) Application / Domain Layer (Control)

- 對應：service package（各種 Service 類別與 OAuth2 登入流程）
 - 主要 Service 類別：
 - 身分驗證：AuthService、CustomOAuth2UserService、OAuth2LoginSuccessHandler
 - 刊登管理：ListingService
 - 提案管理：ProposalService
 - 交換管理：SwapService
 - 出貨與追蹤：ShipmentService、TrackingService
 - 即時交流：ChatService
 - 通知管理：EmailNotificationService
 - 主要職責：
 - 以 Use Case 為單位實作商業邏輯。
 - 維護系統關鍵不變條件 (Domain Invariants)。
 - 串接「提案 → 聊天室 → Email 通知」完整流程。
- ## (3) Persistence Layer (Entity / Repository / DB)
- 對應：entity、repository packages
 - 主要 Entities：
 - User、Listing、Proposal、ProposalItem、Swap
 - Shipment、ShipmentEvent
 - ChatRoom、ChatMessage
 - EmailNotification
 - 以及 DeliveryMethodConverter 等 JPA Converter。
 - 主要 Repositories：
 - UserRepository、ListingRepository、ProposalRepository、SwapRepository
 - ShipmentRepository、ShipmentEventRepository
 - ChatRoomRepository、ChatMessageRepository
 - EmailNotificationRepository
 - 主要職責：
 - 封裝資料存取細節 (CRUD、條件查詢、分頁與排序)。
 - 配合 Service 層的 @Transactional，確保多表更新在單一交易中完成。

此架構採用嚴格分層原則：

- Controller 只呼叫 Service，不直接操作 Repository
- Service 負責商業邏輯與交易（@Transactional），不處理畫面邏輯
- Repository 專注於資料存取，不含商業邏輯
- Entity 為資料模型本身，不主動呼叫 Service
- DTO 作為 API 邊界物件，避免前端直接依賴 Entity 結構

3.3 模組依賴關係 (Modules Dependency Overview)

在模組層級，Exchange Platform 遵守單向依賴與高內聚、低耦合原則。模組之間的主要依賴關係如下：

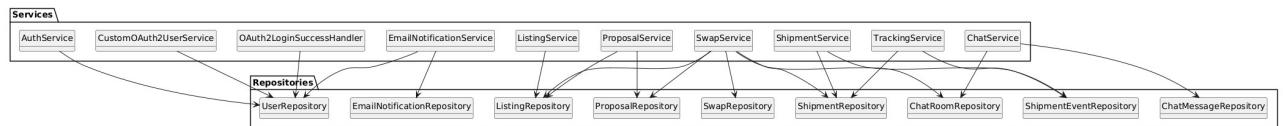


圖 3 Modules Dependency Overview (模組依賴關係)

- Authentication Module：主要被其他模組「查詢目前登入的 User」，不依賴其他業務模組。
- Listing Module：被 Search & Filter、Proposal、Swap 等模組依賴。
- Proposal Module：依賴 Listing Module (需要讀寫 Listing 狀態) 與 Chat / Notification Module (提案成功後需要建立聊天室與寄信)。
- Swap Module：依賴 Proposal Module (由被接受的 Proposal 產生 Swap)，並與 Shipment 模組協作 (依 Swap 建立 Shipment)。
- Shipment & Tracking Module：依賴 Swap 模組 (以 Swap 為出貨的來源)，並與 7-11 交貨便外部網站互動。
- Chat Module：依賴 Proposal Module (依 Proposal 建立聊天室)，並間接與 Notification Module 協作。
- Notification Module：由 Proposal / Swap / Shipment / Chat 等模組觸發，但本身不反向依賴這些模組。

此設計確保了「資料生命週期」清楚：Listing → Proposal → Swap → Shipment / Chat / Notification，每一階段都建立在前一階段的結果之上，不會逆向依賴或產生循環。

3.4 橫切關注點 (Cross-cutting Concerns)

(1) 交易與資料一致性

- 核心 Service (ProposalService、SwapService、ShipmentService、TrackingService) 使用 @Transactional 控制交易邊界。

- 重要場景：
 - 建立 Proposal 時，同時寫入 Proposal + ProposalItem + 更新多筆 Listing 的 locked_by_proposal_id。
 - 建立 Swap 時，同步更新所有相關 Listing 狀態。
 - 新增 ShipmentEvent 時，更新 Shipment 的 last_status。
- 這些操作必須成功或全部回滾，以維護 B-Machine Invariants。

(2) 驗證與防呆

- Controller 層：基本欄位驗證（例如：必填欄位、格式檢查）。
- Service 層：商業邏輯驗證：
 - 提案至少選一張小卡。
 - 目標 Listing 與 Offered Listings 均為 ACTIVE 且未被鎖定。
 - 同一提案方對同一張小卡不得有重複 PENDING 提案。
- 搭配第 7 章之錯誤處理章節說明整體防呆策略

(3) 安全性

- 使用 Spring Security 管控登入狀態。
- Google OAuth2 整合由 Security Config 統一管理，與業務模組分離。

3.5 架構設計亮點整理

綜合前述視圖，本系統在架構上的幾項亮點如下：

- (1) 分層清楚、對應目錄結構：
 - controller / service / repository / entity / dto 嚴格分工，閱讀成本低。
- (2) BCE 貫穿設計與命名
 - Boundary (UI/REST Controller)、Control (Service)、Entity (JPA Entities) 對應清楚。
- (3) 多卡交換 + 鎖定機制集中於 Service：
 - ProposalService 與 SwapService 中實作關鍵商業規則，避免錯誤散落於多處。
- (4) Shipment + ShipmentEvent 事件流模型：
 - 支援雙向寄件與歷程追蹤，而非單一狀態欄位，便於除錯與實務對照。
- (5) Proposal → ChatRoom → Notification 一條龍設計：
 - 提案建立即建立聊天室與 Email 通知，讓使用者的溝通與提醒都在同一脈絡中完成。
- (6) 無金流平台的架構落實：
 - 從資料模型、Service 邏輯到畫面，不設計任何金流相關欄位與外部金流 API，明確界定系統責任邊界。

3.6 模組劃分

整個系統依照 Use Case 與 Domain 分成多個模組，每個模組包含 Controller / Service / Repository / Entity：

(1) Authentication & User Module — 身分驗證與使用者管理：

職責：

- Google OAuth2 / Local Login
- 建立與管理 User
- Session 與授權 (Authorization)
- 基本資料維護

核心類別：

- UiAuthController, AuthController
- AuthService
- UserRepository
- User Entity

(2) Listing Module — 小卡刊登管理

職責：

- 建立、編輯、下架 Listing
- 上傳與管理 Listing 圖片
- 列表顯示、詳細頁
- 屬性與小卡領域屬性 (idol, era, version...)

核心類別：

- UiListingController, ListingController
- ListingService
- ListingRepository
- Listing Entity

(3) Search & Filter Module — 搜尋與篩選服務

職責：

- 多欄位搜尋 (idol_group, member_name, card_code...)
- 多條件篩選 (condition, version, has_protection)
- 分頁 / 排序

核心類別：

- ListingRepository (含自訂查詢)
- ListingService (封裝搜尋流程)

此模組屬於 Listing 子模組，但因功能龐大，於架構文件中獨立呈現。

(4) Proposal Module — 交易提案管理

職責：

- 建立提案
- 多卡交換 (ProposalItem + 多張 Offered Listings)
- Listing 鎖定 (避免撞單)
- 提案流程 (等待回覆、接受、拒絕)

核心類別：

- UiProposalController, ProposalController
- ProposalService
- ProposalRepository, ProposalItemRepository
- Proposal, ProposalItem Entities

架構重點：

- 使用 locked_by_proposal_id 實現 Listing row-level locking
- 服務層中維護提案不變式 (Invariants)

(5) Chat Module — 即時交流 (站內訊息)

職責：

- 提案接受後，自動建立 ChatRoom
- 訊息往返、系統訊息 (如「提案已接受」)
- 聊天內容儲存與查詢

核心類別：

- ChatService
- ChatRoomRepository, ChatMessageRepository
- ChatRoom, ChatMessage Entities

