普元健康管理系統 (HMS) 後端開發

專案執行規劃書

專案名稱	普元健康管理系統 (HMS) 後端開發
專案代號	HMS-BE-2026-01
撰寫日期	2025 年 12 月 25 日
計畫負責人	駱昱辰

普元健康管理系統後端開發

(Health Management System, HMS)

(計畫執行期間：2026.1 至 2026.4)

Prepared by:

駱昱辰

臺中科技大學 i.m.a.c 社群

駱昱辰 (專案經理)

電話：09xx-xxx-xxx

Approved by:

_____(客戶確認)

普元健康科技股份有限公司

王 Aden(產品總監)

電話：(02) 2xxx-xxxx

目　　錄

1. 版次變更記錄

版次	變更項目	變更日期
1.0	第一版	2025.12.25

2. 專案背景與範疇定義 (Project Background and Scope Definition)

2.1 專案背景與商業價值

隨著醫療科技的演進，健康管理已從被動的「疾病治療」轉向主動的「健康促進」與「長期照護」。然而，目前的市場解決方案多數存在數據孤島問題，患者在家測量的血壓、血糖數據難以有效地傳遞給醫療團隊，導致醫師在臨床決策時缺乏長期數據支持。此外，缺乏家屬參與的單機版應用程式，往往導致患者依從性低落，難以維持長期的健康紀錄習慣。

普元 HMS 的開發即是為了回應此一痛點。透過 API-First 的設計策略，本系統將作為一個中樞神經，連接前端行動應用程式、醫療機構的資訊系統以及潛在的第三方健康裝置。商業價值不僅體現在提供穩定的數據儲存服務，更在於透過數據聚合與分析，為醫療機構提供加值服務（如趨勢報表、異常警示），並為患者建立強大的心理支持網絡。

2.2 專案範疇 (Project Scope)

本專案範疇嚴格界定於「後端伺服器系統」的設計、開發與部署。

2.2.1 核心功能範疇 (In-Scope)

依據規格書，本專案將交付五大核心子系統，每個子系統均需實作完整的 RESTful API 介面：

1. **用戶身分管理子系統 (UIM 1.1.0)**：
    - 負責處理使用者帳號的全生命週期管理，包括註冊、登入、驗證、密碼重設與帳號刪

除。
    - 整合 Facebook Graph API 進行 OAuth 第三方登入，降低使用者註冊門檻。
    - 實作基於 JWT (JSON Web Token) 的無狀態身分驗證機制，確保 API 請求的安全性與效能。
    - 符合 NIST 數位身分指引的密碼存儲策略（PBKDF2 或 Bcrypt 雜湊演算法）。
2. **個人資訊管理子系統 (PIM 1.2.0)**：
    - 管理使用者的靜態屬性（如姓名、生日、性別）與動態生理參數（如身高、體重）。
    - 提供個人化的健康閾值設定功能（如收縮壓警示值、飯前血糖目標值），並實作邏輯檢查以防止無效設定（例如最小值大於最大值）。
    - 管理推播通知的偏好設定，允許使用者自定義接收訊息的頻率與類型。
3. **測量上傳子系統 (MUS 1.3.0)**：
    - 接收並驗證時序性健康數據（血壓、血糖、體重、糖化血色素）。
    - 實作關鍵的「醫療安全驗證 (Sanity Check)」邏輯，自動過濾醫學上不可能的數值（如收縮壓 > 300 mmHg 或 < 30 mmHg），確保資料庫內的數據品質。
    - 支援離線數據的補登 (Retroactive Entry)，系統需依據用戶提供的 recorded_at 欄位而非伺服器接收時間來處理時序邏輯。
4. **日記管理子系統 (DMS 1.4.0)**：
    - 提供飲食日記的記錄功能，支援文字描述、多標籤 (Tags) 分類（如「高糖」、「早餐」）。
    - 處理多媒體內容（食物照片），整合物件儲存服務（如 AWS S3），僅在資料庫中儲存資源路徑 (URL)。
    - 提供異質資料的聚合查詢 API，將同一時間區段內的生理數據與飲食紀錄整合回傳，便利前端繪製綜合圖表。
5. **控糖團好友子系統 (FMS 1.5.0)**：
    - 建構社交關係圖譜，支援透過唯一邀請碼 (Invite Code) 建立好友連結。
    - 實作細緻的權限控制 (RBAC)，區分「醫師」、「親友」、「糖友」三種角色的資料存取權限。例如，醫師可查看詳細趨勢報告，而糖友僅能查看激勵性摘要。
    - 整合 Firebase Cloud Messaging (FCM)，在好友數據異常或發布新動態時，主動推播通知至相關聯的使用者裝置。

2.2.2 排除範疇 (Out-of-Scope)

- **前端介面開發**：本專案不包含 iOS 與 Android 行動應用程式的使用者介面設計與程式碼撰寫。
- **硬體韌體開發**：不涉及血糖機、血壓計等硬體裝置的藍牙通訊協定或韌體撰寫。
- **AI 診斷功能**：系統僅提供數據呈現與基礎閾值警示，不包含基於機器學習的自動化診斷或治療建議，以避免觸發「軟體即醫療器材 (SaMD)」的高級別監管要求。

