服飾業進銷存系統規格書

1. 專案概述 (Project Overview)

1.1 專案目標與背景 (Project Goal & Context)

本專案旨在為「一人營運」的服飾業經營者，打造一套輕量級但邏輯嚴謹的企業資源規劃 (ERP) 系統。市面上的 SaaS 進銷存系統往往過於龐大昂貴，或缺乏針對「多規格 SKU (顏色/尺寸)」與「多通路手續費」的特化功能。

本系統的核心價值在於「自動化」與「精確的財務視角」：

- 解決痛點：消除多平台（蝦皮、賣貨便、官網）手續費計算的繁瑣、解決供應商採購比價困難、以及缺乏加權平均成本導致的毛利失真。
- 關鍵產出：提供即時的庫存價值、自動化的數據備份機制，以及整合會員管理的 CRM 功能。
- 使用情境：單機部署、單人操作，無須複雜的權限控管 (ACL)，強調操作效率與介面響應速度。

1.2 技術堆疊詳細規格 (Detailed Tech Stack)

前端 (Frontend)

採用 Single Page Application (SPA) 架構，確保操作體驗流暢如原生應用程式。

- **Core Framework: Nuxt.js 3 (Latest)**
  - 設定*：*採用 `ssr: false` 模式，作為純客戶端渲染 (CSR) 應用，降低部署複雜度。
  - 狀態管理*：*使用 Pinia 進行全域狀態管理 (如購物車暫存、全域設定快取)。
- **UI Library: Vuetify 3 (Latest)**
  - 用途*：*提供標準化 Material Design 元件，特別是強大的 `v-data-table` (用於庫存列表) 與 `v-dialog` (用於單據操作)。

    - 圖標*:* 使用 **Material Design Icons (MDI)**。
- **Styling: TailwindCSS**
    - 用途*:* 處理 **Vuetify** 難以覆蓋的細微排版 **(Flex/Grid** 微調、**Padding/Margin** 工具類**)**。
- **Interaction: SweetAlert2**
    - 用途*:* 封裝全域的 `Confirm` **(**刪除確認**)**、`Toast` **(**成功訊息**)**、`Alert` **(**庫存不足警告**)**，統一互動體驗。

後端 (Backend)

採用標準 **RESTful API** 架構，強調資料一致性與事務安全。

- **Core Framework: Java Spring Boot 3.x**
    - ***JDK*** 版本*:* **Java 17 LTS** 或 **Java 21 LTS**。
    - 建置工具*:* **Maven**。
    - ***API*** 規範*:* 統一回傳格式 **(ResultResponse wrapper)**。
- **Persistence Layer: MyBatis**
    - 選擇理由*:* 相較於 **JPA/Hibernate**，**MyBatis** 能提供對 **SQL** 語句的絕對控制權，便於撰寫複雜的報表查詢與特定的鎖定語法 **(**`UPDLOCK`**)**。
    - ***Code Gen:*** 使用 **MyBatis Generator** 自動產生基礎 **CRUD Mapper** 與 **XML**。
- **Task Scheduling: Spring Scheduler**
    - 用途*:* 執行每月 **1** 號的自動資料庫備份排程，以及系統啟動時的備份檢查。

資料庫 (Database)

- **DBMS: Microsoft SQL Server (Express** 或 **Developer Edition)**
    - 選擇理由*:* 針對交易事務 **(Transaction)** 處理極為穩健，且方便與 **Windows** 環境整合備份指令。
    - ***Schema*** 規範*:* 所有 **Table** 均需具備 `CreatedAt` **(Default GETDATE())** 與 `UpdatedAt`。

並發控制 (Concurrency Control)

即使是單人系統，為了防止「**UI** 連點」、「排程與人工操作衝突」或「未來擴充

多人」，必須在核心層實作鎖定。

- 策略: 悲觀鎖 (Pessimistic Locking)。
- 實作:
  - 在庫存扣減 (`InventoryService.decreaseStock`) 或進貨入庫時，必須在 `@Transactional` 交易內。
  - 使用 SQL Server 特有的 Row-level Locking 語法。
  - MyBatis XML 範例: `SELECT * FROM Inventory WITH (UPDLOCK, ROWLOCK) WHERE SkuID = #{skuId}`。
  - 效果*:* 確保在讀取庫存到寫入庫存的這段微秒時間內，該筆資料被鎖定，防止髒讀 (Dirty Read) 或更新遺失 (Lost Update)。

1.3 系統邊界與非功能需求 (System Boundaries)

- 安全性 (Security): 由於是單機單人使用，不實作登入驗證 (Authentication) 與 授權 (Authorization)。所有 API 請求視為來自 Admin，Audit Log 中的 `CreatedBy` 統一寫入 "SystemAdmin"。
- 部署環境: 預計部署於使用者本地 Windows PC (Localhost)。
- 備份策略: 採用「冷備份」邏輯，利用 SQL Server `BACKUP DATABASE` 指令匯出 `.bak` 檔至指定本機路徑。

---

2. 資料庫設計 (Database Schema Detail)

2. 資料庫設計 (Database Schema Detail) 本章節定義資料庫實體模型 (Physical Data Model)。

- DBMS: Microsoft SQL Server.
- 編碼 (Collation): Chinese_Taiwan_Stroke_CI_AS (不區分大小寫)。
- 通用欄位: 所有 Table (除關聯表外) 預設包含 `CreatedAt` (Default: `GETDATE()`) 與 `UpdatedAt`。

全域資料限制 (Global Data Constraints):