架構亮點：

- 聊天室建立完全由 ProposalService 觸發
- 「提案 → 聊天室 → Email 通知」三者完整串接

(6) Swap Module — 交換成立管理

職責：

- 提案接受 → 建立 Swap
- 維護交換中兩張（或多張）小卡的狀態
- 驗證狀態一致性（避免重複交易）

核心類別：

- UiSwapController
- SwapService
- SwapRepository
- Swap Entity

架構亮點：

- Swap 作為交易核心，管理完整交換生命週期
- Service 層維護 Listing 與 Proposal 與 Swap 之間不變式

(7) Shipment Module — 出貨管理

職責：

- 填寫寄件 / 收件資訊
- 雙向出貨資料（雙方都寄）
- 更新寄出狀態

核心類別：

- ShipmentController
- ShipmentService
- ShipmentRepository
- Shipment Entity

(8) Tracking Module — 物流追蹤事件流

職責：

- 管理物流事件（ShipmentEvent）
- 手動新增事件
- 爬蟲更新事件（7-11 交貨便）
- 狀態機（待寄出 → 運送中 → 到貨 → 已取件）

核心類別：

- TrackingService
- ShipmentEventRepository
- ShipmentEvent Entity

(9) Notification Module — 通知管理

職責：

- 提案通知
- 聊天訊息提醒
- 出貨 / 收貨提醒
- 使用 SMTP 發信

核心類別：

- EmailNotificationService
- EmailNotificationRepository
- EmailNotification Entity

第四章 資料模型與資料庫設計（Data Model & Database Design）

本章說明 Exchange Platform 的資料模型、主要資料表、欄位定義、關聯性與不變條件 (Invariants)。資料模型採用 JPA Entity + MySQL 8 實作，並配合第 3 章所述的 Domain Modules (Listing、Proposal、Swap、Shipment、Chat、Notification...) 進行邏輯切割。

4.1 資料建模原則

本系統資料模型設計的主要原則如下：

(1) 以「明星小卡交換」為核心領域模型

- 非一般二手拍賣，而是針對 idol 小卡情境設計欄位 (idol_group、member_name、album、era、version、card_code...)。
- 支援「多卡換一張」與「拼套」使用情境。

(2) 狀態驅動 (State-driven) 設計

- Listing、Proposal、Swap、Shipment、EmailNotification 等皆具狀態欄位，對應 B-Machine Invariants。
- 狀態轉換僅透過 Service 邏輯執行。

(3) 事件流 (Event Sourcing Lite) 概念

- 物流相關資料以 Shipment+ShipmentEvent 表示，而非單一狀態欄位。
- 便於追蹤「發生過什麼事」而不只是「現在是什麼狀態」。

(4) 無金流設計

- 資料模型完全不包含價格、金額與金流交易紀錄，呼應系統「以物易物、不涉金流」的定位。

4.2 資料模型概述

Exchange Platform 的資料模型圍繞「明星小卡交換」設計，並非一般二手拍賣平台。其核心概念為：

- Listing (刊登)：一張可交換的明星小卡。
- Proposal (提案)：交換意願，其中可包含多張小卡 (ProposalItem)。
- Swap (交換)：提案被接受後形成的正式交易記錄。
- Shipment (出貨)：雙方各一筆，記錄寄件與物流資訊。
- ShipmentEvent (物流事件)：事件流式物流記錄。
- ChatRoom / ChatMessage (聊天)：提案對應一個聊天房間。
- EmailNotification (通知)：所有 Email 的發送紀錄。

所有資料表皆以 **User**、**Listing**、**Proposal**、**Swap** 為主軸進行關聯。

4.3 核心實體與關聯 (Core Entities & Relationships)

以下為 Entity Class Diagram，以下為主要實體與關聯概念說明：

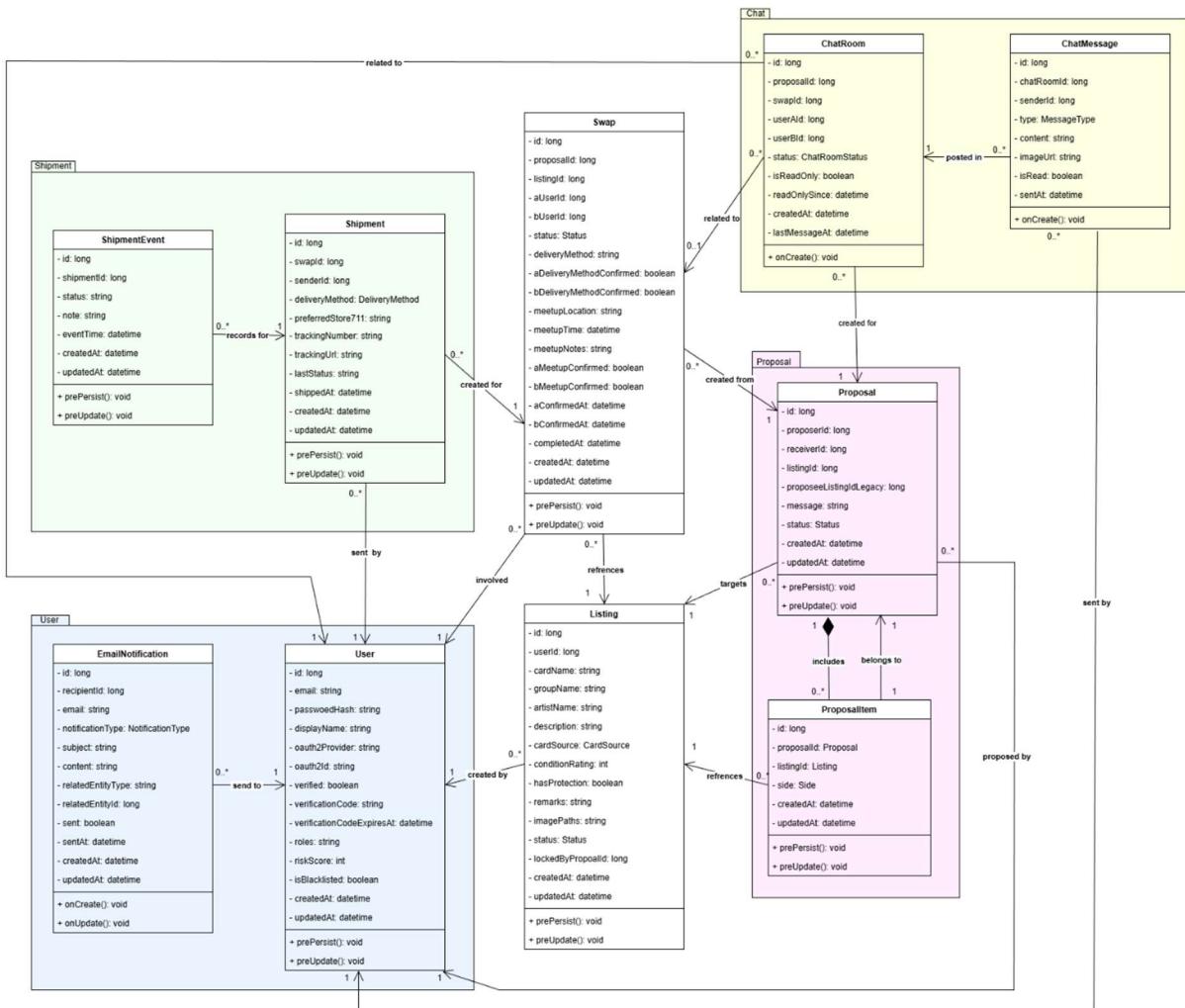


圖 5 Entity Class Diagram

(1) User

- 用途：紀錄平台使用者的基本資訊與登入方式。
- 重要欄位：
 - id、email、password（本地帳號）、displayName
 - oauth2Provider、oauth2Id（Google 登入使用）
- 關聯：
 - 1:N → Listing（使用者可建立多個刊登）
 - 1:N → Proposal（同時作為 proposer 與 owner）
 - 1:N → ChatRoom / ChatMessage / Shipment / Swap 等