2.3 假設與限制 (Assumptions and Constraints)

- **技術限制**：後端開發語言限定為 Python 3.10.11，網頁框架採用 FastAPI。資料庫在初期開發

階段使用 SQLite 以加速迭代，但在架構設計上必須透過 SQLAlchemy ORM 層保持與 PostgreSQL 的相容性，以便未來遷移。

- **法規依賴**：系統設計必須嚴格遵循台灣《個人資料保護法》及其施行細則，特別是針對特種個資（病歷、醫療、基因、性生活、健康檢查）的蒐集、處理與利用規範。資料保存年限需符合《醫療法》第 70 條至少保存七年之規定。
- **第三方服務依賴**：推播功能高度依賴 Google Firebase 服務，第三方登入依賴 Facebook Graph API。專案需規劃當這些服務不可用時的降級策略。

3. 法規遵循與合規性策略 (Regulatory and Compliance Strategy)

鑑於 HMS 系統涉及敏感的個人健康資訊 (PHI)，合規性不僅是法律責任，更是產品生存的基石。本章節詳細分析台灣相關法規對專案執行的具體要求。

3.1 個人資料保護法 (PDPA) 因應策略

台灣《個人資料保護法》及其 2023 年修正案對個資處理提出了更嚴格的要求，特別是隨著「個人資料保護委員會 (PDPC)」的成立，監管力道將大幅增強。

1. **特種個資的蒐集與同意 (Article 6)**：
   - HMS 蒐集的血壓、血糖數據屬於 PDPA 第 6 條定義的「醫療」、「健康檢查」特種資料。
   - **執行策略**：系統必須在用戶註冊流程中，實作強制性的「書面同意」機制（依《電子簽章法》之電子同意）。同意書內容需明確告知蒐集目的（健康管理）、利用期間（會員存續期間或法規要求期間）、利用對象（使用者本人、授權之醫護人員）及利用方式。
   - **技術實作**：在資料庫 UserAuth 表中記錄同意書版本號與同意時間戳記 (consent_timestamp)，確保每一筆資料的蒐集都有據可查。
2. **資料安全維護措施 (Article 27, 48)**：
   - PDPA 要求非公務機關應訂定「安全維護計畫」，防止個資被竊取或洩漏。修正案將罰鍰上限提高至新台幣 1,500 萬元。
   - **執行策略**：
     - **加密儲存**：所有敏感欄位（如身分證號、病歷數據）在寫入資料庫前，需使用 AES-256 演算法進行加密。
     - **傳輸加密**：全站 API 強制使用 HTTPS (TLS 1.2 或 1.3)，禁止明文 HTTP 連線。
     - **存取控制**：實作最小權限原則 (Least Privilege)，僅有經授權的系統管理員可接觸原始資料庫，且需透過 VPN 或堡壘機存取。
3. **資料外洩通報**：
   - 依據 PDPA 第 12 條，若發生個資外洩，必須以適當方式通知當事人。
   - **執行策略**：建立自動化偵測機制，若發現異常流量或大量資料匯出行為，立即觸發警示。並在後端管理系統中預置「資安通報模組」，以便在事故發生時能迅速向受影響用戶發送 Email 或推播通知。

3.2 醫療法與電子病歷規範

雖然 HMS 主要定位為健康管理工具，但若其數據被用於醫療診斷輔助，則需考量《醫療法》相關規範。

1. **資料保存年限**：
   - 依據《醫療法》第 70 條，病歷資料至少應保存七年；未成年者之病歷，至少應保存至其成年後七年。
   - **執行策略**：系統資料庫設計需支援「邏輯刪除」而非立即的物理刪除。當用戶申請刪除帳號時，系統將標記該帳號為 deleted 狀態，並將資料移轉至冷儲存 (Cold Storage/Archive) 區域保存滿法定年限後，再執行永久銷毀 。
2. **軟體即醫療器材 (SaMD) 判定**：
   - 依據台灣食藥署 (TFDA) 指引，若軟體僅用於「健康促進」、「體重控制」或「單純顯示/儲存數據而不涉及診斷建議」，通常不被列管為醫療器材。
   - **執行策略**：在功能設計與行銷用語上，必須嚴格界定 HMS 為「健康紀錄工具」而非「診斷工具」。所有數據警示需標註「僅供參考，請諮詢專業醫師」，避免落入 Class II/III 醫療器材的監管範疇，這將大幅降低法規合規成本與上市時間。

4. 系統架構與技術規格 (System Architecture and Technical Specifications)

