



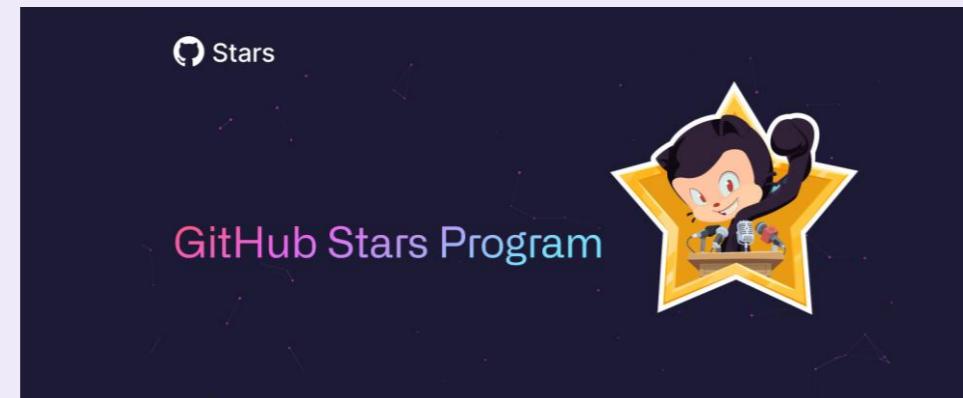
STUDY4
為 學 習 而 生

SPECIAL THANKS

91APP 基金會

önelab
瑞嘉軟體科技

新加坡商 鈦坦科技
TITANSOFT



STUDY4

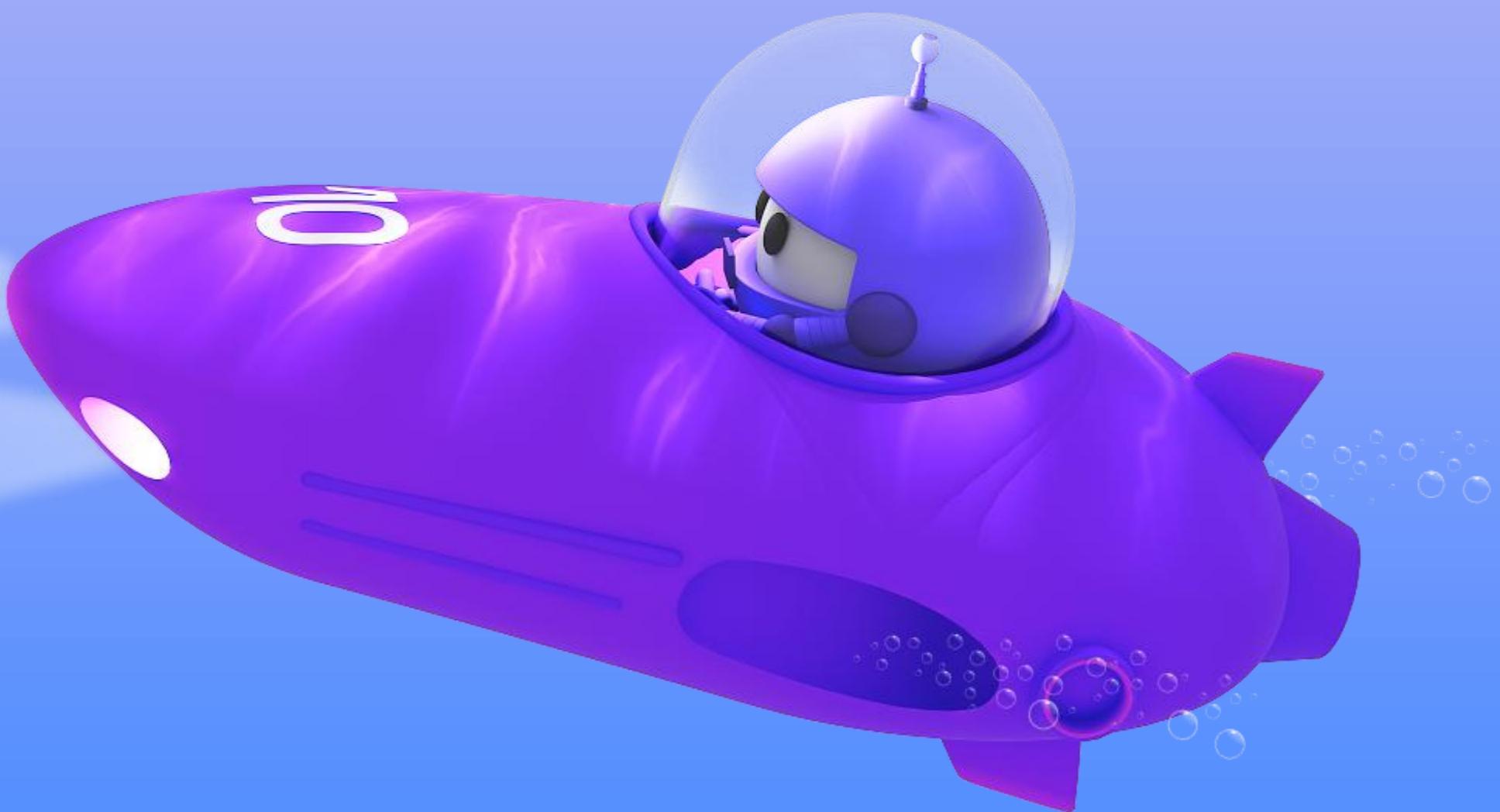
為 學 習 而 生

and all of you



碼農出擊！ GitHub Copilot 帶你飛 打造超效率開發術