(2) Listing

- 用途：代表一張可交換的明星小卡刊登。
- 重要欄位：

- 基本資訊：title、description、imagePath
- 小卡領域欄位：idolGroup、memberName、album、era、version、cardCode
- 狀態欄位：status（例如：ACTIVE、IN_NEGOTIATION、COMPLETED）
- 保護與條件：condition（S/A/B/C）、hasProtection（是否有卡套、上硬卡）
- 鎖定欄位：lockedByProposalId（實作多卡鎖定機制）
- 關聯：
 - N:1 → User (owner)
 - 1:N → ProposalItem（作為被拿來交換的一方）
 - 1:1 → 在 Proposal 中可作為目標 Listing (target)

(3) Proposal & ProposalItem

- Proposal (主實體)：
 - 用途：代表一次交換提案。
 - 重要欄位：
 - proposerId、ownerId
 - targetListingId（被提案方的小卡）
 - status (PENDING、ACCEPTED、DECLINED、WITHDRAWN)
 - message (提案訊息)
- ProposalItem (子實體)：
 - 用途：提案方「拿出來交換」的多張小卡。
 - 關聯：
 - N:1 → Proposal
 - N:1 → Listing (offeredListing)
 - 資料存取：
 - 透過 Proposal 的 @OneToMany(cascade = ALL) 關聯自動儲存，無獨立 Repository。

(4) Swap

- 用途：代表「已被接受」的提案後所成立的一筆交換交易。

- 重要欄位：
 - id、proposalId、proposerId、ownerId
 - status (IN_PROGRESS、COMPLETED、CANCELLED 等)
- 關聯：
 - 1:1 → Proposal
 - 1:N → Shipment (通常各有一筆 Shipment : A → B、B → A)
 - 與涉及的 Listings 形成邏輯上的關聯 (由 Service 將相關 Listing 狀態同步更新)。

(5) Shipment & ShipmentEvent

- Shipment :
 - 用途：代表一次實際寄件行為（寄件方 → 收件方）。
 - 重要欄位：
 - swapId (一定綁定於已成立的 Swap)
 - senderId、receiverId
 - deliveryMethod (交貨便 / 面交)
 - trackingNumber (交貨便貨物編號)
 - lastStatus (最後一個事件狀態，用於快取顯示)
- ShipmentEvent :
 - 用途：紀錄單筆 Shipment 的狀態歷程。
 - 重要欄位：
 - shipmentId
 - eventType (如：CREATED、DROPPED_AT_STORE、IN_TRANSIT、ARRIVED_AT_STORE、PICKED_UP)
 - eventTime、note
- 關聯：
 - 1:N → 由 Shipment 到 ShipmentEvent (事件流)

(6) ChatRoom & ChatMessage

- ChatRoom :
 - 用途：代表提案或交易中雙方的聊天室。
 - 重要欄位：
 - proposalId 或 swapId (與交易脈絡連動)

- title、createdAt
- ChatMessage：
 - 用途：存放聊天訊息與系統訊息。
 - 重要欄位：
 - chatRoomId、senderId
 - content、messageType (USER_MESSAGE / SYSTEM_MESSAGE)
- 關聯：
 - 1:N → ChatRoom 到 ChatMessage

(7) EmailNotification

- 用途：紀錄系統寄送 Email 的請求與狀態。
- 重要欄位：
 - recipientEmail、subject、templateName
 - status (PENDING、SENT、FAILED)
 - relatedProposalId、relatedSwapId 等（選填，用於追蹤來源事件）

4.4 狀態欄位與狀態機

本系統將多個重要實體設計為具狀態機制的 **Aggregate Root**，相關狀態如下：

(1) Listing Status

- ACTIVE → 可被提案
- IN_NEGOTIATION → 被鎖定，不可再提案
- COMPLETED → 已交換，不可再上架
- locked_by_proposal_id 只能為 1 或 null

(2) Proposal Status

- PENDING 才能被 ACCEPTED 或 DECLINED
- ACCEPTED → 必須建立 Swap
- DECLINED / WITHDRAWN → 不得建立 Swap
- Proposal Items 不得為空（至少 1 張換 1 張）

(3) Swap Status

- 一筆 Proposal 最多只有一筆 Swap
- Swap 成立 → 所有 Listing 狀態必須更新成 IN_NEGOTIATION /

COMPLETED

- Swap COMPLETED → 所有 Listing 終止生命週期

(4) Shipment / Event Status

- Shipment 必須綁定 Swap
- Swap 必須有 2 筆 Shipment
- ShipmentEvent 必須按時間排序
- last_status = 最新事件之狀態

4.5 資料庫結構與正規化 (Database Schema & Normalization)

本系統使用 MySQL 8 作為關聯式資料庫，依照第三範式 (3NF) 原則設計 Schema，避免重複資料與更新異常。

- 每個主要 Entity 對應一張資料表：users、listings、proposals、proposal_items、swaps、shipments、shipment_events、chat_rooms、chat_messages、email_notifications 等。
- 外鍵關聯：
 - listings.user_id → users.id
 - proposals.proposer_id / proposals.owner_id → users.id
 - proposals.target_listing_id → listings.id
 - proposal_items.proposal_id → proposals.id
 - proposal_items.listing_id → listings.id
 - swaps.proposal_id → proposals.id
 - shipments.swap_id → swaps.id
 - shipment_events.shipment_id → shipments.id
 - chat_rooms.proposal_id → proposals.id
 - chat_messages.chat_room_id → chat_rooms.id
- 索引設計 (Indexing)：常用查詢條件設計索引，例如：
 - listings(status, idol_group, member_name, album, version)
 - proposals(proposer_id, status)、proposals(owner_id, status)
 - shipments(swap_id)、shipment_events(shipment_id, event_time)
 - chat_messages(chat_room_id, created_at)
 - 能夠兼顧搜尋性能與寫入成本。

4.6 資料一致性與交易策略 (Consistency & Transaction Strategy)

為確保資料一致性與不變式，系統在以下情境中特別使用交易：

(1) 建立 Proposal (多卡提案 + 鎖定)

- 交易內步驟：

- 檢查目標 Listing 與提案方 Listings 狀態。
- 檢查是否已有 PENDING Proposal。
- 建立 Proposal + ProposalItem。
- 更新提案方所有 Offered Listings 的 locked_by_proposal_id。

- 任一步驟失敗則整個交易回滾。

(2) 提案接受 (SwapService) 包含

- 交易內步驟：

- 檢查 Proposal 狀態為 PENDING。
- 建立 Swap。
- 更新所有相關 Listing 狀態為「交易中」。

- 確保不會出現「Swap 已成立但 Listing 狀態未更新」的不一致問題

(3) 更新 Shipment / 新增 ShipmentEvent

- 新增 ShipmentEvent 時同時更新 Shipment 的 last_status。
- 保證 Shipment 與其事件歷程保持一致，避免顯示狀態與實際事件不符。

第五章 Use Case 實作對應

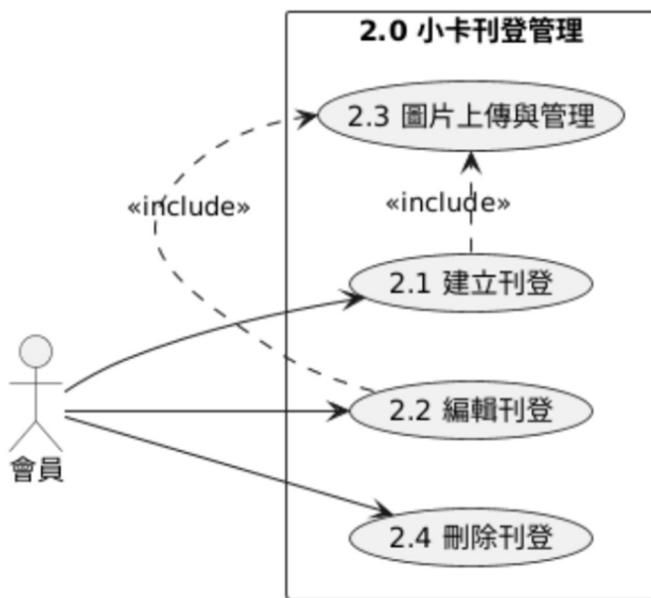
本章說明核心 Use Case 在系統內部的實作方式，將前述之需求與使用案例圖 (Use Case Diagram)，對應至實際的 Controller、Service、Repository、Entity 以及關聯到的外部系統或基礎設施。本章僅聚焦於：