1. 不可變性 (Immutability): 系統內所有 Primary Key (PK) 欄位一經建

立 (Insert)，即禁止修改。若需變更（如 SKU 編碼規則改變），必須將舊資料停用 (`IsActive=0`) 並建立新資料，以確保歷史關聯的正確性。

2. 主檔刪除保護 (Master Data Deletion Policy):
    - 適用範圍: `Products`, `SKUs`, `Suppliers`, `Members`, `SalesChannels`, `ReturnReasons`.
    - Logic: 執行刪除前，必須檢查是否已存在關聯的交易單據 (PO, SO, Logs)。
        - 無交易: 允許物理刪除 (`DELETE`)。
        - 有交易: 禁止物理刪除，僅允許更新 `IsActive = 0` (軟刪除)。

2.1 系統設定與備份 (System Configuration)

用於儲存不需頻繁更動的全域參數與自動化排程紀錄。

Table: `SystemConfigs`

儲存 key-value 形式的設定參數。

- ConfigKey (`NVARCHAR(50)`, PK): 設定鍵值。例: `'ImgStoragePath'`, `'BackupStoragePath'`.
- ConfigValue (`NVARCHAR(MAX)`, Not Null): 設定值。支援長路徑字串。
- Description (`NVARCHAR(200)`, Nullable): 參數說明 (供維護者閱讀)。
- 初始化數據*:* 系統部署時需預先 Insert 必要 Key。

Table: `BackupLogs`

紀錄 `SystemBackupService` 的執行結果。

- LogID (`BIGINT`, PK, Identity): 流水號。
- BackupFileName (`NVARCHAR(255)`): 產生的備份檔名 (e.g., `ERP_Backup_202512.bak`).
- Status (`NVARCHAR(20)`): `'SUCCESS'`, `'FAILED'`, `'SKIPPED'`.

- **Message (`NVARCHAR(MAX)`):** 若失敗，紀錄 Exception Message。
- **ExecutedAt (`DATETIME2`, Default `GETDATE()`):** 執行時間。

Table: ReturnReasons (退貨理由主檔)

- **ReasonID (INT, PK, Identity):** 理由 ID。
- **ReasonType (VARCHAR(10), Not Null):** 適用類型。
  - Values: `'SO'` (銷貨退回), `'PO'` (採購退出), `'ALL'` (通用)。
- **ReasonText (NVARCHAR(50), Not Null):** 理由描述 (e.g., '尺寸不合', '瑕疵品', '改變心意', '配送延遲')。
- **SortOrder (INT, Default 99):** 排序權重。
- **IsActive (BIT, Default 1):** 是否啟用。

2.2 主檔資料 (Master Data)

Table: `Products` (商品單頭)

- **ProductID (`NVARCHAR(50)`, PK):** 商品代號。
- **ProductName (`NVARCHAR(100)`, Not Null):** 商品名稱。
- **BasePrice (`DECIMAL(18, 0)`, Not Null):** 目前標準售價 (整數)。
- **Description (`NVARCHAR(MAX)`, Nullable):** 商品描述。

Table: `ProductImages` (商品圖片)

- **ImageID (`BIGINT`, PK, Identity):** 圖片流水號。
- **ProductID (`NVARCHAR(50)`, FK):** 關聯商品。*Delete Cascade*.
- **FilePath (`NVARCHAR(255)`, Not Null):** 圖片實體檔名 (e.g., `P001_173478900.jpg`)。
- **SortOrder (`INT`, Not Null):** 排序 (1-8)。
  - ***Constraint:*** 程式邏輯需確保同一 ProductID 的 SortOrder 不重複且連續，`1` 為主圖。

Table: `Suppliers` (供應商)

- **SupplierID (`NVARCHAR(20)`, PK):** 供應商代號。
- **SupplierName (`NVARCHAR(100)`, Not Null):** 供應商名稱。

Table: `SalesChannels` (銷售通路)

- **ChannelID (`INT`, PK, Identity):** 通路 ID。
- **ChannelName (`NVARCHAR(50)`, Not Null):** 通路名稱 **(e.g., '蝦皮', '官網').**
- **FeeRate (`DECIMAL(5, 4)`, Default 0.0000):** 平台手續費率。
    - ***Design:*** 使用 **4** 位小數，例如 **`0.0550`** 代表 **5.5%**。
- ReturnShippingFee (DECIMAL(18, 0), Default 0): 退貨運費負擔。
    - Design: 設定該通路發生退貨時，賣家需自行吸收的標準運費成本（例如：蝦皮逆物流 $60、官網宅配 $120）。此金額將用於退貨單的費用認列。

Table: `DeliveryMethods` (配送方式)

- **MethodID (`INT`, PK, Identity):** 方式 ID。
- **MethodName (`NVARCHAR(50)`, Not Null):** 方式名稱 **(e.g., '7-11 店到店').**
- **IsActive (`BIT`, Default 1):** 是否啟用。

Table: `Members` (會員)

- **MemberID (`BIGINT`, PK, Identity):** 會員流水號。
- MemberCode (VARCHAR(30), NOT NULL): 會員編號。
    - ***Index:*** 需建立 **`Unique Index`**。
- **Name (`NVARCHAR(50)`, Not Null):** 會員姓名。
- **Phone (`VARCHAR(20)`, Not Null):** 手機號碼。
    - ***Index:*** 需建立 **`Unique Index`**。
- **Address (`NVARCHAR(255)`, Nullable):** 寄送地址。
- **Note (`NVARCHAR(MAX)`, Nullable):** 備註。

2.3 庫存核心 (Inventory Core)

此部分為系統心臟，涉及金錢與庫存準確度，需最高嚴謹度。

Table: `SKUs` (庫存單元)