本專案採用現代化的後端技術堆疊，確保系統具備高併發處理能力、可維護性與安全性。

4.1 技術堆疊

- **程式語言**: Python 3.10.11。選用 Python 是因其在資料處理與 AI 擴充性上的優勢，且 3.10 版本引入的 Pattern Matching 有助於簡化複雜的資料驗證邏輯。
- **Web 框架**: FastAPI。相比 Django 或 Flask，FastAPI 基於 Starlette (ASGI) 與 Pydantic，提供極高的執行效能（接近 NodeJS/Go）與自動化的 OpenAPI 文件生成能力，完美契合 API-First 的開發策略。
- **資料庫 (Database)**:
  - *開發環境*: SQLite。用於快速原型開發與單元測試，降低基礎設施依賴。
  - *生產環境*: PostgreSQL。考量到未來數據量的增長與併發寫入需求，生產環境將部署於 AWS RDS 或 GCP Cloud SQL 的 PostgreSQL 實例上。架構上透過 SQLAlchemy ORM 進行抽象化，確保從 SQLite 遷移至 PostgreSQL 的無縫接軌。
- **快取與訊息佇列 (Cache & Message Queue)**: Redis。用於快取高頻存取的 User Profile 資料，以及作為 Celery 或 FastAPI BackgroundTasks 的 Broker，處理發送 Email、推播通知等非同步任務。
- **Web 伺服器**: Uvicorn。作為 ASGI 伺服器，負責處理 HTTP 請求並轉發至 FastAPI 應用程式。
- **容器化**: Docker。確保開發、測試與生產環境的一致性。

4.2 系統架構設計

採用分層式架構 (Layered Architecture)，由外而內分別為：

1. **介面層 (Interface Layer/Routers)**：負責接收 HTTP 請求，定義 URL 路徑（如 /api/user, /api/blood/pressure）。利用 Pydantic Schema 進行請求格式與型別的初步驗證。整合 Dependency Injection (Depends) 進行 JWT 身分驗證。
2. **服務層 (Service Layer/Controllers)**：封裝核心業務邏輯。例如：在寫入血糖數據前，呼叫 Sanity Check 函式驗證數值合理性；在建立好友關係時，檢查邀請碼有效性與雙向關係狀態。此層級不直接操作資料庫，而是呼叫 Repository 層。
3. **資料存取層 (Repository Layer)**：負責與資料庫進行 CRUD 互動。使用 SQLAlchemy 執行 SQL 查詢，將資料庫模型 (Models) 轉換為 Python 物件。
4. **基礎設施層 (Infrastructure Layer)**：包含資料庫連線池 (Connection Pool)、Redis 連線、FCM 推播服務整合、Facebook Graph API 客戶端等。

4.3 資料庫設計與 Schema 規劃

資料庫設計需滿足關聯性完整與查詢效能優化。核心 Table 規劃如下：

- UserAuth: 儲存認證資訊。 id (UUID, PK), email (Unique Index), password_hash, fb_id (Index), created_at.
- UserProfile: 儲存個資與設定。 user_id (FK, PK), invite_code (Unique), birthday, height, weight, settings_json (儲存通知偏好).
- BloodPressure: 儲存血壓數據。 id (PK), user_id (FK), systolic, diastolic, pulse, recorded_at (Index), created_at.
    - *索引優化*：建立 (user_id, recorded_at) 複合索引，以加速「查詢特定用戶某段時間內數據」的 API 回應速度。
- Friendship: 儲存社交關係。 id (PK), requester_id (FK), addressee_id (FK), status (Enum: Pending, Accepted, Rejected), relation_type (Enum: Doctor, Family, Peer).

5. 子系統詳細實作計畫 (Detailed Subsystem Implementation Plan)

本章節針對五大子系統進行深度拆解，定義開發重點與邏輯細節。

5.1 用戶身分管理子系統 (UIM)

- **功能目標**：建立安全的進入門戶。
- **關鍵實作邏輯**：
    - *密碼雜湊*：使用 Passlib 套件搭配 Bcrypt 演算法，設定適當的 rounds 參數（如 12），平衡安全性與驗證效能。嚴禁明文儲存密碼。
    - *JWT 簽發*：Payload 僅包含非敏感資訊（如 User ID, Role, Expiration）。Access Token 效期設定為 30-60 分鐘，Refresh Token 設定為 7-14 天，並實作 Token Blacklist 機制（透過

Redis）以支援強制登出功能。

- *Facebook 登入*：後端接收前端傳來的 access_token 後，必須主動向 Facebook Graph API 發送請求以驗證 Token 有效性並取得 facebook_id。若資料庫無此 ID 則自動註冊，若有則回傳系統 JWT。此流程可防止惡意使用者偽造前端回應進行身分詐欺。

5.2 個人資訊管理子系統 (PIM)

- **功能目標**：維護用戶畫像與健康基準。
- **關鍵實作邏輯**：
  - *邀請碼生成*：在用戶註冊時，利用 UUID 或 Hashids 演算法生成一組 6-8 碼的唯一英數字串作為邀請碼，並確保在資料庫中的唯一性 (Unique Constraint)。
  - *閾值驗證*：在更新健康設定 API (PATCH /api/user/default) 中加入邏輯驗證，例如 systolic_max 必須大於 systolic_min，否則回傳 HTTP 422 錯誤。