Tim Lai | 20251129





Tim Lai

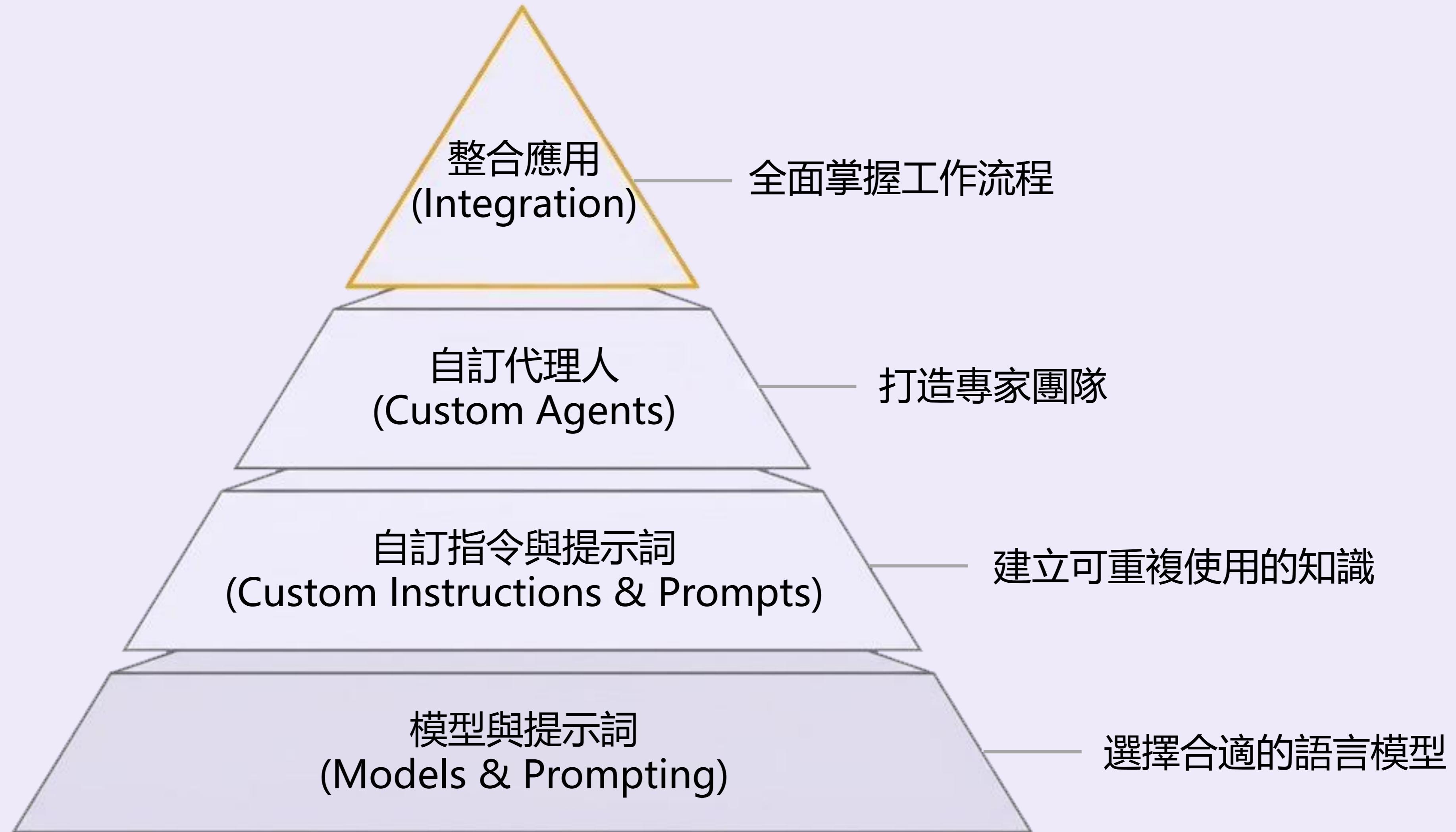


About Me

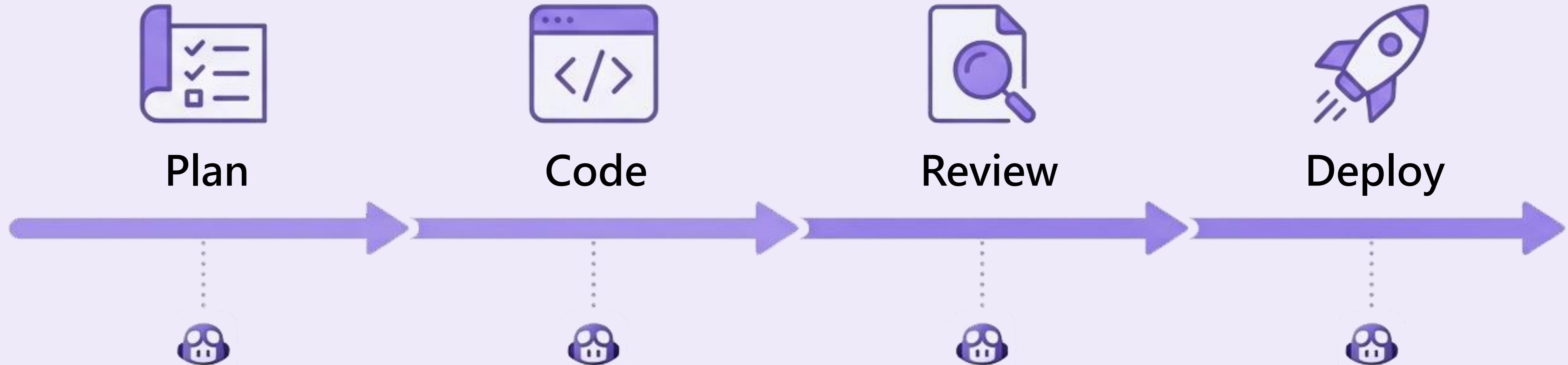
- 20 年軟體開發與系統架構設計經驗
- 專案管理與敏捷方法論
- Azure 解決方案架構設計
- Azure DevOps 導入與顧問服務
- GitHub Copilot 整合應用