- UC-02 小卡刊登管理 (Listing Management)
- UC-04 交易與提案管理 (Exchange Proposal & Transaction Management)

其餘 Use Case (身分驗證、搜尋服務、即時交流、出貨整合、貨品追蹤、Email 通知) 在第 2、3、4 章中已於架構與資料模型層級描述，不再逐一展開。

5.1 UC-02 小卡刊登管理 (Use Case Diagram)

(1) Use Case 範圍與目標



模組： 2.0 小卡刊登管理

主要 Use Cases :

- 2.1 建立刊登
- 2.2 編輯刊登
- 2.3 圖片上傳與管理
- 2.4 刪除刊登

目標：

讓會員能夠為自己的明星小卡建立、調整與刪除刊登，並上傳對應圖片與設定小卡領域屬性(idol_group、member_name、album、era、version、card_code、condition、has_protection 等)，形成後續搜尋、提案與交換的基礎資料。

(2) 參與角色

- **會員 (Member)**：建立／管理自己的小卡刊登。
- **系統 (Platform)**：驗證輸入、儲存 Listing、處理圖片上傳與檔案儲存。

(3) 前置條件

- 使用者已完成登入 (UC-01.3 使用者登入)。
- 使用者帳號狀態正常 (未被停權)。

(4) 後置條件

- **建立刊登成功時：**

- 新的 Listing 記錄已儲存於資料庫。
- 若有圖片上傳，實體檔案已存入 /uploads，路徑已記錄於 Listing.imagePaths。

● 編輯刊登成功時：

- Listing 被更新，並維持狀態與不變式。

● 刪除刊登成功時：

- Listing 狀態轉為「下架」或邏輯刪除（依實作而定），不再出現在搜尋結果中。

(5) 基本流程（以 UC-02.1 建立刊登 為主）

- (a) 會員於前端介面（刊登列表或個人頁）點選「建立刊登」。
- (b) 系統顯示「建立刊登」表單，包括：
 - 小卡基本資訊欄位（idol_group、member_name、album、era、version、card_code 等）
 - 卡況欄位（condition：S/A/B/C）
 - 是否有保護套（has_protection）
 - 需求描述、備註
 - 圖片上傳欄位（多張）
- (c) 會員填寫表單資料並上傳圖片，按下「送出」。
- (d) 系統驗證：欄位完整性與格式、必要領域欄位是否填寫。
- (e) 系統先儲存圖片檔案至 /uploads，並生成對應檔名或 URL。
- (f) 系統呼叫 ListingService 建立 Listing，包含：
 - 綁定 owner (User)
 - 設定 status = ACTIVE
 - 寫入圖片路徑
- (g) ListingService 呼叫 ListingRepository.save() 寫入資料庫。
- (h) 建立成功後，頁面導回「刊登詳情」或「我的刊登」頁，顯示新建立的小卡。

(6) 例外與替代流程

E1：欄位驗證失敗

- 條件：必填欄位未填寫／格式錯誤。
- 系統行為：

- 不呼叫 Service 建立 Listing。
- 回傳錯誤訊息，停留在相同表單，保留使用者已輸入之資料。

E2：圖片上傳失敗

- 條件：檔案過大／格式不符／檔案寫入失敗。
- 系統行為：
 - 顯示錯誤訊息，提示重新上傳。
 - 若尚未呼叫 ListingService，則不會建立 Listing。

E3：編輯／刪除非本人刊登 (UC-02.2 / UC-02.4)

- 條件：使用者嘗試操作不屬於自己的 Listing。
- 系統行為：
 - ListingService 驗證 ownerId == currentUserId，否則丟出錯誤。
 - Controller 返回「無權限操作」錯誤訊息或導向錯誤頁。

(7) 系統內部控制流程 (Controller / Service / Repository)

(a) Boundary (UI Controller / Page)

- UiListingController
 - GET /listings/new：顯示建立刊登表單。
 - POST /listings：接收表單資料與檔案上傳，轉換成 CreateListingRequest DTO 並呼叫 ListingService.createListing(request, currentUser)。
 - 或經由 ListingController 提供 REST API 版本 (POST /api/listings)。

(b) Control (ListingService)

- 驗證領域規則：
 - 基本欄位完整性（必填欄位不為空）。
 - 卡況 condition 是否為允許值。
- 整合圖片儲存：
 - 建立對應的檔名與路徑，存入 Listing.imagePaths。

● 建立 Entity :

➤ Listing listing = new Listing(...);

➤ 設定 owner、status = ACTIVE。

(c) Entity / Repository

- ListingRepository.save(listing)