5.3 測量上傳子系統 (MUS)

- **功能目標**：確保數據的真實性與可用性。
- **關鍵實作邏輯**：
  - *醫療安全驗證 (Sanity Check)*：在 Service 層實作驗證邏輯。例如：收縮壓 (50-300 mmHg)、舒張壓 (30-200 mmHg)、血糖 (10-600 mg/dL)。若數值超出此範圍，視為誤輸入或異常數據，API 應拒絕寫入並回傳具體錯誤訊息，避免污染統計報表。
  - *時序處理*：所有數據查詢與圖表繪製均依據 recorded_at 欄位。系統需處理「補登」情境，即使用戶今日上傳了一筆上週的數據，系統也能正確將其插入歷史時間軸中。

5.4 日記管理子系統 (DMS)

- **功能目標**：整合生活型態數據。
- **關鍵實作邏輯**：
  - *多媒體處理*：採用「預簽署 URL (Presigned URL)」模式。前端先向後端請求上傳權限，後端回傳一個帶有時效的 S3 上傳連結，前端直接將圖片上傳至雲端儲存，最後再將圖片 Key 或 URL 傳回後端儲存。此舉可大幅降低後端伺服器的頻寬與運算負擔。
  - *聚合查詢*：實作 GET /api/user/diary 介面，後端需同時查詢 BloodPressure, BloodSugar, Weight, DiaryDiet 四張表，將結果依時間排序合併為單一 JSON List 回傳，減少前端發送多次請求的延遲 。

5.5 控糖團好友子系統 (FMS)

- **功能目標**：建立信任的資料分享網絡。
- **關鍵實作邏輯**：
  - *狀態機管理*：好友關係狀態流轉 (0: Pending -> 1: Accepted / 2: Rejected) 需具備嚴格的檢核邏輯。例如，只有 addressee_id 為當前用戶的記錄才能執行「接受」或「拒絕」操作。
  - *醫師權限視圖*：實作專屬的 GET /api/doctor/patient/{id}/report 介面。在執行查詢前，必

須通過雙重權限檢查：(1) 請求者是否為醫師角色？(2) 請求者與目標病患是否存續有效的 Friendship 且關係類型為 Doctor？缺一不可。

6. 工作分解結構 (Work Breakdown Structure, WBS)

本專案採用層級式 WBS 進行任務拆解，確保所有交付物皆被定義與管理。

Level 1: 專案階段

- WP1 : 專案啟動與規劃
- WP2 : 系統分析與設計
- WP3 : 核心功能開發 (Sprint 1-6)
- WP4 : 系統整合與測試
- WP5 : 部署與維運轉移

Level 2 & 3: 詳細任務清單與時程

WP1 專案啟動 (Week 1)

- T1.1 建立專案章程與利害關係人名冊。
- T1.2 確認技術選型與開發環境準備 (GitLab/GitHub, Docker)。
- T1.3 召開 Kick-off Meeting，確認團隊角色（李晏昌、林三志、陳羿廷、駱昱辰）。

WP3 核心功能開發 (Dev Phase) - 採用 Scrum 模式

Sprint	週期	重點交付項目 (Deliverables)	負責人	預估工時
Sprint 1	Week 2-3	**基礎建設 & UIM** - DB Schema 初始化 (Alembic) - API 專案骨架 (FastAPI) - 註冊/登入 API (JWT) - FB OAuth 串接	李晏昌 陳羿廷	80 人時

Sprint 2	Week 4-5	**PIM & MUS (Part 1)** - 用戶資料 CRUD - 健康閾值設定 - 血壓上傳與驗證邏輯 - 體重上傳 API	林三志 駱昱辰	80 人時
Sprint 3	Week 6-7	**MUS (Part 2) & DMS** - 血糖上傳與 Sanity Check - 飲食日記 CRUD - 圖片上傳 (Mock/S3 integration) - 數據聚合查詢 API	林三志 駱昱辰	80 人時
Sprint 4	Week 8-9	**FMS (Part 1)** - 邀請碼生成與查詢 - 發送/接受/拒絕邀請流程 - 好友列表查詢 (含狀態過濾)	陳羿廷 李晏昌	80 人時
Sprint 5	Week 10-11	**FMS (Part 2) & Reporting**	陳羿廷	80 人時

		- 醫師專屬報表 API (RBAC) - FCM 推播通知整合 - 權限與資安加固	李晏昌	
Sprint 6	Week 12	**Buffer & Optimization** - 效能優化 (Redis Cache) - API 文件 (Swagger) 完善 - 遺留 Bug 修復	全體	40 人時

WP4 系統整合與測試 (Week 13)

- T4.1 執行整合測試 (Integration Testing)，驗證各子系統間的資料流。
- T4.2 執行壓力測試 (Load Testing)，模擬 50+ Concurrent Users 情境。
- T4.3 資安弱點掃描 (Vulnerability Scan)。

6.1 衝刺內容詳細說明 (Sprint Details Breakdown)

