英文單字查詢工具

軟體開發文件

組別：第5組

成員：112270042李能安、113170020林睿傳

113170526郭思延、113171114楊景翔

113170134董奕辰、112221476譚培成

指導老師：賴素純 老師

中華民國113年11月

**目錄**

[1. 專案簡介 2](#_Toc180851433)

[2. 開發環境 2](#_Toc180851434)

[3. 系統開發流程 2](#_Toc180851435)

[4. 程式設計主要技巧 4](#_Toc180851436)

[5. 原始碼重點解說 6](#_Toc180851437)

[6. 套件使用 7](#_Toc180851438)

[7. 系統文件結構概述 7](#_Toc180851439)

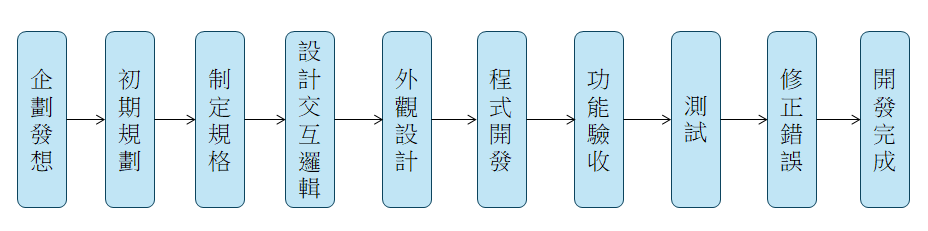
1. **專案簡介**

* **專案名稱**：英文單字查詢工具
* **版本**：1.0
* **專案目標**：建立一個易於使用的應用程式，讓用戶可以快速查詢英文單字，並與 OpenAI API 整合，提供更深入的語言理解。

1. **開發環境**

* IDE: PyCharm
* Version: Python 3.12

1. **系統開發流程**



1. **規劃發想**

這是開發的第一步，旨在確定應用的主要目標和功能。我們集思廣益，提出了多種功能想法，並根據用戶需求選定了最優解。這個過程中我們還需要考慮應用的使用場景和潛在用戶。

1. **初期規劃**

在此階段，我們制定了項目的整體計劃，包含主要的時間線、開發分工及資源分配等。同時確立了技術棧（如選擇 Python 和 tkinter），並評估應用程式的技術可行性。

1. **制定規格**

根據前期的發想與規劃，詳細列出各項功能需求和技術規格，例如使用者界面應具備哪些功能按鈕，應如何與 OpenAI 的 API 進行互動等。這些規格文檔為後續開發提供了明確的參考依據。

1. **設計互動邏輯**

設計互動邏輯是為了確保使用者操作時，應用程式能夠正確處理並回應。例如，當使用者輸入查詢字詞時，應用程式如何將查詢結果從 OpenAI 獲取並展示給使用者。我們針對各種使用情境進行了流程圖的繪製，確保使用者體驗流暢無誤。

1. **外觀設計**

使用 tkinter 套件設計應用的外觀，包括查詢框、按鈕、結果顯示區域等，旨在讓使用者界面簡單直觀，方便所有年齡層的用戶使用。色彩搭配和佈局設計也在此階段進行，以提升視覺效果和使用者的舒適度。