- **SkuID (`VARCHAR(100)`, PK):** 組合鍵字串 (`ProductID+SupplierID+Color+Size`).
    - ***Validation:*** 程式端 **Regex `^[A-Za-z0-9]+$`**。
- **ProductID (`NVARCHAR(50)`, FK):** 關聯商品。
- **SupplierID (`NVARCHAR(20)`, FK):** 關聯供應商。
- **Color (`NVARCHAR(20)`, Not Null):** 顏色代碼/名稱 (拆分欄位以便查詢)。
- **Size (`NVARCHAR(20)`, Not Null):** 尺寸代碼/名稱 (拆分欄位以便查詢)。
- **PurchasePrice (`DECIMAL(18, 0)`, Not Null):** 預設進貨成本。
    - ***Trigger Logic:*** 修改此欄位時，需觸發 **Service** 更新 **`Draft`** 狀態 **PO** 的單價。
- **IsActive (`BIT`, Default 1):** 若供應商該色號停產可設為 **0**。
- **SafetyStock (INT, Default 0):** 安全庫存量。
    - **UI Logic:**在庫存列表顯示時，若 `Quantity <= SafetyStock`，該行數量需標示為紅色或顯示警告圖示，提示使用者補貨。

Table: `Inventory` (即時庫存)

- **SkuID (`VARCHAR(100)`, PK, FK):** 關聯 **SKU**。
- **Quantity (`INT`, Not Null, Default 0):** 目前庫存量。允許負數 (若開啟負庫存功能)。
- **AvgCost (`DECIMAL(18, 4)`, Not Null, Default 0):** 加權平均成本。
    - ***Design:*** 使用 **4** 位小數 **(e.g., 150.3333)** 避免多次運算後的四捨五入誤差，顯示時再取整數。
- **Concurrency Note:**
    - 所有涉及 **Update `Quantity`** 或 **`AvgCost`** 的交易，**SQL** 查詢必須包含 **`WITH (UPDLOCK, ROWLOCK)`**。
- **Zero Cleanup Logic:** 為了避免浮點數運算導致的尾差（例如：數量為 0 但成本剩餘 $0.0001），所有涉及庫存扣減的 Service (Sales, Adjustments, Returns) 必須包含以下邏輯：
    - `IF (UpdatedQuantity == 0) THEN SET AvgCost = 0;`
    - 確保當庫存歸零時，庫存價值強制歸零。

Table: `StockTransactionLogs` (庫存流水帳)

- LogID (BIGINT, PK, Identity): 流水號。
- DocType (VARCHAR(20), Not Null): 單據類型。
  - PO_IN (採購進貨)
  - PO_RET (採購退回)
  - SO_OUT (銷貨出庫)
  - SO_RET (銷貨退回)
  - ADJ (庫存調整)
- DocNo (NVARCHAR(50), Not Null): 來源單號。
- SkuID (VARCHAR(100), Not Null): 異動 SKU。
- QtyChange (INT, Not Null): 異動數量 (進貨為正，銷貨為負)。
- CostBefore (DECIMAL(18, 4)): 異動前平均成本。
- CostAfter (DECIMAL(18, 4)): 異動後平均成本。
- CreatedAt (DATETIME2, Default GETDATE()): 發生時間。

2.4 採購與進貨 (Procurement)

Table: PurchaseOrders (採購單)

- DocNo (NVARCHAR(50), PK): 單號 (e.g., PO202512210001).
- DocDate (DATE, Not Null): 單據日期。
- Status (TINYINT, Not Null):
  - 0: Draft (未確認)
  - 1: Confirmed (已確認/待進貨)
  - 2: Closed (已結案)
  - 3: ForceClosed (強制結案)
- ExternalDocNo (NVARCHAR(100), Nullable): 供應商收據/發票號碼。

Table: PODetails (採購明細)

- DetailID (BIGINT, PK, Identity): 明細流水號。
- DocNo (NVARCHAR(50), FK): 關聯 PO。
- SkuID (VARCHAR(100), FK): 採購品項。

- **OrderQty (`INT`, Not Null):** 採購量。
- **ReceivedQty (`INT`, Default 0):** 已到貨量。
- **ActualPrice (`DECIMAL(18, 0)`, Not Null):** 實際進貨單價。
    - *Logic:* 建立時預設帶入 `SKUs.PurchasePrice`。若 `Status` 為 Draft/Confirmed，SKU 主檔價格變動會更新此欄位。

Table: ReceivingNotes (進貨單)

- **RIDocNo (`NVARCHAR(50)`, PK):** 進貨單號 (e.g., RI20251221001)。
- **PODocNo (`NVARCHAR(50)`, FK):** 來源採購單。
- **DocDate (`DATE`, Not Null, Default `GETDATE()`):** 進貨/入庫日期。
- **Status (`TINYINT`, Not Null):**
    - `0`: Draft (點收中，不影響庫存)。
    - `1`: Confirmed (確認入庫，觸發成本計算與庫存增加)。
- **Note (`NVARCHAR(200)`, Nullable):** 備註 (e.g., 外箱破損，但內容物無損)。

Table: RIDetails (進貨明細)

- **DetailID (`BIGINT`, PK, Identity):** 流水號。
- **RIDocNo (`NVARCHAR(50)`, FK):** 關聯進貨單。
- **SkuID (`VARCHAR(100)`, FK):** 進貨品項。
- **Qty (`INT`, Not Null):** 本次實收數量。
    - *Logic:* 預設帶入 PO 中該 SKU 的「未交數量」(`OrderQty - ReceivedQty`)，允許使用者手動修改（例如廠商只送來一半）。

Table: PurchaseReturns (採購退回/退出單)