為了補充上表之交付項目，並確保開發團隊對每一階段的產出有具體共識，以下定義各衝刺週期的詳細範疇與交付標準：

Sprint 1: 基礎建設 & UIM (Week 2-3)

- **範疇說明**：建立 FastAPI 專案結構、配置 Docker 開發環境與 PostgreSQL 資料庫連線。實作 JWT 驗證核心，包含密碼雜湊 (Bcrypt) 與 Facebook OAuth 流程的後端 Callback 處理。
- **關鍵交付物**：
    - 可運作的 API Server (Staging 環境 URL)。
    - /docs 路徑下的 Swagger UI，需包含 Auth (Login/Refresh) 相關介面。

- 資料庫 Schema 遷移腳本 (Alembic) 與 ER Model 初版。

Sprint 2: PIM & MUS Part 1 (Week 4-5)

- **範疇說明**：完成使用者個人資料表 (UserProfile) 的 CRUD。實作血壓 (BloodPressure) 的資料模型，重點在於處理時序邏輯（recorded_at 與 created_at 的區別）與基礎寫入 API。
- **前置依賴**：需等待 Sprint 1 的 UserAuth 表定案後，方可建立 Foreign Key 關聯。
- **關鍵交付物**：
  - 使用者設定 API (包含健康閾值檢查邏輯)。
  - 血壓數據上傳 API (需包含補登邏輯驗證)。
  - Postman 自動化測試集合 (針對 PIM 模組)。

Sprint 3: MUS (Part 2) & DMS (Week 6-7)

- **範疇說明**：擴充測量上傳系統，實作血糖與體重數據的「醫療安全驗證 (Sanity Check)」邏輯。開發日記管理系統 (DMS)，整合 AWS S3 實作圖片上傳（Presigned URL 模式），並開發「異質資料聚合查詢」功能。
- **前置依賴**：需先取得 AWS S3 Bucket 存取權限與設定 CORS。
- **關鍵交付物**：
  - 血糖與體重上傳 API (含異常數值 Error Handling)。
  - 日記 CRUD API (支援多標籤與圖片連結儲存)。
  - 聚合查詢 API：單一 Request 可回傳指定日期的所有生理與日記數據。

Sprint 4: FMS Part 1 - 社交核心 (Week 8-9)

- **範疇說明**：建構控糖團好友系統的底層邏輯。實作唯一邀請碼生成演算法 (Hashids/UUID)，以及好友關係的狀態機流轉 (Pending -> Accepted/Rejected) 與查詢過濾。
- **關鍵交付物**：
  - 邀請碼生成與驗證 API。
  - 好友邀請發送、接受與拒絕 API (需驗證雙向狀態)。
  - 好友列表查詢 API (支援依關係類型篩選)。

Sprint 5: FMS Part 2 - 進階權限與推播 (Week 10-11)

- **範疇說明**：實作細緻的權限控制 (RBAC)，特別是「醫師角色」對病患數據的存取授權檢核。整合 Firebase (FCM) 服務，實作當好友發布動態或數據異常時的推播通知觸發機制。
- **前置依賴**：需完成 Google Firebase 專案設定並取得 Server Key。
- **關鍵交付物**：

- 醫師專屬報表 API (需通過雙重權限驗證測試)。
- FCM 推播整合模組 (提供單元測試證明可觸發通知)。
- 資安加固報告 (針對 API 權限漏洞的自我檢測)。

Sprint 6: Buffer & Optimization (Week 12)

- **範疇說明**：針對高頻查詢 API 導入 Redis 快取機制以提升回應速度。完善所有 API 的 OpenAPI (Swagger) 文件註解。修復前幾個 Sprint 遺留的 Bug，並進行壓力測試優化。
- **關鍵交付物**：
  - Redis 快取實作 (針對 UserProfile 與近期數據查詢)。
  - 完整的 API 規格文件 (Swagger/ReDoc)。
  - 系統效能測試報告 (確認平均回應時間 < 500ms)。
  - 可部署的 Production Docker Image。

7. 資源需求與預算規劃 (Resource Requirements and Budget)

本專案預算編列涵蓋人力成本、雲端基礎設施及軟體授權費用。預算估算期間為 **4 個月**（含專案啟動準備、開發衝刺及上線後首月維運緩衝）。

7.1 人力資源費用明細 (Manpower Cost Breakdown)

依據 2025 年台灣軟體工程師平均薪資水平與專案投入時間（3.5 個月開發 + 0.5 個月維運移轉）估算。

職稱 (Role)	人數	月薪估算 (TWD)	投入工時 (月)	小計 (TWD)	備註
專案經理 / 架構師 (駱昱辰)	1	150,000	3.5	**525,000**	**兼任 Scrum Product Owner (PO)**。 負責架構決策、需求優先序與利害關係人溝通。
資深後端工	1	120,000	3.5	**420,000**	**兼任 Scrum**

程師 (林三志)					**Master**。 負責排除開發障礙、流程引導、核心邏輯與資安審查。
後端工程師 (陳羿廷,李晏昌)	2	90,000	3.5	**630,000**	**擔任 Development Team 成員**。 負責各子系統 API 實作、單元測試與文件撰寫。
人力成本總計	**4**	-	-	**1,575,000**	-