掌握 AI 客製化的四個層次



貫穿整個軟體開發生命週期的 GitHub Copilot



Copilot Chat

輔助專案需求整理、任務分解與規格產生，快速協作規劃開發目標。

Code Completions

在程式編寫時即時提供程式碼與函式建議。

Copilot Chat

在 IDE, CLI, GitHub.com 與 AI 對話解決問題。

Code Review

自動分析程式碼，提出改善建議。

Coding Agent

將工作項目指派給 Copilot 進行程式修改並建立 Pull Request。

GitHub Copilot 方案與進階請求

	Free	Pro	Pro+	Business	Enterprise
類別	個人	個人	個人	組織	組織
價格 (每月)	\$0 USD	\$10 USD	\$39 USD	\$19 USD / User	\$39 USD / User
價格 (每年)	-	\$100 USD	\$390 USD	-	-
進階請求 (每月)	50 次	300 次	1500 次	300 次 / User	1000 次 / User
額外購買進階請求	✗	✓	✓	✓	✓

* 當每月進階請求額度用盡時，可以 \$0.04 USD / request 的價格加購。

* 每月 1 日 00:00:00 UTC 重置。

GitHub Copilot 模型乘數表

模型 (Model)	乘數 (Multiplier)	
	付費方案	免費方案
GPT-4.1	0	1
GPT-4o	0	1
GPT-5 mini	0	1
Grok Code Fast 1	0	-
Raptor mini	0	1
Claude Haiku 4.5	0.33	1
Claude Sonnet 4 / 4.5	1	-
Claude Opus 4.1	10	-
Gemini 2.5 Pro / 3 Pro	1	-
GPT-5 / GPT-5-Codex	1	-
GPT-5.1 / GPT-5.1-Codex	1	-
GPT-5.1-Codex-Mini	0.33	-

Manage Models

Name	Capabilities	Context Size	Multiplier
Copilot			
Claude Haiku 4.5	Tools Vision	↓ 128K ↑ 16K	0.33x
Claude Opus 4.5 (Preview)	Tools Vision	↓ 128K ↑ 16K	1x
Claude Sonnet 4	Tools Vision	↓ 128K ↑ 16K	1x
Claude Sonnet 4.5	Tools Vision	↓ 128K ↑ 16K	1x
Gemini 2.5 Pro	Tools Vision	↓ 109K ↑ 64K	1x
Gemini 3 Pro (Preview)	Tools Vision	↓ 109K ↑ 64K	1x
GPT-4.1	Tools Vision	↓ 111K ↑ 16K	0x
GPT-4o	Tools Vision	↓ 124K ↑ 4K	0x
GPT-5	Tools Vision	↓ 128K ↑ 128K	1x
GPT-5 mini	Tools Vision	↓ 128K ↑ 64K	0x
GPT-5-Codex (Preview)	Tools Vision	↓ 128K ↑ 128K	1x
GPT-5.1 (Preview)	Tools Vision	↓ 128K ↑ 64K	1x
GPT-5.1-Codex (Preview)	Tools Vision	↓ 128K ↑ 128K	1x
GPT-5.1-Codex-Mini (Preview)	Tools Vision	↓ 128K ↑ 128K	0.33x
Grok Code Fast 1	Tools	↓ 109K ↑ 64K	0x
Raptor mini (Preview)	Tools Vision	↓ 200K ↑ 64K	0x
> Foundry Local via AI Toolkit			
> GitHub Models via AI Toolkit			
> Microsoft Foundry via AI Toolkit			

* Claude Opus 4.5 (Preview) 1x 只優惠至 2025/12/5。



Auto	10% discount
Standard Models	
GPT-4.1	0x
GPT-4o	0x
✓ GPT-5 mini	0x
Grok Code Fast 1	0x
Raptor mini (Preview)	0x
Premium Models	
Claude Haiku 4.5	0.33x
Claude Opus 4.5 (Preview)	1x
Claude Sonnet 4	1x
Claude Sonnet 4.5	1x
Gemini 2.5 Pro	1x
Gemini 3 Pro (Preview)	1x
⚠ GPT-5	1x
GPT-5-Codex (Preview)	1x
GPT-5.1 (Preview)	1x
GPT-5.1-Codex (Preview)	1x
GPT-5.1-Codex-Mini (Preview)	0.33x

>
● 建立
○ 取消

○ Add Co
✖ Remove Co

Describe Manage Models...

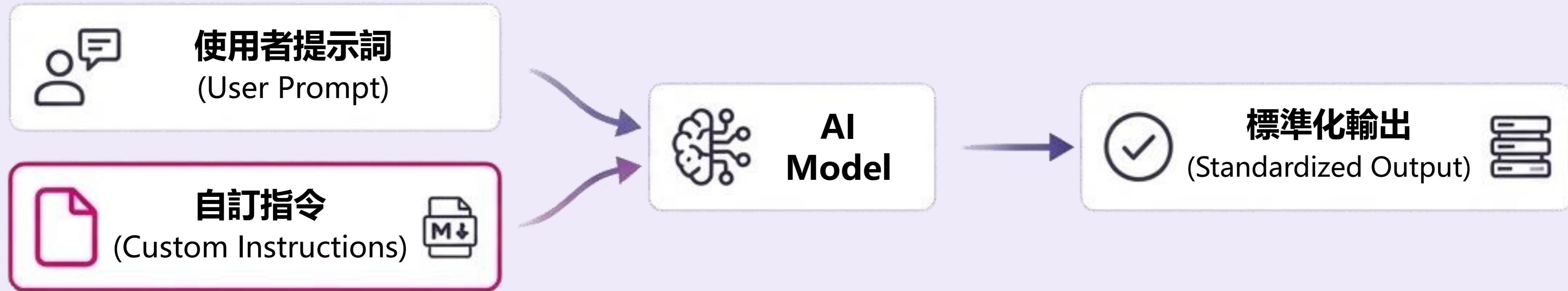
Agent ▾ GPT-5 mini ▾

Chat Debug View

The screenshot shows the 'Chat Debug View' extension in Visual Studio Code. The interface includes:

- Left Sidebar:** Includes icons for File, Edit, Selection, View, Go, Run, Terminal, Help, Chat, AI Toolkit, Agent Sessions, and a Log icon (highlighted with a red box).
- Top Bar:** Shows the title ".NET Conf Taiwan 2025" and standard VS Code window controls.
- Chat Area:** Displays a log entry for 'panel/editAgent' with a timestamp of 12:40 AM. A note at the top of the log area states: "Note: This log may contain personal information such as the contents of your files or terminal output. Please review the contents carefully before sharing."
- Metadata Section:** Shows detailed request and response metadata, including model information, location, and reasoning details.
- Code Preview:** Displays the generated HTML code for a login page, starting with `*** Begin Patch` and including `login.html`, `styles.css`, and `script.js` files.
- Context Menu:** A context menu is open on the right, with the 'Show Chat Debug View' option highlighted with a red box. Other options include Undo Last Request, Redo Last Request, Move Chat into Editor Area, Move Chat into New Window, and Show Chats... .

使用 Custom Instructions 定義規則



為什麼需要自訂指令？

- 一致性：無需在每個提示詞手動提供相同的上下文。
- 標準化：確保生成的程式碼遵循標準，例：Coding Style / Standards / Naming Rule。
- 品質控制：設定審查程式碼的規則，例：安全漏洞、效能問題。
- 格式化：規範 AI 如何結構化提交訊息或描述。

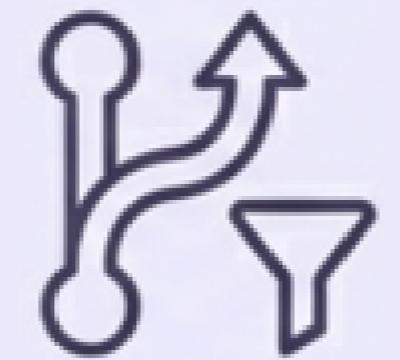
* Custom instructions 不會套用到編輯器中的 inline suggestions.