- **ReturnDocNo (NVARCHAR(50), PK):** 退貨單號 (e.g., PR202512...)。
- **SupplierID (NVARCHAR(20), FK):** 接收退貨的供應商。
- **DocDate (DATE, Not Null):** 退貨日期。
- **Status (TINYINT):** 0: Draft, 1: Confirmed (扣帳)。
- **ReasonID (INT, FK):** 關聯 `ReturnReasons`。
    - **UI:** 下拉選單，僅列出 `ReasonType IN ('PO', 'ALL')` 的項目。

Table: PurchaseReturnDetails (採購退回明細)

- `DetailID` (BIGINT, PK, Identity): 流水號。
- `ReturnDocNo` (FK): 關聯主檔。
- `SkuID` (FK): 退回品項。
- `Qty` (INT, Not Null): 退回數量。
- `ReturnPrice` (DECIMAL(18, 0)): 退貨單價 (通常為原進貨價，用於向供應商請款)。
- `CostAtMoment` (DECIMAL(18, 4)): 成本快照 (用於扣減庫存價值，依當下加權平均成本)。

2.5 銷貨與退貨 (Sales & Returns)

Table: `SalesOrders` (銷貨單)

- **DocNo (`NVARCHAR(50)`, PK): 單號 (e.g., SO202512210001).**
- **DocDate (`DATETIME2`, Not Null): 銷貨時間。**
- **MemberID (`BIGINT`, FK): 會員 (預設 1 為 Guest).**
- **ChannelID (`INT`, FK): 銷售通路。**
- **DeliveryMethodID (`INT`, FK): 配送方式。**
- **TrackingNo (`NVARCHAR(50)`, Nullable): 物流單號。**
- **ExternalDocNo (`NVARCHAR(100)`, Nullable): 網購平台訂單編號。**
- **PlatformFee (`DECIMAL(18, 0)`, Default 0): 手續費成本 (依費率計算後寫死)。**
- **PaymentStatus (`TINYINT`, Default 0): `0`: Unpaid, `1`: Paid.**
- **PaidAmount (`DECIMAL(18, 0)`, Default 0): 實收金額。**
- **TotalAmount (`DECIMAL(18, 0)`): 訂單總額 (快照)。**

Table: `SODetails` (銷貨明細)

- **DetailID (`BIGINT`, PK, Identity): 明細流水號。**
- **DocNo (`NVARCHAR(50)`, FK): 關聯 SO。**
- **SkuID (`VARCHAR(100)`, FK): 銷貨品項。**
- **Qty (`INT`, Not Null): 數量。**
- **UnitPrice (`DECIMAL(18, 0)`, Not Null): 售價快照 (Snapshot)。**
    - ***Logic:* SO 建立當下寫入，後續 Product 價格變動不影響此**

值。
- **CostAtMoment (`DECIMAL(18, 4)`, Not Null):** 成本快照 (Snapshot)。
    - ***Logic:*** **SO 建立當下的 `Inventory.AvgCost`，用於鎖定該筆交易的利潤計算。**

Table: SalesReturns (銷貨退回)

- **`ReturnDocNo (NVARCHAR(50), PK)` (格式範例: SR202512...)。**
- **ReturnDate (DATETIME2, Default GETDATE()): 退貨時間。**
- **OriginalSODetailID (BIGINT, FK): 關聯原始銷貨明細。**
    - **Crucial: 透過此 ID 查找 `SODetails.CostAtMoment` 與 `UnitPrice`，確保退款與退庫存金額正確。**
- **Qty (INT, Not Null): 退貨數量。**
- **ReasonID (INT, FK): 關聯 `ReturnReasons`。**
    - **UI: 下拉選單顯示 `ReasonText`，僅列出 `ReasonType IN ('SO', 'ALL')` 且啟用的項目。**
- **RefundType (VARCHAR(20), Default 'CASH'): 目前僅支援現金/刷退 ('CASH')。**
- **ReturnShippingFee (DECIMAL(18, 0), Default 0): 逆物流費用（賣家負擔）。**
    - **Logic: 建立時由 `SalesChannels.ReturnShippingFee` 預設帶入，允許使用者手動修改。此金額將列入營業費用計算。**

2.6 庫存盤點與調整 (Inventory Count & Adjustment) - 新增章節

Table: StockTakingNotes (盤點單)

- `DocNo` (NVARCHAR(50), PK): 盤點單號 (e.g., ST2025...)。
- `DocDate` (DATE, Default GETDATE()): 盤點日期。
- `Status` (TINYINT):
    - `0: Draft` (盤點中)
    - `1: Counted` (數量輸入完畢/待簽核)
    - `2: Approved` (已簽核/鎖定)
    - `3: Void` (作廢)

Table: StockTakingDetails (盤點明細)

- `DetailID` (PK): 流水號。
- `DocNo` (FK): 關聯盤點單。
- `SkuID` (FK): 盤點品項。
- `SystemQty` (INT): 系統庫存快照 (建立盤點單當下的 `Inventory.Quantity`)。
- `CountQty` (INT): 實盤數量 (使用者輸入)。
- `DiffQty` (INT): 差異量 (`CountQty - SystemQty`)。*Computed Column*.