7.2 基礎設施與開發工具費用明細 (Infrastructure & Tools Cost Breakdown)

採用雲端服務 (AWS/GCP) 以降低初期資本支出 (CAPEX)，費用包含開發環境、測試環境 (Staging) 與生產環境 (Production) 預估用量。

類別	項目 (Item)	規格/方案說明	單價/月 (TWD)	期間 (月)	小計 (TWD)
運算資源	AWS EC2 / App Runner	t3.medium x 2 (高可用性配置)	2,800	4	11,200
資料庫	AWS RDS for PostgreSQL	db.t3.medium (含自動備份)	2,400	4	9,600
快取服務	AWS ElastiCache (Redis)	cache.t3.micro (Cluster Mode)	1,600	4	6,400

儲存服務	AWS S3 + CloudFront	圖片儲存與 CDN 流量費	1,000	4	4,000
開發工具	SaaS 訂閱費	GitHub Copilot, Jira, Postman Ent.	3,000	4	12,000
雜項	網域與通訊費	Domain, SSL, SMS 簡訊費, FCM	2,000	4	8,000
基礎設施總計	-	-	**12,800**	**4**	**51,200**

7.3 專案總預算彙整 (Total Budget Summary)

預算項目	金額 (TWD)	佔比	說明
1. 人力資源成本	1,575,000	93.9%	專案主要支出，確保高素質工程團隊投入。
2. 基礎設施與工具	51,200	3.0%	雲端租賃費用，實報實銷。
3. 風險預備金 (Reserves)	50,000	2.9%	用於應對匯率波動或臨時性資源擴充。
專案總預算 (Total)	**1,676,200**	**100%**	(未稅)

8. 風險管理計畫 (Risk Management Plan)

8.1 風險評估表 (Risk Assessment)

本專案採用風險矩陣 (Risk Matrix) 進行評估，針對高風險項目制定緩解措施。

風險項目 (Risk Item)	可能性	影響度	風險等級	緩解與應變措施 (Mitigation Plan)
R1. 法規變動風險 PDPA 修正案導致現行個資處理流程不合規。	中 (3)	極高 (5)	高 (15)	1. 建立專案法規檢核表，定期追蹤 PDPC 公告。 2. 資料庫設計保留「同意書版本」欄位，以便法規變更時強制用戶重新簽署新版條款。
R2. 醫療數據安全驗證失效 使用者輸入異常數值導致數據庫污染。	低 (2)	高 (4)	中 (8)	1. 在 API 層 (Pydantic) 與 DB 層 (Check Constraints) 實作雙重驗證。 2. 建立完整的單元測試覆蓋邊界值 (Boundary Value Analysis)。
R3. 第三方服務中斷 Facebook 登入或 Firebase 推播故障。	中 (3)	中 (3)	中 (9)	1. 系統必須保留本地 Email 登入作為備援機制。 2. 實作推播失敗的 Retry 機制與 Dead Letter Queue，並在 App 內建通知中心，不完全依賴推播。
R4. 專案時程延宕 需求變更或技術難點導致 Sprint 目標未達成。	中 (3)	中 (3)	中 (9)	1. 嚴格執行 Scrum 流程，每日站立會議及時發現阻礙。 2. 預留 Sprint 6 作為 Buffer，不安排新功能開發，專注於整合與修復。

8.2 量化監控與矯正機制 (Quantitative Monitoring and Correction Mechanism)

為了彌補 Daily Scrum 與 Sprint Review 可能產生的溫水煮青蛙效應，本專案參考傳統專案管理的

「矯正基準 (Correction Baseline)」概念，並結合敏捷開發的燃盡圖 (Burn-down Chart) 指標，制定以下三級量化監控機制。此機制旨在當進度發生實質性偏差時，強制觸發特定的管理行動，而非僅止於討論。

監控層級	**監控指標 (Metric)**	**觸發條件 (Trigger / Baseline)**	**強制矯正行動 (Mandatory Action)**
第一級：預警 (Warning)	Sprint 燃盡圖 (Burn-down Chart)	Sprint 第 5 天時，剩餘工作量偏差超過 **10%**。	**啟動「內部調整」**： 1. Scrum Master 介入，移除每日站立會議中的非關鍵障礙。 2. 重新分配團隊內部人力，確保關鍵路徑任務優先完成。
第二級：嚴重 (Critical)	Sprint 燃盡圖 (Burn-down Chart)	Sprint 第 8 天時，剩餘工作量偏差超過 **20%** (矯正基準)。	**啟動「範疇削減 (De-scoping)」**： 1. 立即通知 Product Owner (PO)。 2. 將此 Sprint 中優先級最低的 User Story 移回 Product Backlog。 3. 確保本次 Sprint 的核心目標 (Sprint Goal) 仍能達成。
第三級：緊急 (Emergency)	Sprint 目標達成率 (Sprint Goal Success)	**連續 2 個 Sprint** 未達成目標，或 Bug 累積數量導致 Velocity 下降 30% 以上。	**啟動「技術債償還」機制**： 1. **暫停新功能開發**：下一個 Sprint 強制轉為「技術債