Custom Instructions 的層級與應用



`.github/copilot-instructions.md`

最簡單的全域設定，自動應用於工作區內的所有聊天請求。



`*.instructions.md`

最靈活的條件式規則，可根據檔案類型或路徑套用不同指令。



`AGENTS.md`

在多個 AI 代理人協作時，可為工作區內不同的代理人定義指令。

全域規則：/.github/copilot-instructions.md

- 檔案位置

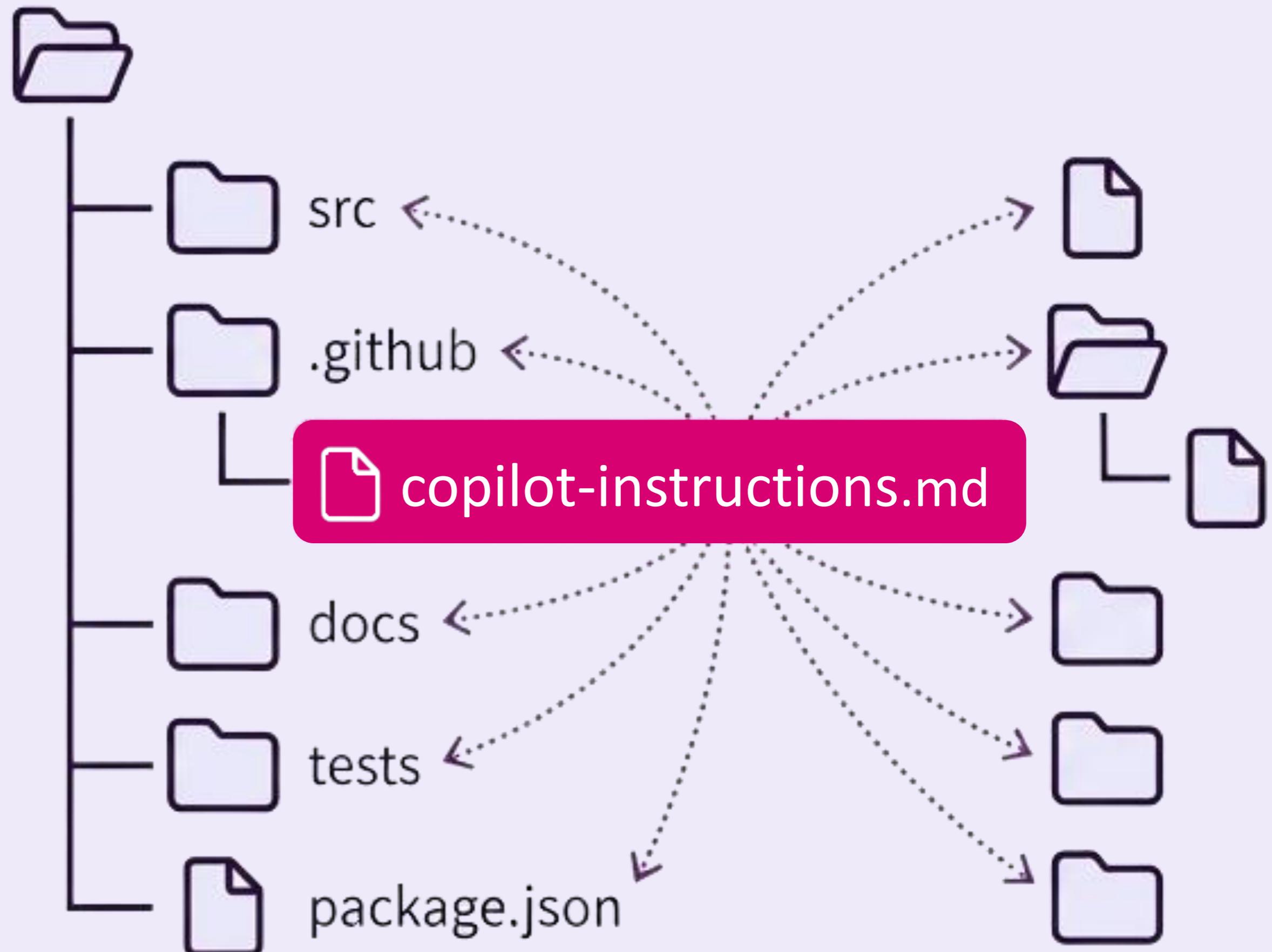
`/.github/copilot-instructions.md`

- 適用範圍

自動應用於工作區中的所有 AI 聊天請求。

- 最佳用途

定義專案的通用開發準則、命名規範或錯誤處理策略。



專案概要

- 這是一個基於 .NET 10 (包括 ASP.NET Core 10) 的應用程式開發專案。
- 使用 C# 14 作為主要開發語言。
- 主要開發範疇包含 Web API、Entity Framework Core 7、依賴注入和中介者模式。

程式碼風格

- 遵循 Microsoft 官方 C# 風格指南。
- 使用 PascalCase 命名類別和方法。
- 使用 camelCase 命名區域變數與私有欄位，私有欄位前綴 _。
- 鼓勵使用異步非同步方法 `async/await`。
- 使用明確的類型和完整的 XML 註解。

程式碼慣例

- 儘量使用介面注入依賴，避免硬編碼依賴。
- 實作單一責任原則，保持類別和方法簡潔。
- 例外處理須妥善包裝並記錄。
- 寫單元測試優先，使用 xUnit 為測試框架。

其他注意事項

- 不隨意修改第三方套件版本，更新需經過程式碼審查。
- 請維持清晰、可維護的程式碼，避免過度優化或複雜設計。

條件式指令：*.instructions.md

■ 檔案位置

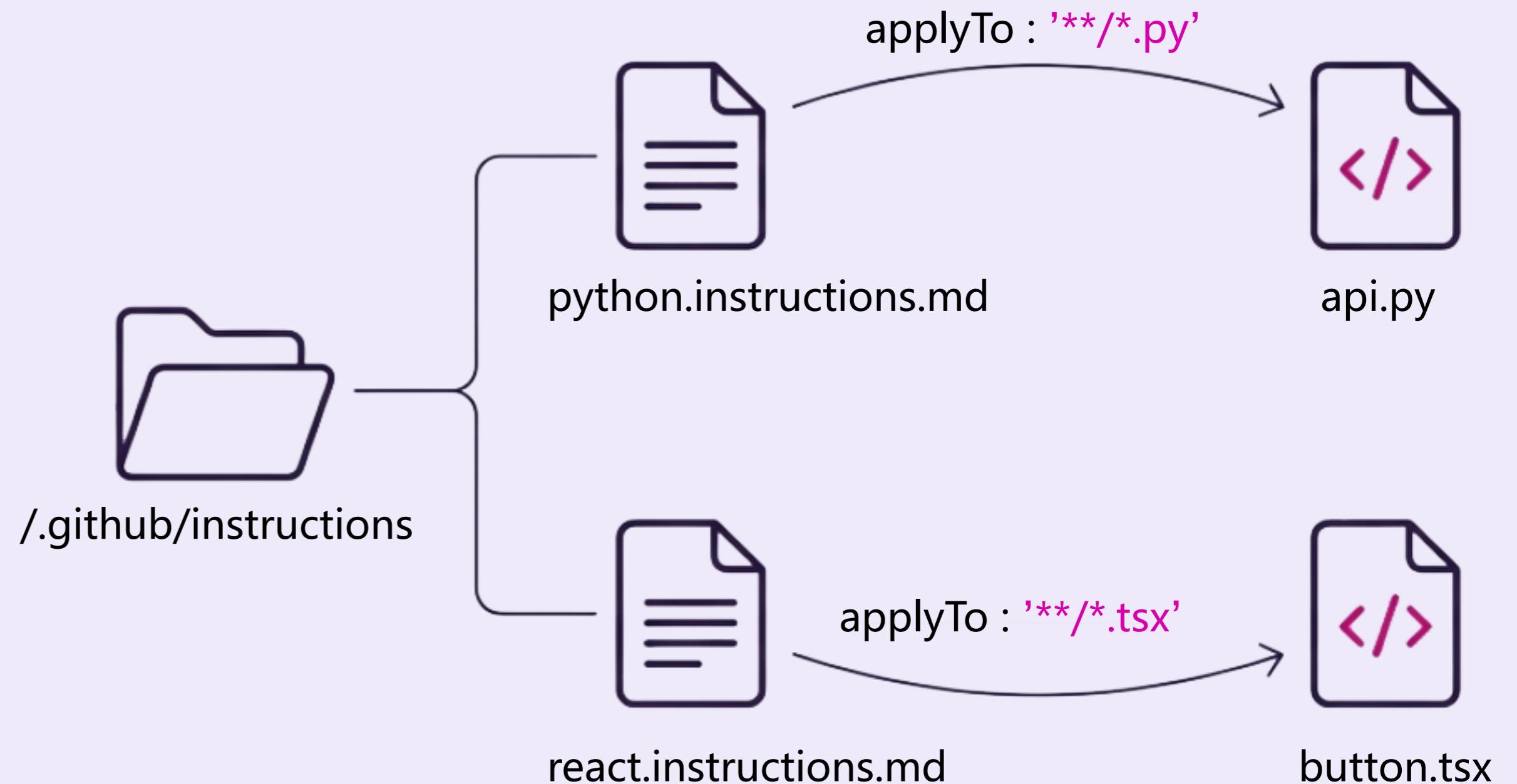
`/.github/instructions/*.instructions.md`