Table: AdjustmentNotes (庫存調整單)

- `AdjDocNo` (NVARCHAR(50), PK): 調整單號 (e.g., ADJ2025...)。
- `SourceDocNo` (NVARCHAR(50), Nullable): 來源單號 (通常關聯 `StockTakingNotes.DocNo`)。
- `Reason` (NVARCHAR(200)): 調整原因 (e.g., 盤盈、盤虧、報廢)。
- `AdjDate` (DATETIME2): 調整時間。

Table: AdjustmentDetails (調整明細)

- `DetailID` (PK): 流水號。
- `AdjDocNo` (FK): 關聯調整單。
- `SkuID` (FK): 調整品項。
- `AdjQty` (INT): 調整數量 (正數增加庫存，負數減少庫存)。
- `CostAtMoment` (DECIMAL(18, 4)): 成本快照 (用於會計帳紀錄損失或資產增加)。

刪除與停用策略 **(Deletion & Archive Policy):**

- **Constraint:** 針對 `Products` 與 `SKUs` 資料表，系統需實作「交易檢查」。
- **Logic:** 在執行刪除動作前，必須檢查 `StockTransactionLogs` 與 `SODetails` 是否存在該 ID 的紀錄。
    - 無交易紀錄：允許物理刪除 (`DELETE`)。
    - 有交易紀錄：禁止物理刪除 (Throw Exception)，僅允許將 `IsActive` 更新為 `0` (停用/軟刪除)。此舉確保歷史報表與帳務資料的完整性。

---

3. 核心業務邏輯 (Core Business Logic)

本章節定義系統後端 **Service** 層的關鍵演算法與排程邏輯。

3.1 系統自動備份機制 (Auto Backup Mechanism)

資料安全是單機系統的命脈。本模組負責將 SQL Server 資料庫匯出為標準 `.bak` 檔案，確保在硬體故障時可還原。

- Service 組件: `SystemBackupService`
- 配置依賴:
  1. 從 `SystemConfigs` 讀取 `BackupStoragePath` (e.g., `D:/MyERP/Backups/`)。
  2. 若路徑不存在，Service 初始化時需自動建立資料夾 (`Files.createDirectories`)。

A. 定時排程 (Scheduled Job)

- Trigger: `@Scheduled(cron = "0 0 0 1 * ?")` (每月 1 號午夜執行)。
- 命名規則: `ERP_AutoBackup_{yyyyMM}.bak` (例如: `ERP_AutoBackup_202512.bak`)。
- 執行流程:
  1. 組合完整檔案路徑。
  2. 執行 Native SQL:
     ```
     SQL
     BACKUP DATABASE [YourDBName]
     TO DISK =
     'D:/MyERP/Backups/ERP_AutoBackup_202512.bak'
     WITH FORMAT, MEDIANAME = 'ERP_Backup', NAME =
     'Monthly Auto Backup';
     ```
  3. 錯誤處理: 若 SQL 執行失敗 (如權限不足)，需 Catch Exception 並寫入 `Backups` Log 表，狀態標記為 `FAILED`。

B. 啟動檢查 (Boot Check Strategy)

- Trigger: `ApplicationReadyEvent` (當 Spring Boot 啟動完成後)。
- 邏輯:
  1. 取得當前月份字串 (e.g., `2025-12`)。
  2. 掃描 `BackupStoragePath` 目錄。
  3. 比對: 是否存在檔名包含 `202512` (或 `2025-12`) 的檔案。

4. 決策:
    - 不存在 (False): 視為「本月尚未備份」或「錯過排程」，立即觸發上述備份流程。
    - 已存在 (True): 寫入 Log "Backup check passed"，不執行任何動作。
- 目的: 確保即使電腦在 1 號當天沒開機，下次開機時系統也會自動補做備份。

3.2 圖片處理邏輯 (Image Processing Logic)

本模組管理商品圖片的實體儲存與顯示順序，確保前端畫面一致性。

- Service 組件: `ProductImageService`
- 儲存規範 (Storage Policy):
    - 資料庫 (DB): `ProductImages.FilePath` 僅儲存相對路徑與檔名。
        - Format: `/{yyyy}/{MM}/{ProductID}_{Timestamp}.jpg` (e.g., `/2025/12/P001_173478900.jpg`)。
    - 實體儲存 (Physical): 檔案寫入 `SystemConfigs.ImgStoragePath` 加上上述相對路徑的位置。
    - API 輸出: 後端 DTO 回傳圖片 URL 時，需動態讀取 `SystemConfigs.ImgStoragePath` 並與 DB 中的相對路徑組裝。
    - Benefit: 當系統遷移至不同電腦或磁碟機時（例如 D 槽移至 E 槽），僅需修改 `SystemConfigs` 的路徑設定，無須修改資料庫內的每一筆圖片紀錄。
-

A. 主圖顯示規則 (Main Thumbnail)

- 定義: `SortOrder` 數值最小的圖片即為主圖。通常為 `1`。
- API 回應: 在 `ProductListDTO` 中，後端需預先篩選出該 Product 的

主圖 URL 回傳，避免前端下載所有圖片。

B. 刪除與重排序演算法 (Re-indexing Algorithm)

- 情境: 商品有圖片排序 `[1, 2, 3, 4]`。使用者刪除了圖片 1。
- 問題: 剩餘 `[2, 3, 4]`，系統將無 `SortOrder=1` 的圖片。
- 處理流程 (Transaction):
  - 刪除: 刪除 DB 紀錄與實體檔案。
  - 重整: 撈取該 ProductID 剩餘的所有圖片，依 `SortOrder` 升冪排序。
  - 更新: 在記憶體中將 `SortOrder` 重寫為 `1, 2, 3...`。
  - 批次寫入: 執行 Batch Update 更新 DB。
  - 結果*:* 原本的圖片 2 變為 1 (新主圖)，圖片 3 變為 2。

3.3 銷貨價格保護與手續費 (Price Protection & Fees)