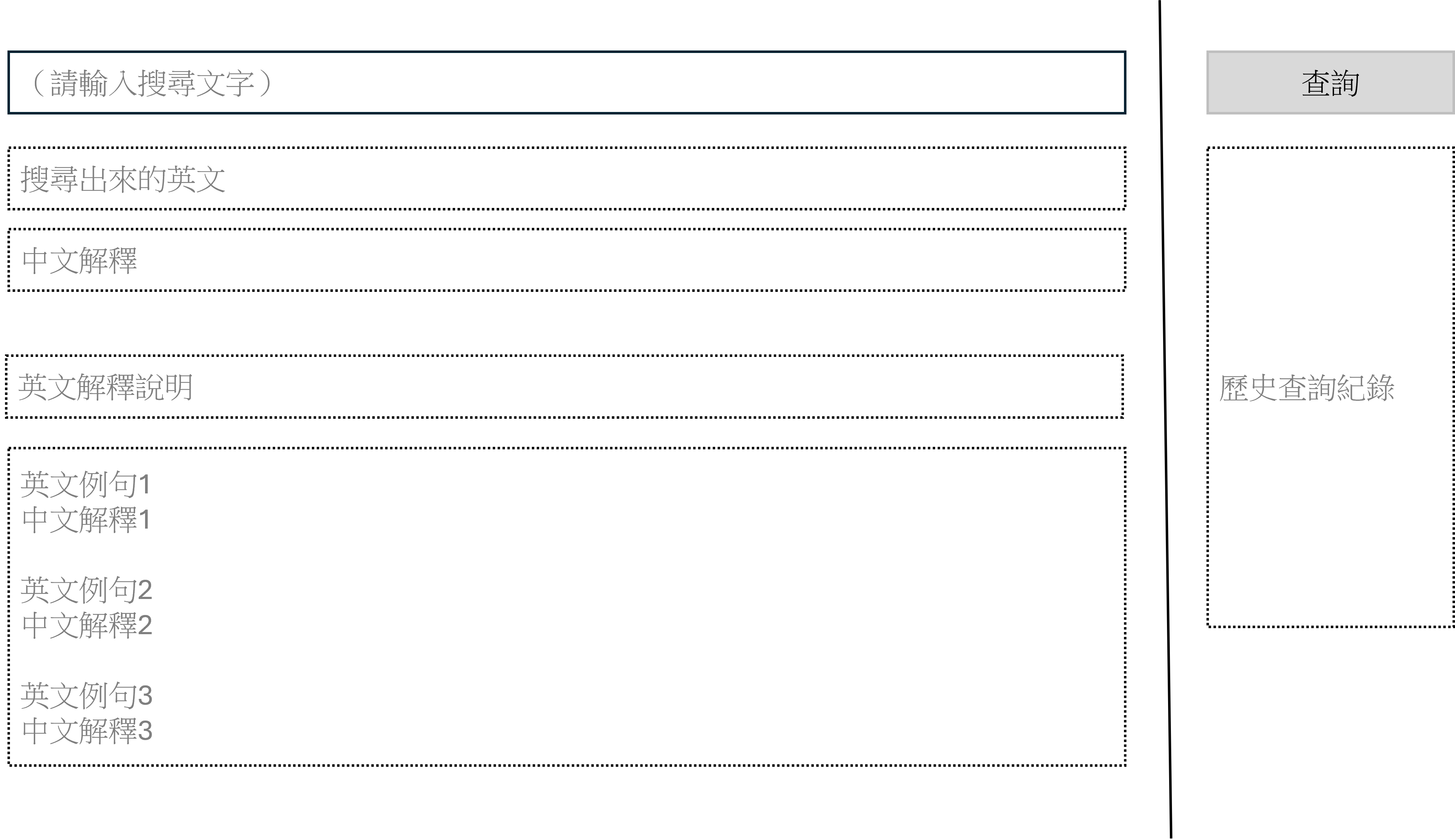


圖1使用者介面設計圖

1. **程式開發**

程式開發是根據前面的需求與設計文件，將每個模組逐步實作出來。這裡分別進行資料模組（Model）、視圖模組（View）及邏輯處理（ViewModel）的實作，同時確保模組間的協作運作正常。

1. **功能驗收**

每個功能完成後，我們進行功能驗收，以確保各個模組達到預期的效果。例如，確認查詢結果顯示的正確性，歷史記錄功能是否正常儲存使用者查詢等。

1. **測試**

在開發完成後，進行全面的測試，包括單元測試和整合測試。單元測試確保每個模組獨立運作正常，而整合測試則確保各模組之間協作順暢。用戶體驗測試也是其中的重要一環，以確認應用程式符合使用者的需求。

1. **修正錯誤**

測試過程中發現的錯誤會立即進行修正，並重新測試以確保問題解決。我們採用迭代式的方法持續修正問題，直到系統穩定。

1. **開發完成**

完成所有功能的開發和測試，確保系統的每個部分都能穩定運行後，我們進行最終的驗收，並準備部署。

1. **程式設計主要技巧**
2. **外觀部分使用 tkinter 套件：**

tkinter 是 Python 內建的 GUI 工具包，用來設計使用者界面。透過 tkinter，我們能快速構建應用程式的視覺部分，例如輸入框、按鈕及查詢結果顯示區域。由於 tkinter 是內建的，所以它具有高度兼容性，適合快速建立和測試 GUI。

1. **資料部分使用 MVVM 架構：**
   1. **MVVM（Model-View-ViewModel）架構**

將應用程式分為三個主要部分，以提高可維護性和可擴充性。這種架構的主要目的是將應用程式的邏輯與視圖分離，使代碼更加易於管理和測試。

* + 1. **Model（模型）：**

Model 層負責管理應用程式的數據和商業邏輯。它與應用的資料來源互動，例如從 API 獲取數據，並將數據處理後提供給 ViewModel。這樣的設計確保了數據與應用的顯示分離，方便數據邏輯的獨立測試和修改。

* + 1. **View（視圖）：**

View 是應用程式的使用者界面，負責顯示資料並與使用者互動。它接收來自 ViewModel 的數據並進行展示，所有與界面的互動（例如按鈕點擊、資料輸入）都會由 View 來處理，並傳送給 ViewModel。這確保了界面的變更不會影響底層邏輯。