- 透過 JPA / Hibernate 寫入 MySQL。

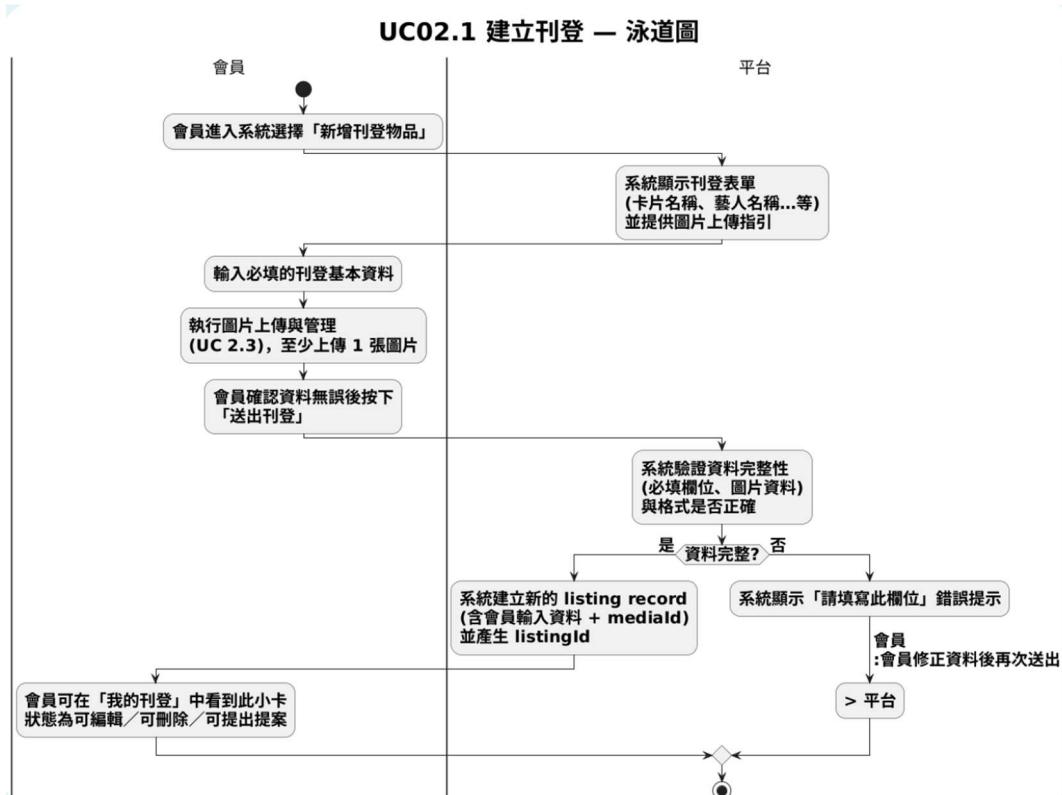
(d) 回應

- Service 回傳新建立的 Listing 或對應的 DTO。

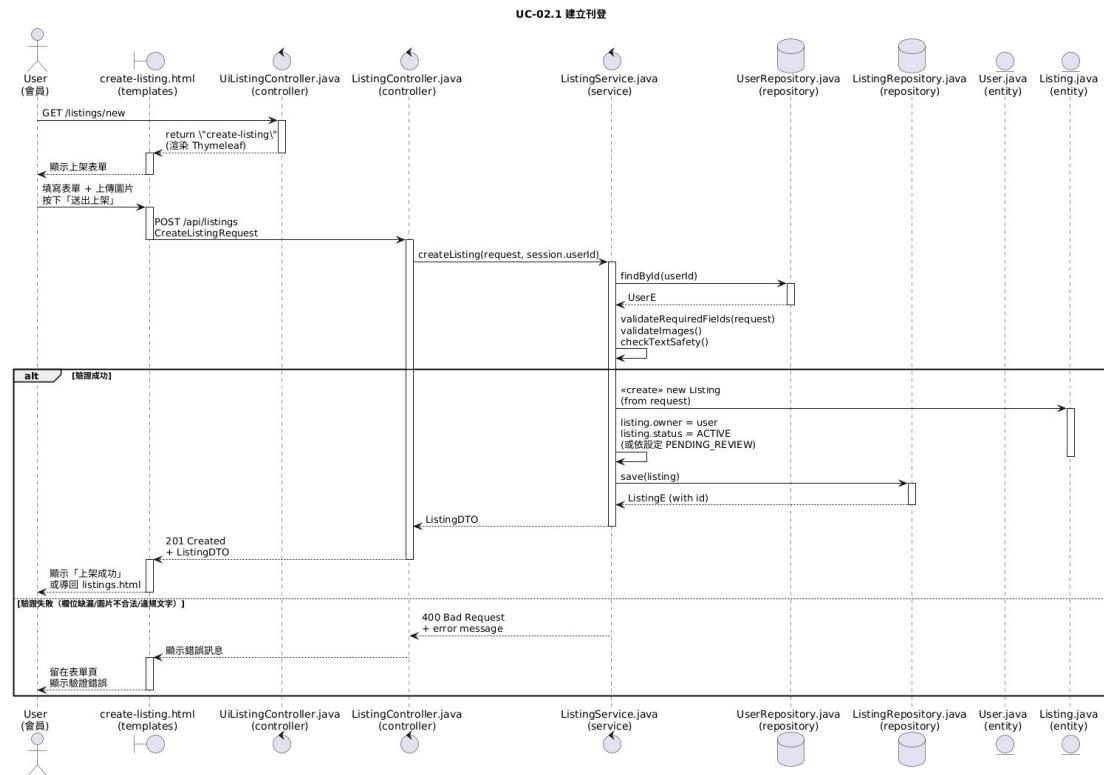
- Controller 根據結果導頁或回傳 JSON。

在本 Use Case 中，圖片上傳與實體檔案管理屬於 Infra Concern，由 Service 或 Utility 類別集中處理，Controller 不直接操作檔案系統路徑。

5.2 UC-02.1 建立刊登的泳道圖（Swimlane Diagram）



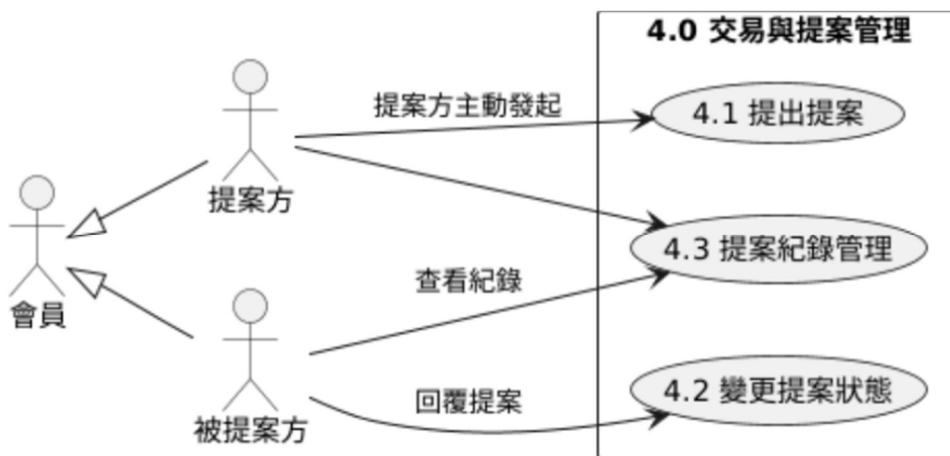
5.3 UC-02.1 建立刊登的循序圖 (Sequence Diagram)



5.4 UC-04 交易與提案管理

UC-04 為整個系統的核心之一，負責「從意願交換到交易成立」的關鍵流程。本節將以 UC-04.1 提出提案為主，並簡述 UC-04.2 / 4.3 的控制流程。

(1) Use Case 範圍與目標 (Use Case Diagram)



模組： 4.0 交易與提案管理

Use Cases：

- 4.1 提出提案
- 4.2 變更提案狀態（接受／拒絕／撤回）
- 4.3 提案紀錄管理（我的交換紀錄瀏覽）

目標：

- 讓提案方可以針對目標 Listing，選擇自己要拿出來交換的小卡並送出提案。
- 系統在送出提案時，負責驗證所有相關條件、建立 Proposal / ProposalItems、鎖定相關 Listing，並串接聊天室與 Email 通知。
- 被提案方可以在後續回覆提案（接受／拒絕），並在「我的交換」頁查看歷史紀錄。

(2) 參與角色

- 提案方（Proposer）：主動發起交換的會員。
- 被提案方（Owner）：被提案的小卡的持有者。
- 系統（Platform）：驗證提案、維護狀態機、建立聊天室與通知。

(3) 前置條件

- 提案方與被提案方皆為已登入會員。
- 目標 Listing 為 ACTIVE 且未被其他 Proposal 鎖定或交易完成。
- 提案方至少擁有一張可拿出來交換的小卡（自己的 Listing 且為 ACTIVE 且未鎖定）。

(4) 後置條件（成功送出提案）

- 系統已建立新的 Proposal 與對應的 ProposalItem 記錄。
- 系統已將提案方選擇的小卡 locked_by_proposal_id 設為此 Proposal id。
- 系統已建立對應的 ChatRoom 與第一則系統訊息。
- 系統已建立 EmailNotification，發送或排程寄送「你收到新的交換提案」通知給被提案方。
- 提案方可在「我的交換」頁中看到該筆提案的狀態（PENDING）。

(5) 提出提案 — 基本流程

對應你提供的 Use Case 圖 & Sequence / Swimlane：

- (a) 提案方在刊登列表或詳情頁點選「提出提案」。
- (b) 系統顯示提案表單：
 - 目標小卡資訊（被提案 Listing）。
 - 提案方自己的可用小卡清單（尚未被鎖定的 ACTIVE Listings）。
 - 提案訊息欄位（選填）。
- (c) 提案方勾選至少一張要拿出來交換的小卡，輸入訊息，按「送出提案」。
- (d) 系統驗證提案內容與小卡狀態：
 - 至少選一張小卡。
 - 目標小卡存在且為 ACTIVE，未被鎖定。
 - 提案方選擇的每張小卡皆為提案方所有、為 ACTIVE 且未被鎖定。
 - 對同一目標小卡不存在重複的 PENDING 提案。
- (e) 驗證通過 → 系統：
 - 建立 Proposal 主檔與 ProposalItem。
 - 將提案方選擇的小卡鎖定： $locked_by_proposal_id = proposal.id$ 。
 - 建立對應聊天室（ChatRoom）並寫入系統訊息。
 - 建立 EmailNotification 並寄送通知給被提案方。
- (f) 系統回傳成功結果，畫面顯示「提案已送出，可前往我的交換頁面查看」。

(6) 提出提案 — 例外流程

對應你 Sequence Diagram 中的 alt/else 分支：

- E1：未選擇任何小卡
 - 條件：提案方沒有勾選任何要交換的小卡。
 - 行為：
 - ProposalService 回傳錯誤「至少選擇一張小卡」。
 - Controller 將錯誤訊息帶回提案表單頁面。
 - 前端顯示錯誤，停留在同一頁。
- E2：目標小卡或提案方小卡已被鎖定／非 ACTIVE
 - 條件：

- 目標 Listing status != ACTIVE 或 locked_by_proposal_id != null 。
- 提案方任一張選擇的小卡為非 ACTIVE 或已被其他 Proposal 鎖定。