本模組確保財務數據的歷史準確性，防止「今日漲價，導致上個月報表獲利虛增」的情況。

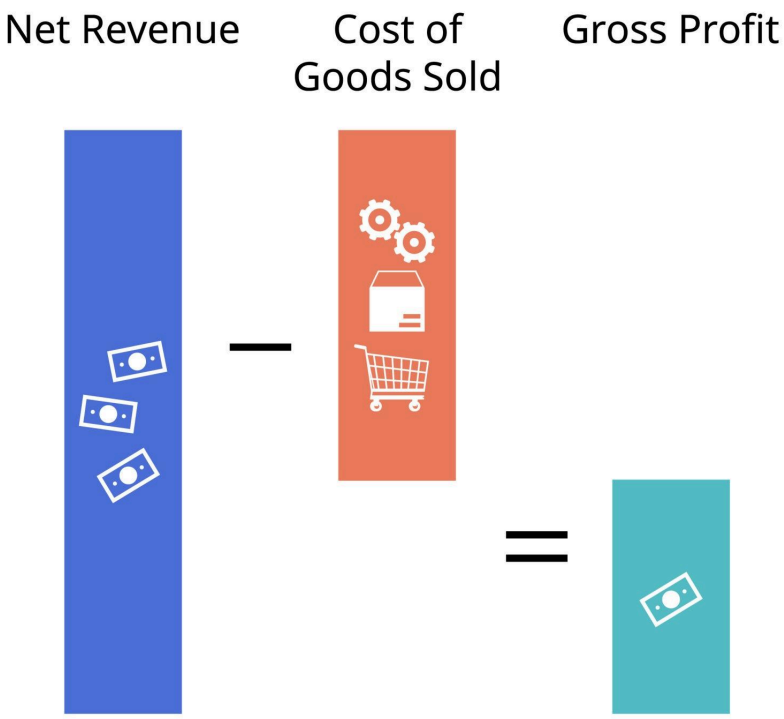
A. 售價快照 (Price Snapshot)

- 時機: `SalesOrder` 狀態由 Draft 轉為 Confirmed (或建立當下)。
- 動作: 將 `Products.BasePrice` 的值 複製 (Copy by Value) 到 `SODetails.UnitPrice`。
- 不可變性: `SODetails` 一旦建立，除非使用者手動修改該張單據，否則絕不隨 `Products` 主檔連動。

B. 手續費計算 (Platform Fee Calculation)

- 公式: `Fee = Round( OrderTotal * Channel.FeeRate )`。
- 互動流程:
  1. 使用者在前端選擇 `SalesChannel` (e.g., "蝦皮")。
  2. 前端呼叫 API 或讀取快取的 `FeeRate` (e.g., 0.05)。
  3. 系統自動計算金額並填入 `SalesOrder.PlatformFee` 欄位。

4. 允許覆寫: 使用者可將自動計算的 **$100** 改為 **$0** (如: 客服特例免運)。
5. 重算: 若使用者修改了訂單內的商品數量, `PlatformFee` 需依據新的總金額重新計算 (若未被手動鎖定)。



3.4 經營報表邏輯 (Business Intelligence Logic)

本模組透過 **SQL** 聚合查詢, 提供經營者決策數據。

A. 庫存價值表 (Inventory Value Report)

- 目的: 了解目前壓在倉庫裡的錢有多少。
  SQL 邏輯:
  SQL
  SELECT
  SUM(Quantity * AvgCost) AS TotalInventoryValue,

```
    COUNT(CASE WHEN Quantity > 0 THEN 1 END) AS
TotalStockItems
FROM Inventory
WHERE Quantity <> 0  -- 通常只計算非零庫存
```

- 注意: 若包含「負庫存」，該數值會扣減總資產（視為對供應商的負債/待補貨成本）。

B. 銷售利潤表 (Sales Profit Report)

- 目的: 計算特定區間的真實淨利 (Net Profit)。
- 輸入: `StartDate`, `EndDate`.
- 計算欄位:
  1. 總營收 (Gross Revenue): `SUM(SODetails.Qty * SODetails.UnitPrice)`
  2. 總銷貨成本 (Gross COGS): `SUM(SODetails.Qty * SODetails.CostAtMoment)`
  3. 退款總額 (Total Refunds):
     - Logic: 透過 `SalesReturns` 關聯 `SODetails` 計算。
     - Formula: `SUM(SalesReturns.Qty * SODetails.UnitPrice)`
  4. 退貨成本迴轉 (Returned COGS):
     - Logic: 商品退回倉庫變回資產，需將原銷貨成本扣除。
     - Formula: `SUM(SalesReturns.Qty * SODetails.CostAtMoment)`
  5. 平台費用 (Fees): `SUM(SalesOrders.PlatformFee)`
  6. 退貨運費損失 (Return Shipping Loss): `SUM(SalesReturns.ReturnShippingFee)`
  7. 淨毛利 (Net Margin):
     - Formula: Net Margin = (Gross Revenue - Total Refunds) - (Gross COGS - Returned COGS) - Fees - Return Shipping Loss
- 過濾條件: 僅計算 `Status` 為 `Confirmed` 或 `Closed` 的單據。

C. 滯銷商品表 (Dead Stock Report)

- 目的: 找出賣不動的貨。
- 輸入: **DaysThreshold** (預設 **90** 天)。
- 邏輯:
    1. 找出目前 **Quantity > 0** 的 **SKU**。
    2. **LEFT JOIN** 最近一次的 **SalesOrders**。
    3. 篩選: **LastSaleDate** 早於 **(Today - DaysThreshold)** 或者 **LastSaleDate IS NULL** (從未賣出過)。
    4. 排序: 依 **Quantity * AvgCost** 降冪排序 (優先處理積壓資金最多的商品)。

3.5 採購退回邏輯 (Purchase Return Logic)

- 庫存與成本:
    - 當採購退回單 `Confirmed` 時，系統執行 `InventoryService.decreaseStock` 。
    - 會計處理: 退貨視為庫存資產減少。減少的庫存價值 = `Qty * Current AvgCost`。
    - 應收帳款: 向供應商請求的退款金額 = `Qty * ReturnPrice` (原進貨價)。
- 防呆: 退貨數量不可大於目前庫存量 。
- 寫入 Log: 寫入 StockTransactionLogs (DocType: 'PO_RET', QtyChange: -Qty, ...)