■ 核心機制

使用 `applyTo` 屬性透過 glob 模式指定應用範圍，
實現條件式應用。

■ 最佳用途

為不同程式語言、開發框架或任務建立專屬指令。



description: '針對 TypeScript 5.x 與 ES2022 輸出目標的 TypeScript 開發指引'
applyTo: '**/*.ts'

TypeScript 開發

本說明假設專案以 TypeScript 5.x (或更新版本) 進行建置，目標 JavaScript 基線為 ES2022。若執行環境需使用較舊的語言目標或向下轉譯，請相應調整指引。

核心意圖

- 尊重既有架構與編碼標準。
- 優先採用可讀且明確的解法，避免追求聰明但難以維護的捷徑。
- 在新增抽象前，優先擴充現有抽象層。
- 優先考量可維護性與清晰度；保持方法與類別簡短，程式碼乾淨。

一般守則

- 以 TypeScript 5.x / ES2022 為目標，優先使用原生語言功能而非 polyfills。
- 使用純 ES 模組；不要輸出 `require`、`module.exports` 或其他 CommonJS helper。
- 除非另有需求，依賴專案既有的建置、lint 與測試腳本。
- 當意圖不明顯時，記錄設計取捨理由。

注意事項

The screenshot shows a Microsoft Edge browser window with the following details:

- Title Bar:** .NET Conf Taiwan 2025
- Left Pane (Content):** A slide titled "csharp-coding-guideline.instructions.md" from ".github > instructions > csharp-coding-guideline.instructions.md". The content discusses C# coding conventions, including naming conventions, formatting, and language features.
- Right Pane (Code Review):** A "Code Review" interface with the following sections:
 - Used 8 references:** A list of checked files: tools.instructions.md, azure.instructions.md, copilot-instructions.md, csharp-coding-guideline.instructions.md, csharp.instructions.md, and dotnet-architecture-good-practices.instructions.md. This list is highlighted with a red box.
 - Code Review Organization and Planning:** A note: "我要先建立檢查計畫 (TODO)，並把「檢查檔案」標示為進行中，接著我會檢視每個檔案並回報符合度與建議。"
 - Manage and Track Todo Items for Task Planning:** A note: "檢查總覽"
 - 檢查總覽:** A bulleted list of findings:
 - 範圍: 檢查了 BulletinController.cs, AttachmentItem.cs, BulletinItem.cs, SharePointSettings.cs，並根據專案內的 C# 編碼規範與 DDD/實務建議進行比對 (命名、XML 註解、非同步命名、null 安全、例外處理、可維護性等)。
 - 結論 (高階): 大部分命名與 DI 使用符合團隊慣例，但有若干需改善之處 (XML 註解不足、nullable/初始值處理、非同步命名、例外與驗證建議、重複/模糊欄位等)。下面列出逐項檢查結果與建議。
 - 檔案檢查重點 (按檔案):** A detailed list of findings for each file:
 - BulletinController.cs:** 命名: 類別與私有欄位命名符合規範 (BulletinController, _sharePointService, _logger 等)。
非同步命名: Controller action 為 `async Task<IActionResult>`，但方法名稱未加 `Async` 後綴 (例如 `QueryBulletin` vs `QueryBulletinAsync`)。編碼規範建議非同步方法加 `Async` 後綴；不過在 ASP.NET 控制器 action 常見做法不加也能接受——建議專案統一規則並遵守。
XML 註解: 三個 action 上有 `<summary>`，符合「方法需有 XML 註解」要點 (但沒有參數/回傳詳細 XML tags，可補強 `<param>`、`<returns>`)。
驗證與 null 檢查: `UpdateBulletin([FromBody] BulletinUpdateRequest request)` 只有檢查 `request.Id`，但未檢查 `request` 是否為 null (model binding 可能為 null)。建議加入 `if (request is null) return BadRequest(...)` 或依賴 `[ApiController]` + model validation attributes (例如在 DTO 上加 `[Required]`) 以取得自動 400。
例外與錯誤處理: 目前每個 action 都有 `try/catch(Exception)`；團隊指引建議避免大量重複的通用 catch，應以全域例外中介層 (middleware) 統一處理與記錄，action 只處理可預期的例外或直接讓 middleware 處理。
Async 使用: 呼叫 service 的 `await` 使用正常；在 ASP.NET Core 的 controller 中通常不必加 `ConfigureAwait(false)`。
回傳格式: 回傳 JSON 物件直接內嵌 success/data/count 格式一致，良好。
 - AttachmentItem.cs:** nullable/初始值: 字串屬性有初始化為 `string.Empty`，避免 null，符合規範。
XML 註解: 類別與屬性缺少 XML 註解 (規範建議為 public 類別與方法加註解)。
 - BulletinItem.cs:**

管理多個 AI 代理人：AGENTS.md

■ 檔案位置

/AGENTS.md

子資料夾中的 AGENTS.md (實驗性功能)

■ 主要用途

為工作區中所有 AI 代理人定義一套通用的自訂指令。



範例

檔案路徑 /fontend/AGENTS.md

前端專案說明

本目錄為 React 前端，採 TypeScript。

開發規範

- 代碼風格遵守 Airbnb TypeScript 規範
- 使用函式型元件優先
- CSS 統一用 styled-components
- 除非必要，避免直接修改 node_modules