➤ 行為：

- Service 拋出錯誤（例如：「目標小卡目前不可被提案」、「你選擇的某張小卡已在其他提案中使用」）。
- Controller 顯示錯誤訊息，停留在同一頁。

● E3：已存在 Pending 提案

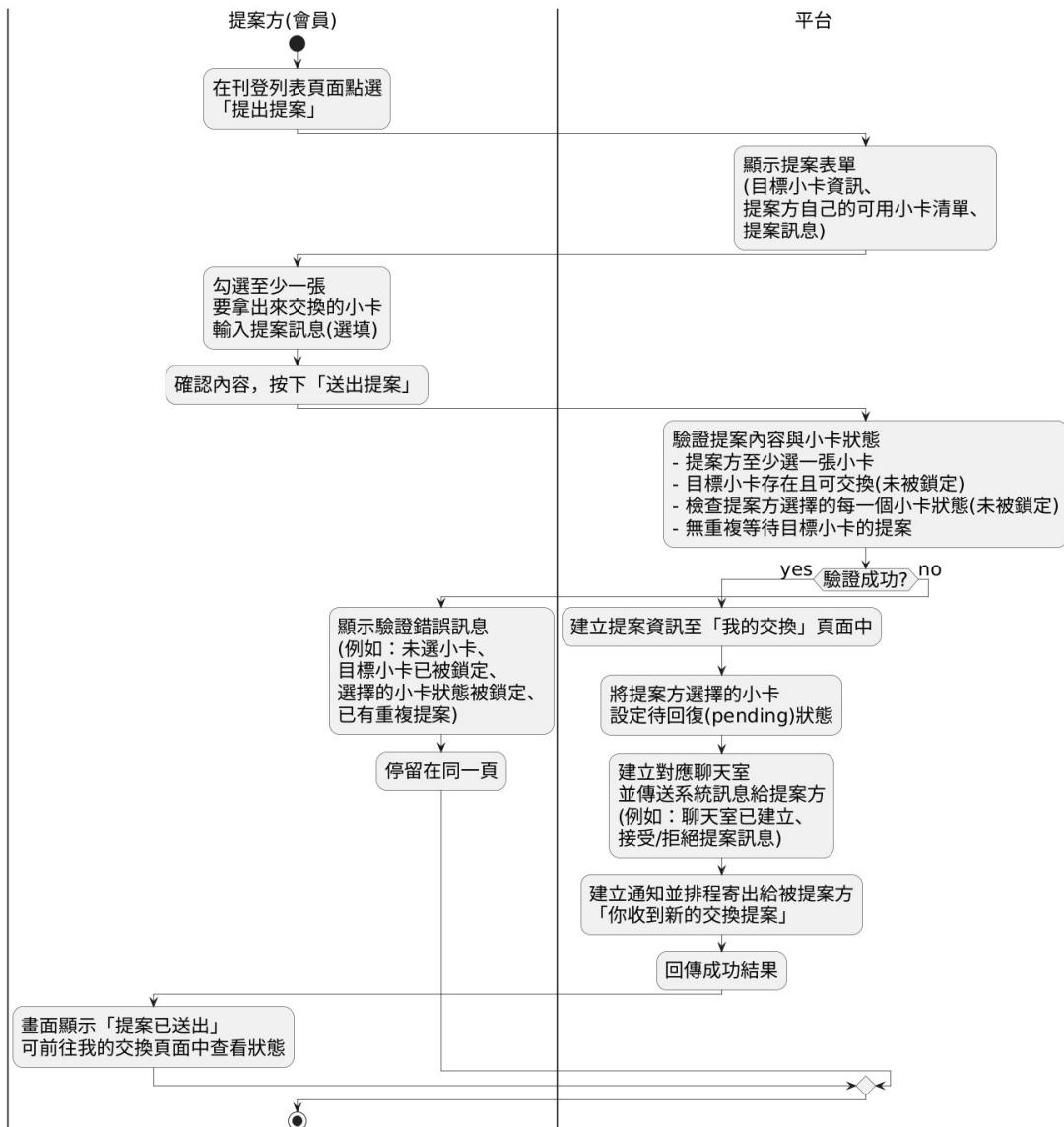
➤ 條件：

- ProposalRepository.findPendingByProposerAndTarget(propose rId, targetListingId) 回傳既有提案。

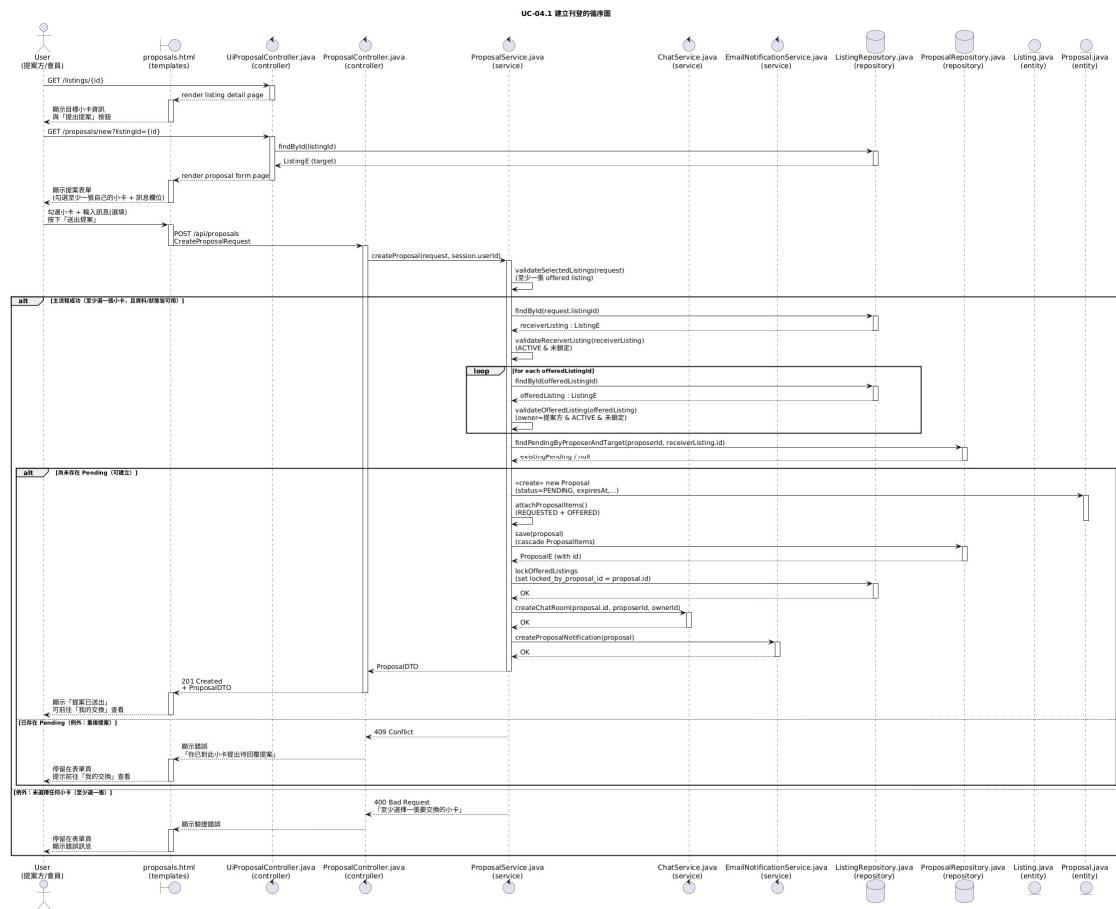
➤ 行為：

- 提案不會重複建立。
- 顯示：「你已對此小卡提出待回覆的提案，可前往『我的交換』查看」。

5.5 UC-04.1 建立刊登的泳道圖 (Swimlane Diagram)



5.6 UC-04.1 建立刊登的循序圖 (Swimlane Diagram)



第六章 非功能性需求與架構對應

本章說明 Exchange Platform 在架構層級如何因應系統的非功能性需求 (Non-Functional Requirements, NFR)，並明確指出各項需求在本系統中的實作或設計策略。