			Sprint (Technical Debt Sprint)」。 2. **專注修復**：全體人力投入解決架構問題、重構程式碼或解決測試瓶頸。 3. **根本原因分析**：召開特別回顧會議，在解決阻礙前不進入新的功能開發週期。

9. 建構管理計畫 (Configuration Management Plan)

本專案參考 CMMI/ISO 規範之建構管理精神，但針對敏捷開發與雲端協作特性進行輕量化調整，重點在於確保「文件」與「程式碼」在開發過程中的一致性與可追溯性。

9.1 建構項目與版本控管 (Configuration Items & Version Control)

本專案將建構項目 (Configuration Items, CI) 區分為兩大類，並採取差異化的管理策略：

- **文件類 (Type A - Documents)**：包含專案執行規劃書、軟體需求規格書 (SRS)、系統架構設計圖。
  - **存放位置**：雲端協作平台 (如 Google Drive 或 Confluence)。
  - **版控策略**：
    - 所有關鍵文件必須標註版本號 (如 v1.0, v1.1)。
    - **重大里程碑 (Milestone)** 或 Sprint 結束時，需建立「快照 (Snapshot)」或另存 PDF 歸檔，嚴禁直接覆蓋舊版而不留紀錄。
    - 變更歷程需記錄：日期、變更者、變更摘要。
- **程式碼類 (Type B - Code)**：包含後端原始碼、Dockerfile、IaC 設定檔、資料庫遷移腳本 (Alembic)。
  - **存放位置**：GitHub (Private Repository)。
  - **版控策略**：
    - 採用 **Git Flow** 或 **GitHub Flow** 分支策略。
    - 主分支 (Main/Master) 必須隨時保持可部署狀態 (Deployable)。
    - 所有程式碼合併 (Merge) 前必須通過 Pull Request (PR) 審查與 CI 自動化測試。

9.2 基準線管理 (Baseline Management)

為確保開發進度可被追蹤且在災難發生時可還原，本專案設定「Sprint 結束點」為建立基準線的時刻。

- **基準線定義 (Baseline Definition)**：一個完整的專案基準線 = **[GitHub 上的特定 Commit Tag]** + **[對應版本的規格文件]**。
- **執行時機**：每次 Sprint Review 驗收通過後。
- **操作方式**：
  1. **程式碼標記**：在 GitHub 上對 main 分支打上 Tag（例如 v1.0.0-sprint1）。
  2. **文件標記**：在規格書或規劃書的修訂紀錄中，明確標註該文件版本對應的 Git Tag。
- **回滾機制 (Rollback)**：若新功能上線後導致系統崩潰或出現嚴重邏輯錯誤，團隊可依據最近一次的基準線，將程式碼與環境設定迅速回溯至上一個穩定的 Sprint 版本，確保服務可用性，並避免文件與程式碼脫鉤。

9.3 變更控制流程 (Change Control Process)

雖然本專案採用 Scrum 敏捷開發，擁抱需求變更，但為避免「範圍蔓延 (Scope Creep)」影響專案承諾與品質，本專案執行嚴格的變更控制紀律。

- **核心原則**：「所有新增需求必須放入 Product Backlog，由 PO (專案經理) 評估優先級，排入**下一個 Sprint**，原則上不可干擾進行中的 Sprint。」
- **變更處理程序**：
  1. **提出變更 (Request)**：所有新增功能或規格異動，需在 Jira/Issue Tracking System 中建立 Ticket，禁止口頭承諾。
  2. **評估 (Evaluation)**：由 PO 評估該需求的商業價值與急迫性，並由技術團隊估算 Story Points。
  3. **排程 (Scheduling)**：
     - **一般變更**：放入 Product Backlog，待下一次 Sprint Planning 會議時決定是否納入。
     - **緊急變更 (Emergency)**：若遇法規強制要求或嚴重阻礙商業運作之變更，需經 PO 核准後方可插單 (Interrupt)，但必須遵守「一進一出 (One-in, One-out)」原則，即移入一個新工作項目的同時，必須移出同等 Story Points 的低優先級工作，以維持團隊的開發速率 (Velocity) 穩定。

10. 溝通管理計畫 (Communication Management Plan)

鑑於小規模敏捷團隊高度依賴成員間的緊密協作，為防止溝通斷層或成員無預警失聯導致專案停擺，本計畫制定明確的溝通頻率與異常通報機制。

10.1 溝通與會議機制 (Communication & Meeting Matrix)