3.6 盤點與自動調整邏輯 (Stock Taking & Auto-Adjustment)

此流程確保帳實相符，並自動處理差異。

- 流程步驟:
    - 快照 **(Snapshot):** 建立盤點單時，系統鎖定範圍內的 SKU，並將當下 `Inventory.Quantity` 寫入 `SystemQty`。
    - 實盤 **(Counting):** 使用者輸入 `CountQty`。
    - 簽核與觸發 **(Approval & Trigger):**
        - 當使用者點擊「簽核 (Approve)」時，系統計算所有項目的 `DiffQty`。
        - 若 `DiffQty != 0` 的項目存在，前端跳出 **Confirm Dialog**:
          「發現 X 筆庫存差異，系統將自動生成調整單以修正庫存。是否繼續？」
        - **Yes:**
            1. 將盤點單狀態改為 `Approved`。
            2. 自動生成 一張 `AdjustmentNote`。
            3. 將所有差異項目寫入 `AdjustmentDetails` (若 `DiffQty` 為 -2，則調整單 `AdjQty` 為 -2)。
            4. 執行庫存異動 (寫入 `Inventory` 與 `StockTransactionLogs`) 。

成本認定 **(Cost Determination):**

- 盤虧 **(AdjQty < 0):** 視為損失。`Cost = ABS(AdjQty) * Current AvgCost`。
- 盤盈 **(AdjQty > 0):** 視為資產增加。
    - **Logic:** 系統優先使用 `Current AvgCost`。
    - **Zero Cost Handling:** 若 `Current AvgCost == 0` (庫存歸零狀態)，則預設帶入該 SKU 的 `PurchasePrice` (主檔進價)。
    - **Flexibility: UI** 必須開放此欄位 **(Unit Cost)** 供管理者修改，以應對「過季商品重現，價值已非原價」的特殊狀況。
-

3.7 銷貨退回與不換貨政策 **(Sales Return & No Exchange)**

- 嚴格退貨政策**:** 系統不支援直接換貨操作。所有換貨需求一律拆解為兩個動作：
    - 銷貨退回 **(Sales Return):** 建立退貨單，退回商品入庫，退款給客戶。
    - 新銷貨 **(New Sales Order):** 建立新訂單，購買新商品。
- 運費認列**:**
    - 建立 `SalesReturns` 時，系統讀取原單據 `SalesOrders` 的 `ChannelID`。
    - 查詢 `SalesChannels.ReturnShippingFee`。
    - 將此費用記錄於退貨單的 `Expense` 欄位（需新增於 SalesReturns Table），作為當月營業費用，不影響商品庫存成本。
- 目的：當商品退回時，必須以「當初售出的成本」重新計算加權平均成本，而非使用「當前成本」或「當前市價」，以避免資產價值失真。
- 執行步驟 **(Transactional):**
    - 鎖定 **(Lock):** 鎖定該 **SKU** 的 **Inventory** 資料列 **(UPDLOCK)**。
    - 取得原成本**:** 透過 **SalesReturns.OriginalSODetailID** 查詢原銷貨明細 **(SODetails)**，取得該商品售出當下的成本 **CostAtMoment**。
    - 計算新平均成本 **(Recalculate AvgCost):**
        - 公式**:**
          $$NewAvgCost = \frac{(CurrentQty \times CurrentAvgCost) + (ReturnQty \times OriginalCostAtMoment)}{CurrentQty + ReturnQty}$$
        - **Crash Protection** (防呆)**:** 在執行除法前，程式需判斷分母：
            - **IF ((CurrentQty + ReturnQty) <= 0)**
            - **THEN SET NewAvgCost = 0; (**依據歸零邏輯處理**)**
            - **ELSE** 執行上述公式**;**
            - 此判斷防止因負庫存導致的分母為 ***0 (DivideByZero)*** 錯誤。
    - 更新庫存**:**
        - **Inventory.AvgCost = NewAvgCost**
        - **Inventory.Quantity += ReturnQty**
    - 寫入 **Log:** 寫入 **StockTransactionLogs (DocType: 'SO_RET', CostBefore: CurrentAvgCost, CostAfter: NewAvgCost)**。

3.8 進貨入庫邏輯 **(Receiving Process)**

- 情境: 當使用者將 ReceivingNotes 狀態由 Draft 轉為 Confirmed 時觸發。
- 交易原子性: 下列 1~4 步驟須包含在同一個 @Transactional 中。
    - 鎖定 **(Locking):** 依據明細中的 SkuID，鎖定 Inventory 表對應行 (UPDLOCK)。
    - 成本重算 **(Cost Recalculation):**
        - 取得該 SKU 在來源 PO 的實際進貨價 (PODetails.ActualPrice)。
        - 執行 加權平均公式:
          $$NewAvgCost = \frac{(CurrentQty \times CurrentAvgCost) + (RI.Qty \times PO.ActualPrice)}{CurrentQty + RI.Qty}$$
        - 更新 Inventory.AvgCost。
    - 庫存增加 **(Stock Up):**
        - Inventory.Quantity += RIDetails.Qty。
        - 寫入 StockTransactionLogs (DocType: 'PO_IN')。
    - 回寫採購單 **(Update PO):**
        - 更新 PODetails.ReceivedQty += RIDetails.Qty。
        - 自動結案判斷: 檢查該 PO 所有品項，若 SUM(ReceivedQty) >= SUM(OrderQty)，自動將 PO Status 更新為 2: Closed。
- 防呆檢查:
    - 超收警告: 若 RIDetails.Qty > (PO.OrderQty - PO.ReceivedQty)，前端顯示警告「本次進貨量大於採購未交量」，但允許強制執行 (應對廠商多送贈品或溢裝的情況)。