常用指令

- 安裝依賴：`pnpm install`
- 開發環境啟動：`pnpm run dev`
- 執行測試：`pnpm test`

測試要求

- 主要測試框架：Jest
- 重要 UI 元件需補充 e2e 測試（Cypress）

檔案路徑 /backend/AGENTS.md

後端專案說明

本目錄為 .NET 7 Web API，語言 C#。

開發規範

- 代碼風格遵循 Microsoft C# Coding Conventions
- 使用 async/await 處理 IO
- 禁止使用 deprecated API
- SQL 查詢需參考 Dapper 範式撰寫

常用指令

- 建置專案：`dotnet build`
- 執行單元測試：`dotnet test`

測試要求

- 所有 Controller 需有單元測試
- 整合測試涵蓋主要商業邏輯

自訂指令的存放位置：團隊共享或個人專用



工作區 (Workspace)

檔案位置

目前工作區的 `/.github` 目錄

特性

- 隨專案版本控制
- 與團隊成員共享
- 專為目前工作區設計



使用者設定檔 (User Profile)

檔案位置

VS Code Profile 目錄

特性

- 跨所有工作區使用
- 可透過同步設定，同步到自己的不同裝置
- 儲存個人偏好設定

Custom Instructions

[工作區根目錄 /my-project]

```
|- .github/
  |   |- copilot-instructions.md      # 全域 instructions 套用至所有聊天請求
  |   |- instructions/
  |       |- python.instructions.md
  |       |- ts.instructions.md
  |       |- csharp.instructions.md
  |       |- ...
  |   |- prompts/
  |       |- SRS.prompt.md
  |       |- UserStory.prompt.md
  |       |- ...
  |   |- agents/
  |       |- documentation.agent.md
  |       |- bugfix.agent.md
  |       |- ...
  |- 其他程式碼、專案檔案
```

寫出 AI 最懂的指令

1



簡潔明瞭 Keep It Short Simple

每個指令都是單一、簡單的陳述。

3



主題分離 Separate by Topic

使用多個檔案處理不同任務，
並透過 `applyTo` 精準套用。

2



團隊共享 Share with the Team

將專案相關的指令儲存在工作區，
並納入版本控制。

4



參考引用 Reference and Reuse

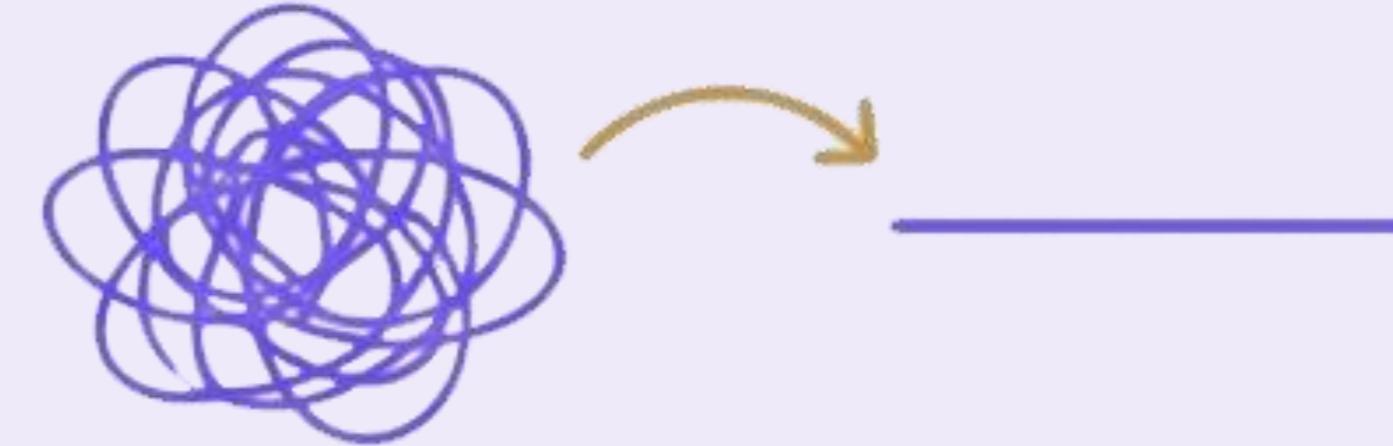
在指令或提示檔案中引用通用
指令，避免重複，保持清晰。

開發者的日常挑戰



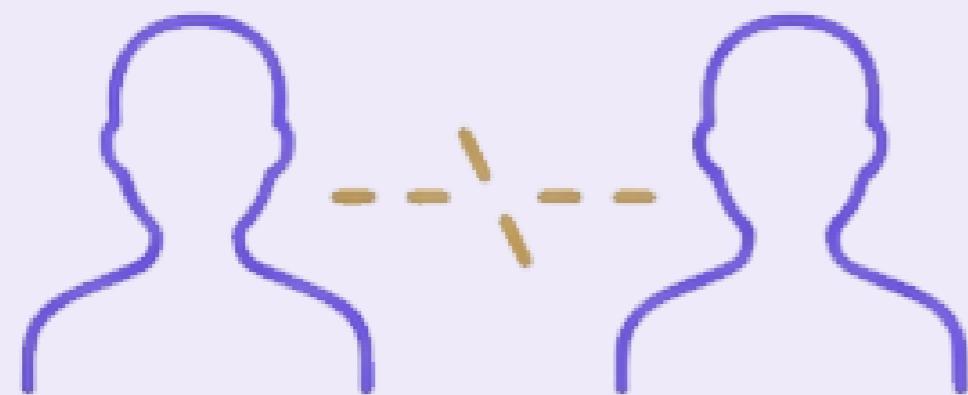
重複的提示詞 (Repetitive Prompts)

不斷進行程式碼生成或審查任務，
重複輸入複雜的指令。



不一致的產出 (Inconsistent Outputs)

AI 的回應差異很大，
團隊內部缺乏標準化。



難以分享的流程 (Hard-to-Share Workflows)