* + 1. **ViewModel（視圖模型）：**

ViewModel 作為 View 與 Model 之間的橋樑，負責處理來自 Model 的資料並進行格式化，以便 View 能夠顯示。它還處理使用者的輸入，並將相應的變更回饋給 Model。ViewModel 通常會透過資料綁定與 View 進行互動，這意味著當 Model 中的數據變動時，View 也會自動更新，實現了更好的動態互動。

* 1. **View 和 ViewModel 使用 CallBack 來綁定資料變動事件**

這種綁定方式確保了在資料變更時，界面可以即時更新，用戶的操作也能及時反饋到數據層。

1. **原始碼重點解說**
2. **MainView.py & MyEntry.py**

MainView.py 和 MyEntry.py 主要負責外觀部分的設計，使用 tkinter 套件構建輸入字段、查詢按鈕、查詢結果顯示和歷史記錄等四個主要部分。這些模組的設計確保了用戶能夠直觀地輸入查詢關鍵字，查看查詢結果，以及查看歷史記錄。 MyEntry.py 可能包含自定義的輸入欄位，使得用戶體驗更加流暢。

1. **OpenAiSentencesApi.py**

這個模組負責應用程式與 OpenAI 的接口，管理與 API 的互動。它負責向 OpenAI 發送請求並獲取結果，並且處理可能的網絡錯誤與延遲情況，確保應用程式能可靠地從 OpenAI 獲取數據。

1. **OpenAiSentencesApiResponseModel.py**

這個模組專注於處理從 OpenAI API 獲得的 JSON response，使用反序列化（Deserialization）將資料轉化為 Python 可操作的物件。這樣的處理方式使得後續的資料操作更加直觀和便捷，減少了處理原始 JSON 結構的複雜性。

1. **main.py**

main.py 是應用程式的進入點，負責整合各個模組，並執行應用程式。當使用者啟動應用時，main.py 將載入主視圖，並初始化所有必須的物件和模組，以確保應用程式能夠正常運行。

1. **ApiKey.py**

ApiKey.py 主要存放 OpenAI 的 API 密鑰，並確保其安全性。在開發過程中，我們採取了措施以避免將密鑰直接硬編碼到主代碼中，這樣的設計使得密鑰管理更加靈活，且提高了系統的安全性。

1. **套件使用**

* **tkinter：** Python 內置的 UI 套件，不需要額外安裝，用於構建應用程式的用戶界面。這個套件簡單且功能強大，適合進行快速的 UI 開發。
* **dataclass\_wizard：** 用於 JSON 物件的解析和處理，尤其在處理 API 返回的複雜結構時，可以將其轉化為 Python 的資料類別，便於操作和管理。
* **requests：** 用於發送 HTTP 請求，以與 OpenAI 的 API 互動。requests 是一個簡單而強大的 HTTP 請求工具，可以輕鬆發送 GET 和 POST 請求，並處理返回的資料。
* **其他依賴套件：** 根據應用的需要，還安裝了一些額外的依賴套件，以確保所有功能模組的正常運作。

這些套件及技術組合能夠提供穩定的用戶界面與後端數據交互，確保應用程式具備良好的用戶體驗和易於維護的代碼架構。

1. **系統文件結構概述**
2. **功能需求規劃**

我們首先確定應用程式的主要功能，包括單字查詢、歷史記錄管理、與 OpenAI API 的交互等。通過多次會議，我們列出了應用程式的核心功能需求，並確保這些需求能夠滿足使用者的主要使用場景。

1. **系統架構設計**

使用 MVVM 架構來劃分系統的各個模組，這樣可以有效分離資料層與顯示層，並確保各部分之間的責任單一且清晰。這樣的架構使得代碼的可維護性大大提升，尤其在後期功能擴充和修改時，能夠保持高效和簡便。

1. **視覺與互動設計**

視覺設計部分主要通過 tkinter 來完成。我們確保每一個 UI 元素都符合直觀性和易用性的原則，例如查詢按鈕的大小和位置、結果顯示的清晰度等，這些都是經過多次測試和優化後確定的。此外，我們還考慮了使用者的不同需求，增加了歷史記錄功能，以方便用戶快速重新查詢之前的結果。

1. **測試與驗收**

在測試過程中，我們進行了多層次的測試，包括功能測試、用戶體驗測試以及性能測試。功能測試確保應用的每個部分按預期工作，用戶體驗測試則重點在於使用者操作的便捷性和整體使用感受。而性能測試主要是檢測應用在不同網絡環境下的表現，特別是在與 OpenAI 交互時的響應速度。

1. **部署與維護**

在應用完成開發和測試之後，我們進行了部署。部署過程中我們選擇了一個穩定的 Python 運行環境，並對可能的依賴問題進行了測試和解決。未來的維護計劃包括持續的功能更新和錯誤修復，並根據使用者的反饋對應用進行改進和優化。