NFR 是架構設計的核心判準，用以確保系統在效能、安全性、可靠度、可維護性等面向都能達到課程要求及實務可用性。

6.1 效能

需求描述

- 系統需具備可接受的頁面載入速度與資料存取效率。
- 使用者瀏覽刊登、送出提案、查看交換紀錄的操作不應出現明顯延遲。
- 伺服器需能在單主機部署下支撐多名使用者同時使用。

架構對應設計

- Thymeleaf SSR (Server-Side Rendering)
 - 由後端直接組裝 HTML，降低前端渲染負擔，縮短首屏載入時間。
- Spring Data JPA + Lazy Loading 機制
 - 避免一次載入大量關聯資料，減少不必要的 SQL 查詢。
- 分頁查詢 (Pagination)
 - 搜尋列表、我的交換頁等皆採分頁，降低一次查詢資料量。
- 圖片儲存在本機檔案系統
 - 相較於 DB Blob 儲存，更快的 I/O 效能與更低 DB 壓力。
- Transaction Boundary 清晰
 - 僅核心寫入操作建立交易，有效控制單次請求的處理時間。

6.2 安全性

需求描述

- 系統需保護使用者資料、帳號資訊，避免越權操作。
- 登入流程需安全可靠。
- 使用者不得讀取、修改非自己擁有的 Listing / Proposal。

架構對應設計

- Google OAuth2 第三方登入
 - 授權機制由 Google 負責，降低密碼儲存風險。
- Session-Based Authentication
 - 使用 Spring Security 維護 User Session，避免未授權使用者進入保護區域。
- Access Control (Owner-Only Operations)
 - ListingService、ProposalService、SwapService 層均檢查 ownerId == currentUserId。
- 禁止直接操作 Repository
 - Controller 不可直接呼叫 Repository，所有存取必須經 Service 驗證邏輯。
- 檔案存取權限控制
 - 上傳檔案僅能由應用程式存取，對外僅提供安全 URL。
- 輸入資料驗證 (Validation)
 - 避免非法文字、注入攻擊、格式異常的資料寫入 DB。

6.3 資料一致性

需求描述

- 提案建立、鎖定小卡、建立聊天室與通知必須保持一致。
- 任何錯誤不得導致「半成功交易」或資料不完整。

架構對應設計

- Service Layer + @Transactional
 - 提案建立流程（Proposal + ProposalItems + Listing Lock + ChatRoom + EmailNotification）均包在單一交易內。
- Domain Invariant Enforcement（不變式）
 - Listing, Proposal, Swap, Shipment 皆具備明確狀態機制（ACTIVE/PENDING/ACCEPTED/...）。
- Row-Level Locking by Design
 - Listing 以 locked_by_proposal_id 控制併發錯誤，避免撞單。
- ShipmentEvent 事件流
 - 手動物流更新必須符合事件序列，避免不合法狀態跳躍。

6.4 可用性

需求描述

- 系統介面需明確、易懂，流程不能過於複雜。
- 操作錯誤需具備完整提示。

架構對應設計

- Thymeleaf SSR：預填資料與錯誤提示
 - 表單驗證失敗時顯示錯誤訊息並保留使用者輸入。
- 統一表單送出流
 - 所有表單均由 Controller 回傳 View，使用者不會跳脫流程。
- 統一錯誤處理（Exception Handler）
 - 提供 4xx/5xx 錯誤頁面，避免使用者看到 stacktrace。
- 模組化頁面（Listing / Proposal / Swap）
 - 讓使用者能清楚分辨上架、提案、出貨與收貨流程。

6.5 可靠性

需求描述

- 系統需避免常見的操作失敗情況（例如提案送一半失敗）。
- 儲存的資料需保持完整性。

架構對應設計

- 原子性交易
 - 重大流程（建立提案、接受提案、建立 Swap、出貨狀態更新）皆在 @Transactional 中運作。
- 錯誤回滾
 - 若任一子步驟失敗（如 Email 寄送失敗），能確保主資料不被破壞。
- Repository 分層
 - 資料存取統一由 JPA 控制，減少 human error 注入錯誤查詢。
- 本機檔案與資料庫分離
 - 圖片 I/O 錯誤不會破壞主要業務資料。

6.6 可維護性

需求描述

- 系統應能被團隊不同成員理解、維護與擴充。
- 架構須避免 Fat Controller / God Class。

架構對應設計

- 分層式單體架構 (Layered Monolith)
 - Boundary / Control / Entity 分工明確。
- Controller 薄，Service 厚
 - Controller 僅負責接收輸入、回傳結果。
 - 所有商業邏輯集中在 Service。
- 模組化分析 (1~8 模組)
 - Listing、Proposal、Chat、Shipment、EmailNotification 分成獨立 Service。
- DTO 分類明確
 - Request DTO：接收表單／API
 - Response DTO：回傳給前端
 - Internal DTO：跨 Service 參數傳遞（如 EmailTask）
- 清楚註解與命名
 - Entity、Repository、Service 命名皆以領域概念為基準 (Domain-Driven)。

6.7 可擴充性

需求描述

- 未來若增加新模組（如評價系統、圖鑑系統、配對模型）需能快速整合。
- 若改為前後端分離或 App，需要 API 能快速擴展。

架構對應設計

- REST API Controller 已預先拆出
 - 所有操作（建立 Listing、提出提案、更新出貨狀態）都有 API Endpoint，可直接替換為 Mobile / SPA。
- 圖片儲存抽象層
 - 雖目前使用本機 /uploads，Service 已能抽換成 S3。
- SMTP 可替換為 SendGrid / MailGun
 - EmailNotificationService 已抽象 SMTP 端點。
- 交貨便爬蟲可替換為正式 API
 - 目前以「爬蟲+手動事件」方式運作，仍保留未來正式 API 整合空間。
- Domain-Centric Design
 - 模組設計依 Use Case／Domain 劃分，新增功能不會破壞原本流程。

6.8 可測試性

需求描述

- 系統需容易撰寫單元測試與流程測試。
- 資料模型與 Service 邏輯需可被偵錯與驗證。

架構對應設計

- Service Layer 高內聚，可單獨測試
 - 透過 Mock Repository 可測驗提案流程、鎖定邏輯、狀態機運作。
- DTO-Based Input/Output
 - 使 Controller 層可用 MockMvc 輕易模擬。
- 明確的狀態機設計
 - Proposal / Swap / Shipment 狀態轉換有明確規則，比較容易撰寫 Test Case。
- Repository 介面標準化
 - JPA 查詢可透過嵌入式 H2 DB 或 MySQL Docker 撰寫測試

6.9 操作性

需求描述

- 系統在單主機部署下需容易啟動、除錯、部屬。

- 日誌記錄需足以協助除錯。

架構對應設計

- 單一 Spring Boot Jar 可啟動整個系統
 - 適合課堂 Demo 與本機除錯。
- Logback 設定
 - 記錄 Controller / Service / Error 訊息。
- 分層日誌
 - 錯誤、資料庫查詢、Email 送出皆有獨立 log category。

6.10 系統限制

需求描述

- 課堂專案僅需單主機部署，不採用 Kubernetes、微服務、Redis、Queue 等架構。

架構對應設計

- Layered Monolith
 - 在單專案 / 單主機限制下，以 OOAD + BCE 分層即可滿足擴充性需求。
- 本機檔案系統
 - 避免引入 S3、GCS 等高複雜度設備。
- 無金流平台
 - 避免 PCI-DSS 等合規需求，使安全性與資料處理大幅簡化。

6.11 NFR 與設計對應表

NFR 類別	設計對應措施
Performance	SSR、JPA Lazy、分頁、交易範圍縮小、Local File Storage
Security	OAuth2、Session Security、Owner Check、Validation、檔案存取保護
Data	@Transactional 、 Row Locking 、 Domain Invariants 、
Consistency	ShipmentEvent
Usability	表單錯誤提示、SSR Render、Exception Handler、分頁 UI