會議/活動類型	頻率/時間	形式	參與人員	目標與產出

Sprint Planning	每雙週週一 10:00	實體/線上	全體成員	決定本次 Sprint Backlog，估算 Story Points。
Daily Standup	每日 10:00 (15mins)	線上 (Discord/Meet)	全體成員	回報昨日進度、今日計畫、遭遇困難 (Blockers)。
Sprint Review	每雙週週五 14:00	實體/線上	全體成員 (+ 利害關係人)	展示可運作軟體 (Demo)，收集回饋。
Sprint Retrospective	每雙週週五 15:00	實體/線上	全體成員	檢討流程問題 (Keep/Drop/Try)，優化協作模式。
異常狀況通報	事件發生當下	通訊軟體 (Line/ Discord)	相關人員	若因病、事假或技術卡關無法完成 Task，**需提前 24 小時**通報。

10.2 成員參與監控與異動管理 (Member Engagement & Change Management)

為確保專案人力穩定，針對成員參與狀況執行以下規範：

- **異常通報守則**：成員若預期無法按時交付任務，必須在 Deadline 前 **24 小時** 於群組主動通報，並提出補救方案（如：週末趕工或請求支援），**嚴禁「時間到才說做不完」**。
- **失聯處理機制**：若成員無故缺席 Daily Standup 連續 **2 次** 或訊息未讀未回超過 **24 小時**，Scrum Master 將啟動緊急聯絡程序，並評估是否重新分配其任務。
- **成員退出預告**：參考專案管理標準，若成員因故需退出專案，需至少提前 **2 週 (一個 Sprint)** 提出，並有義務完成交接文件與程式碼解說，確保知識傳承不中斷。

11. 品質保證與測試策略 (Quality Assurance and Testing Strategy)

為確保系統達到醫療級的可靠性，本專案將執行嚴格的品質保證流程。

11.1 測試層級 (Testing Levels)

- **單元測試 (Unit Testing)**：
  - 目標：驗證個別函式與方法的邏輯正確性。
  - 工具：Pytest。
  - 標準：針對核心商業邏輯（如密碼雜湊、醫療數值檢核、權限判斷），程式碼覆蓋率 (Code Coverage) 需達 80% 以上。
- **整合測試 (Integration Testing)**：
  - 目標：驗證 API Endpoint 與資料庫、Redis 及外部服務的互動。
  - 工具：Pytest + TestClient (FastAPI 內建)。
  - 策略：使用 Docker Container 建立獨立的測試資料庫，確保測試環境資料隔離，測試後自動回滾 (Rollback)。
- **負載測試 (Load Testing)**：
  - 目標：驗證系統在高併發下的穩定性。
  - 工具：Locust 或 k6。
  - 標準：在 50 個併發使用者 (Concurrent Users) 下，API 平均回應時間需小於 500ms，錯誤率低於 1%。

11.2 完成定義 (Definition of Done, DoD)

每個工作項目 (Task/User Story) 在標示為「完成」前，必須滿足以下條件：

1. [ ] 程式碼已通過 Flake8/Black 靜態分析檢查。
2. [ ] 相關單元測試與整合測試已通過。
3. [ ] API 文件 (Swagger/OpenAPI) 已更新並與程式碼一致。
4. [ ] 已通過至少一位其他開發者的程式碼審查 (Peer Review)。
5. [ ] 針對 PDPA 敏感資料處理邏輯已再次確認合規性。

11.3 雙層驗收標準 (Two-Tier Acceptance Criteria)

為確保開發產出能轉化為實際可用的成果，本專案實施雙層驗收機制，區分工程交付與用戶驗收的責任邊界。

Tier 1: 內部工程驗收 (Internal Engineering Review)

- **驗收對象**：開發人員提交給審查人員 (Reviewer/Tech Lead)。
- **執行時機**：Merge Request (MR/PR) 發起時。
- **驗收標準**：
  - 完全符合上述 **11.2 Definition of Done (DoD)** 之規範。
  - CI Pipeline (Unit Test, Linting) 全數通過 (Green)。
- **產出結果**：程式碼被核准合併至 develop 或 main 分支，Jira Ticket 狀態更新為 **"Resolved"**。

Tier 2: 外部用戶驗收 (User Acceptance Testing, UAT)

- **驗收對象**：產品負責人 (PO) 或 模擬用戶 (Simulated User/Tester)。
- **執行時機**：功能部署至 Staging 環境後，或於 Sprint Review 會議前 24 小時內。
- **執行方式**：由非開發人員存取 Staging 環境之 **Swagger UI (OpenAPI)** 網頁介面，或使用預先定義之 **Postman Collection**，依照 User Story 的驗收條件進行實際操作。
- **驗收標準**：
  - **功能正確性**：快樂路徑 (Happy Path) 與異常路徑 (Unhappy Path) 皆符合預期，JSON 回傳格式正確。
  - **可用性**：API 回應時間符合 SLA，且錯誤訊息需清晰易懂（對人類友善，而非噴出 Traceback）。
  - **無阻礙性缺陷**：確認無 Critical 或 Major 等級的 Bug 遺留。
- **產出結果**：Jira Ticket 狀態由 "Resolved" 更新為 **"Closed"**，該功能正式視為「已交付」。若驗收失敗，則退回並計入下一輪 Sprint 的修復工作。