工作目標報告
員工姓名：　胡智強 Johnny
部門：　研發一部
主管：　謝仁豪 Ren
日期： 01/06

1 月預定完成事項

目標一　(1/6~1/9)

具體內容： 轉錄新增上傳音檔功能

達成標準:

- 上傳音檔功能撰寫
- 針對 ios 以及 Android 特殊格式音檔做轉檔調整
- 針對 M4A、3GP、MOV、MP4 視訊格式做音訊擷取
- 討論上傳檔案大小限制
- 討論架構是否遷移至後端

目標二　(1/13~1/16)

具體內容： 新增 SciPlanGPT group chat 上傳文檔功能

達成標準:

- 上傳文檔功能撰寫
- 討論上傳各式限制（.pdf, .docx, .xlsx, .jpg）
- 討論上傳檔案大小限制
- 分塊上傳（Chunk Upload）
- 討論上傳檔案大小限制
- 討論架構是否遷移至後端
- 文檔串接至 Leo 之 create embedding function

2 月預定完成事項

目標一 (2/3~2/21)

具體內容： 系統架構重新設計

達成標準:

- ✓ 需求分析
  - **收集需求**：確保了解系統的新需求，包括功能需求、性能需求、安全性需求等。
  - **確認現有問題**：總結現有系統的瓶頸，例如性能不足、可擴展性差、安全問題或技術老化。

- 系統目標設定

- **性能提升**：縮短響應時間或提高處理能力。
- **可擴展性**：支持更多用戶或數據。
- **穩定性和容錯**：提高系統可用性。

- 現有技術分析

  技術審核：分析現有系統的架構、模組、代碼質量、數據庫設計和部署方式。

  性能分析：利用工具（如 APM 或監控系統）檢測性能瓶頸。

  依賴檢查：列出所有使用的技術框架及其版本，檢查是否需要更新或替換。

- 設計新架構
  - 單體架構（Monolithic）
  - 微服務架構（Microservices）
  - 事件驅動架構（Event-driven）
  - 無伺服器架構（Serverless）
  - 混合架構
- 原型設計

  開發一個最小可行產品（MVP）或系統原型，用於驗證新架構的設計是否滿足需求。

- 部署與監控

目標二（2/3~2/21）

具體內容：整合 Ren 設計之前端頁面

達成標準

- **更換新前端架構**：
  - 根據設計和功能需求分解開發任務，確保每個模塊的開發工作有明確的範疇與時間進度。
  - 確保每個模塊都有清晰的界面功能定義，包括交互邏輯、輸入輸出數據格式，以及與後端 API 的依賴關係。
- **設置開發環境**：
  - 確保本地開發環境配置完成，並支持快速構建、測試和部署流程：
    - 使用版本控制系統（如 Git）管理代碼。

        - 配置前端開發框架（如 React 或 Vue.js）與構建工具（如 Vite 或 Webpack）。
        - 配置自動化測試工具與開發服務器。
- **接口測試**：
    - 測試現有後端 API 是否符合新前端技術的需求，並確保：
        - API 輸出結果符合新界面的數據要求。
        - 接口響應時間和性能滿足用戶需求。
- **測試與驗證**：
    - **單元測試（Unit Testing）**：
        - 為每個前端模塊撰寫單元測試代碼，確保獨立模塊功能穩定。
    - **接口測試**：
        - 使用工具（如 Postman 或 Swagger）驗證前後端交互，確保數據傳遞準確無誤。
    - **功能測試**：
        - 模擬用戶行為，測試界面各功能的完整性與穩定性。
- **響應式設計**：
    - 確保新界面支持多設備（桌面端、移動端和平板），使用 CSS Media Query 和響應式框架（如 Bootstrap 或 Tailwind）進行設計與實現。
- **串接上傳文檔新頁面與 Leo 的 create embedding function**：
    - 開發文檔上傳頁面，並確保頁面與後端的 create embedding function 成功串接。
    - 確保上傳文檔後的流程執行流暢，包括：
        - 文檔檢查與處理。
        - 調用 create embedding function 返回結果。
        - 在界面上顯示處理進度與結果。

目標三（2/21~3/14)

具體內容：更換現有 firebase 架構，實現 Redis 與 NAS 整合架構

達成標準

- **確定需求與架構設計**
    - **確定使用場景**：

- Redis：
  - 存儲熱數據（如用戶會話、即時查詢結果）。
  - 作為 NAS 數據的緩存層，減少高頻請求對 NAS 的壓力。
- NAS：
  - 存儲冷數據（如文檔、媒體文件）及長期存儲需求。

- **設計數據分層策略**：
  - 定義數據的「熱」和「冷」標準（例如最近 X 小時內訪問的為熱數據）。
  - 設置數據過期機制（TTL），自動將不常用數據從 Redis 移動到 NAS。
- **架構草圖**：
  - 客戶端訪問時，首先查詢 Redis。
  - 若 Redis 未命中（Cache Miss），從 NAS 獲取數據，並回寫到 Redis。

- **設置 Redis 與 NAS 環境**
  - **Redis 部署**：
    - 安裝 Redis 並設置高可用性（如 Redis Sentinel 或 Redis Cluster）。
    - 配置持久化（AOF 或 RDB），確保重啟後數據不丟失。

  - **NAS 部署**：
    - 配置 NAS 系統（如 Synology、QNAP、AWS EFS 或其他）。
    - 設置用戶訪問權限和安全策略，確保數據安全性。
    - **選擇中間層工具**：如果需要進一步簡化開發，可以考慮使用專門的緩存同步工具（如 RedisGears、Nginx Cache Module）。

- **開發整合邏輯**
  - **Redis 作為緩存層**
- **查詢流程**：
  - 用戶請求數據，首先查詢 Redis。
  - 如果命中，直接返回 Redis 中的數據。
  - 如果未命中：

        - 從 NAS 提取數據。
        - 回寫到 Redis，設置適當的 TTL（如 1 小時）。
- **開發要點**：
    - 使用 Redis 的數據結構（如 Hash、String、List）存儲元數據或小文件。
    - 使用唯一鍵名規範，如 file:{file_id}。
        - **將冷數據移動到 NAS**
- **定期清理策略**：
    - 使用 Redis 的 LRU 機制清除過期數據。
    - 開發定時任務（如使用 Python + Celery）將不常訪問的數據從 Redis 移動到 NAS。
- **實現數據存儲**：
    - Redis 中僅保存數據的索引或小文件。
    - 大文件存儲在 NAS，並記錄文件位置。
- **測試整合**

    - **模擬負載測試**：

        - 使用測試工具模擬高頻數據訪問，觀察 Redis 和 NAS 的性能。

    - **數據一致性測試**：

        - 測試 Redis 與 NAS 中數據同步是否正常。

    - **邊界情況測試**：

        - 測試 Redis 緩存過期後是否正確從 NAS 獲取數據。
        - 測試 NAS 不可用時系統是否能正確回應。
- **部署與監控**

    - **正式部署**：

        - 在生產環境部署 Redis 和 NAS，根據負載量設置不同節點（如多台 Redis 和 NAS 分片）。
        - 配置緩存與存儲的分佈式架構。

- **監控設置**：
    - **Redis**：

        - 使用 Redis 自帶的監控工具或 RedisInsight 查看內存使用、請求量、緩存命中率等。
    - NAS:
        - 配置 NAS 的容量告警、訪問日誌和數據完整性檢查。
- **性能優化**:
    - 增加 Redis 的內存分配或啟用分片以提高命中率。
    - 對高頻數據查詢使用 Pipeline 或 Lua Script。