NFR 類別	設計對應措施
Reliability	Transaction Rollback、單點錯誤隔離、Repository 標準化
Maintainability	分層架構、Service Thick、DTO 分層、模組拆分
Scalability	API-ready、Storage/Linux 抽換、SMTP 抽換、Domain 驅動設計
Testability	Mock Repository、狀態機測試、MockMvc、H2 測試
Operability	單 Jar 部署、Logback、分層日誌
Constraints	Layered Monolith、Local Storage、無金流

第七章 風險與改善方向

風險編號	風險描述	可能影響	緩解策略
R1	併發鎖定失敗，兩人同時搶同一張卡	產生重複或不一致的交換紀錄	善用 locked_by_proposal_id、加上資料庫層級鎖或樂觀鎖機制，並在 Service 加入重試或衝突處理
R2	手動更新物流不確實或延遲	造成雙方誤解或爭議	要求附上出貨／收貨證據圖片，ShipmentEvent 留存時間戳，未來可搭配爭議處理流程
R3	Email 通知寄送失敗	使用者收不到提案／出貨通知	EmailNotification 記錄寄信狀態，失敗可重送，並提供站內通知作為備援
R4	安全性不足（XSS / SQL Injection 等）	資料被竄改或外洩	持續使用參數化查詢與輸入驗證、加強測試與程式碼審查

風險編號	風險描述	可能影響	緩解策略
R5	部署環境為單一節點	服務中斷風險較高	未來可將 DB 與 Application 拆機，並預留雲端擴充空間
R6	不支援大規模使用	在大量使用者下效能下降	日後可加入 Cache、調整索引、進行壓力測試與效能調校

第八章 結語與後續擴充建議

本章總結 Exchange Platform 的整體架構設計成果，並提出未來可在系統、架構、功能與非功能需求等面向進行的擴充方向。本平台以「明星小卡交換」為核心，透過完整的 Domain Model、提案鎖定機制、雙向出貨流程、事件流物流追蹤與多模組 Use Case 設計，成功建立一套具備實務可行性、可維護性與可擴充性的專案架構。

8.1 系統成果總結

本專案在課程範疇下，已完成一套具備實際交換流程的 Web 應用系統，具體成果如下：

(1) 以 Domain-Driven 的方式建構明星小卡交換模型

- 小卡 Listing 擁有 idol_group、member_name、album、era、version、card_code 等專屬欄位
→ 與真實小卡交換情境一致。
- 完整資料字典、資料庫模型與狀態機，提供一致性與可維護性。

(2) 完整的提案模型與併發防護

系統成功處理「多卡換一張」、「重複提案」、「撞單」等複雜場景，包含：

- Listing 的 row-level locking (locked_by_proposal_id)
- Proposal + ProposalItem 結構支援多對多交換
- Pending 限制避免惡意或重複提案

- Service + Transaction 邊界確保邏輯原子性

此部分為全系統最關鍵亮點。

(3) Swap + Shipment + ShipmentEvent 的事件流物流設計

- 每次交換建立雙向 Shipment
- 手動物流事件模型 (ShipmentEvent) 清楚紀錄物流歷程
- 7-11 交貨便查詢頁整合爬蟲模組提升使用性
- 配合出貨、收貨、交易完成的狀態機一致性

是課堂範圍內少見的完整「雙向交易物流」流程。

(4) 提案 → 聊天室 → 通知的一條龍流程

本系統將「提案」視為所有互動的起點：

- 提案成功後建立 ChatRoom
- 自動寫入系統訊息
- 由 EmailNotificationService 負責寄出郵件通知

從行為層到技術層，都有效整合，提升平台互動完整度。

(5) 架構清晰、易於維護的 BCE + Layered Monolith

- Controller (Boundary)
- Service (Control)
- Entity / Repository (Entity)

完整符合課堂 BCE 設計概念，並且每個 Use Case 均對應控管邏輯、流程圖、Sequence Diagram，展現良好架構與文件一致性。

(6) 無金流平台的安全性與規範

平台避免金錢交易，透過：

- 不提供價格欄位
- 後端文字檢查（電話、Line ID、金額）
- 前端 UI 顯示限制
- 資料模型完全不含 price / amount 欄位

有效降低詐騙、責任歸屬與金流風險。

8.2 系統限制

雖已具備核心交換功能，仍有部分限制如下：

(1) 缺乏即時雙向 WebSocket 聊天

目前聊天室為同步拉資料方式 (Polling 或 Refresh)，不支援即時推播。

(2) 外部物流查詢以爬蟲為主

此方式可能受 HTML 結構變動影響，未使用正式 API。

(3) 未採用分散式架構

目前為單體部署，不支援多節點、高併發或水平方向擴充。

(4) 缺乏完整管理者後台 (Admin)

缺少平台管理、檢舉、封鎖、違規審查等管理者操作。

(5) 手機版 UI 尚未完全最佳化

仍偏向桌面瀏覽體驗，尚未針對 RWD 深度調整。

8.3 後續擴充建議

以下依技術、功能、架構提出擴充方向。

A. 功能性擴充 (Functional Enhancements)

(1) 完整的通知中心 (In-App Notification Center)

目前依賴 Email，未提供站內通知。

可新增：

- 提案提醒
- 提案狀態變更
- 聊天訊息提醒
- 出貨進度通知

(2) 小卡收藏、追蹤與推薦功能

可以使平台更貼近粉絲使用習慣：

- 收藏卡片／收藏賣家
- 推薦相似成員或同輯卡片

- 推薦交換可能性高的使用者 (Matching Engine)

(3) 提案方與被提案方的評價系統

- 交易結束後可互評
- 避免詐騙、爽約
- 提供更安全的交換生態系統

(4) 詳細交易歷程頁

將提案 → swap → 出貨 → 收貨端到端以 Timeline 呈現。

B. 技術性擴充 (Technical Improvements)

(1) WebSocket / SSE 即時聊天

提升聊天即時性：

- 即時訊息推播
- 提案狀態更新推播
- 出貨進度自動更新

(2) 前後端分離 (SPA / Mobile App)

可依 Use Case 拆分 API：

- React / Vue / Next.js
- Flutter App

平台未依賴 SSR，因此具備良好擴充彈性。

(3) 引入物件儲存服務

將圖片儲存從本機移動至：

- AWS S3
- GCP Cloud Storage

提升擴充性與 CDN 效能。

(4) 更安全的使用者內容審查

避免詐騙：

- 檢測 Line ID / 電話替代寫法

- NLP 過濾可疑內容
- 圖片 Hash 比對可疑卡片（撞圖）

C. 架構性擴充 (Architectural Enhancements)

(1) 由 Monolith → Modular Monolith → Microservices

未來可逐步拆成模組：

- Auth Module
- Listing Module
- Proposal Module
- Messaging Module
- Shipping Module

或以 Event-Driven (Kafka / RabbitMQ) 方式分散高流量模組。

(2) 自動化部署 (CI/CD)

可導入 GitHub Actions / GitLab CI：

- 自動建置
- 自動部屬
- 單元測試與整合測試

(3) Logging / Monitoring

加入：

- ELK / Grafana
- Application Metrics
- Error Alert 通知

協助日後維運。

8.4 本專案之學習價值

此專案完整符合課程「系統分析與設計」之學習目標，具體包括：

- 實作完整 BCE 架構
- 規劃 Use Case → Sequence → Activity → Data Model 的完整流程
- 嚴謹的 Domain Model 與狀態機管理
- 與外部系統整合 (OAuth2、SMTP、物流查詢)

- 合理的系統設計、例外處理與非功能需求分析

也展現了團隊在大型專案結構化思考、技術落實與文件撰寫上的成熟度。

8.5 結語

Exchange Platform 在「明星小卡交換」的明確場域中，透過精細化的 Domain Model、嚴謹的提案鎖定流程、雙向出貨邏輯與事件式物流追蹤，成功建立一套具備完整交換體驗的 Web 平台。

整份 SADD 文件提供：

- 系統架構完整描述
- 資料模型與資料流向
- Use Case → Sequence → Activity 的對應
- 非功能需求與設計對應
- 錯誤處理、防呆、狀態機規劃

本系統具備良好的維護性、可讀性與擴充基礎，亦展現了高於課堂作業需求的工程品質。若未來持續擴充，本平台可逐步演進至完整的交易撮合與使用者社群生態系統。