---

4. UI/UX 規範 (User Interface & Experience)

本章節定義前端互動標準，目標是將複雜的 **ERP** 邏輯轉化為直覺的操作體驗，並充分利用 **Vuetify** 元件庫特性。

4.1 SKU 生成器 (SKU Builder)

針對「商品資料建立」頁面，為了避免使用者手動輸入錯誤的格式，**SKU ID** 欄位採「唯讀組合」模式。

- **UI** 元件結構:
    1. **Product:** `v-autocomplete` (搜尋/選擇現有商品)。
    2. **Supplier:** `v-select` (下拉選擇供應商)。
    3. **Color/Size:** `v-combobox` (支援輸入新值或選擇既有值)。
    4. **SKU Preview:** 一個唯讀的 `v-text-field` 或 `v-chip`。
- 互動邏輯:
    1. 當使用者填寫上述四個欄位時，**Vue** `computed` 屬性即時觸發。

2. 格式化: 自動將各欄位值去空白、轉大寫，並以無連接符或特定符號組合。
    - `Display: [PROD01] + [SUP01] + [RED] + [XL]`
    - `Value: PROD01SUP01REDXL`

- 即時重複檢核: 當長度 > 5 時，前端透過 `debounce (500ms)` 呼叫 API `/api/skus/check-exists`。
    1. 若重複*:* 預覽框變紅，並顯示錯誤訊息「此 SKU 組合已存在」。
- 互動優化: 在「建立」按鈕旁加入警語:「SKU 建立後無法修改規格（顏色/尺寸），僅能停用重建，請仔細確認。」

4.2 盤點作業介面 (Stock Taking Dashboard) - [取代原換貨模式]

- 介面設計: 採用 `v-data-table` 呈現盤點清單。
- 欄位: 商品資訊、系統庫存 (唯讀/灰底)、實盤數量 (可編輯 Input)、差異量 (自動計算，若不為 0 顯示紅字)。
- 互動:
    - 快速填入: 提供「全部相符」按鈕，一鍵將 `CountQty` 填為 `SystemQty` (便於僅修改少數差異項)。
    - 差異提示: 當輸入數量與系統不符時，該 Row 背景色變為淡黃色 (Warning)，提醒使用者注意。
- 簽核確認: 點擊「完成盤點」時，若有差異，彈出 SweetAlert2 確認視窗，明確告知將生成調整單。

4.3 條碼與快速輸入 (Input Optimization)

因不支援硬體掃描槍，且 SKU ID 複雜難記，系統必須提供強大的「模糊搜尋 (Fuzzy Search)」與「鍵盤導航」。

- 搜尋元件: 全面使用 `v-autocomplete` 取代標準 `v-select`。
- 搜尋邏輯:
    - 使用者輸入: "紅 裙" (以空白分隔關鍵字)。
    - 前端過濾: 同時比對 `ProductName`, `Color`, `Size`, `SkuID`。
    - 顯示格式: 採兩行顯示 (Line 1: 商品名稱, Line 2: 顏色/尺寸 - 庫存量)。
        - *Example:* 百褶長裙 (P001) 紅色 / M號 | 庫存: 5 | $390
- 鍵盤操作優化:

    - `Tab`: 聚焦搜尋框。
    - `Type`: 輸入關鍵字。
    - `Arrow Down`: 選擇商品。
    - `Enter`: 加入清單並聚焦於「數量」欄位。
    - `Enter (again)`: 確認數量並自動聚焦回搜尋框（準備輸入下一項）。
    - 目標*:* 熟練使用者可完全不使用滑鼠完成一張單據。

4.4 防呆與警告機制 (Error Prevention)

使用 **SweetAlert2** 統一處理所有阻擋與警告，依嚴重程度區分顏色與行為。

A. 負庫存警告 (Soft Block - Warning)

- 情境: 銷貨單 (SO) 按下「確認」時，某品項庫存不足。
- **UI** 行為:
    - 彈出 黃色 (Warning) 對話框。
    - 標題: "庫存不足警告"。
    - 內容: "商品 [P001-RED-M] 目前庫存為 0，此操作將產生負庫存 (-1)。是否繼續？"
    - 按鈕: [取消] (Default Focus) / [強制執行]。

B. 流程阻擋 (Hard Block - Error)

- 情境:
    - 採購退回: 欲退貨數量 > 目前庫存量 (不能退不存在的貨)。
    - 單據簽核: 庫存不足時，禁止「簽核 (Approve)」(若業務邏輯設定簽核不允許負庫存)。
- **UI** 行為:
    - 彈出 紅色 (Error) 對話框。
    - 標題: "操作無法執行"。
    - 內容: "庫存不足，無法執行退回。"
    - 按鈕: 僅有 [確定]。

C. 未存檔攔截 (Navigation Guard)

- 情境: 使用者在「編輯模式」下嘗試切換頁面或關閉分頁。

- **UI 行為:**
  - 觸發 **Vue Router** 的 **`onBeforeRouteLeave`**。
  - 瀏覽器原生/**SweetAlert** 提示: **"**您有未儲存的變更，確定要離開嗎？**"**

銷貨退回介面 **(Sales Return UI)**

- 運費提示: 在建立退貨單時，若該通路設定有 `ReturnShippingFee`，介面應顯示提示：「此通路 (蝦皮) 將產生 $60 逆物流運費成本」。