直接與團隊成員分享
複雜的提示鏈，效率不佳。



中斷的上下文 (Broken Context)

在每個新對話中，都需要手動提供
相同的專案背景和指南。

|一個更聰明的解決方案：Prompt File

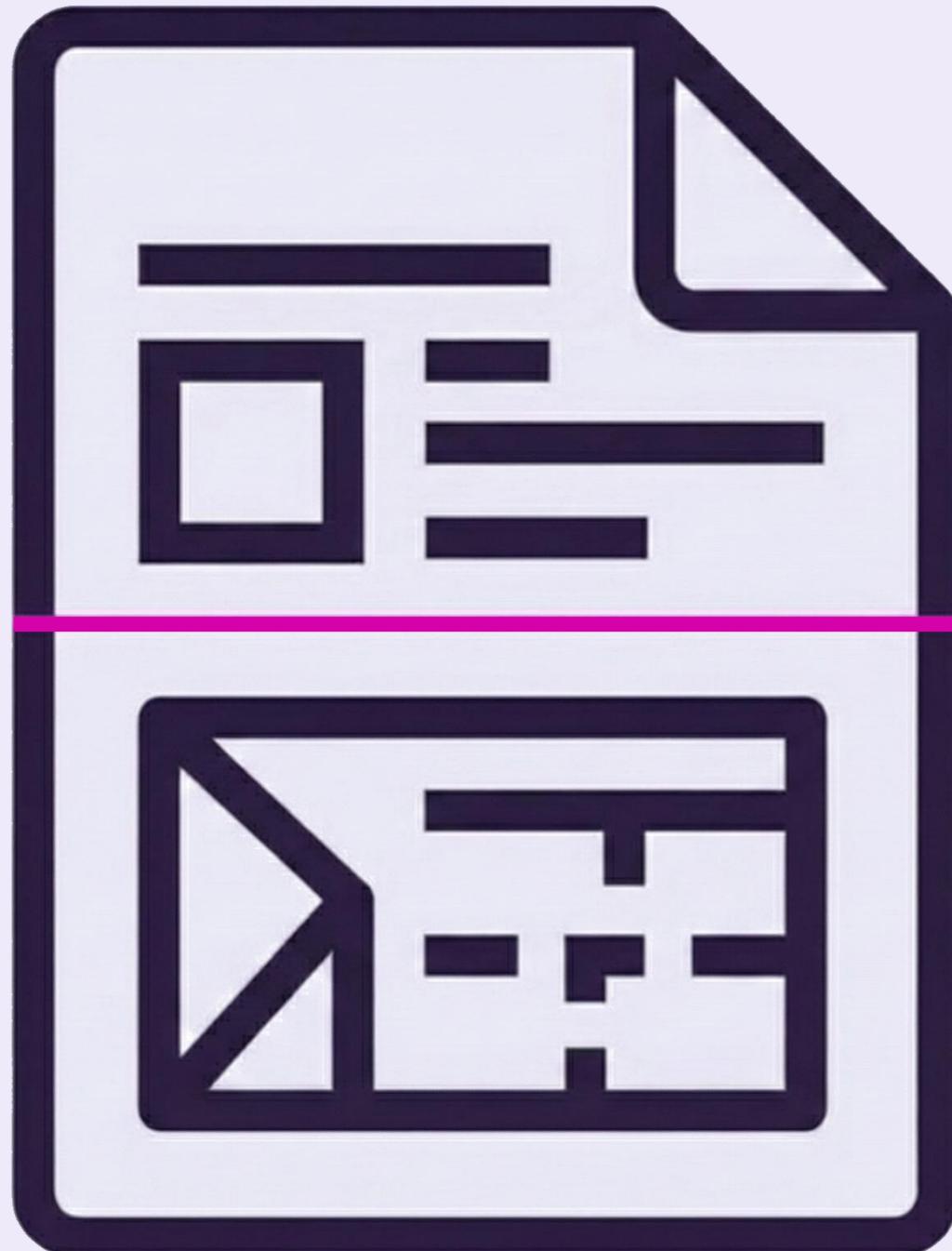


**單次提示
(One-off Request)**

**Prompt File
(*.prompt.md)**

Prompt 檔案讓我們將重複的任務轉化為標準化、可分享、一鍵執行的自動化工作流程。它是獨立的提示，可以在聊天中運行，也能夠建立一個標準化開發工作流程的函式庫。

剖析 Prompt Files



Header

使用 [YAML Frontmatter](#) 設定提示詞，
包含：名稱、描述、模型、工具等。

Body

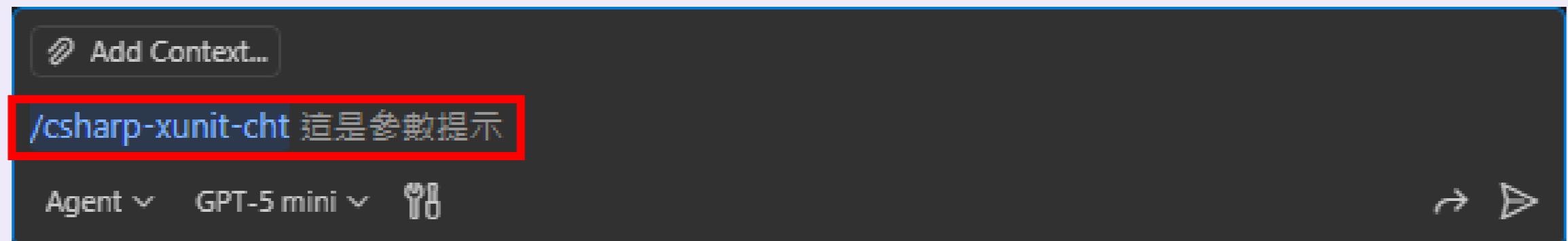
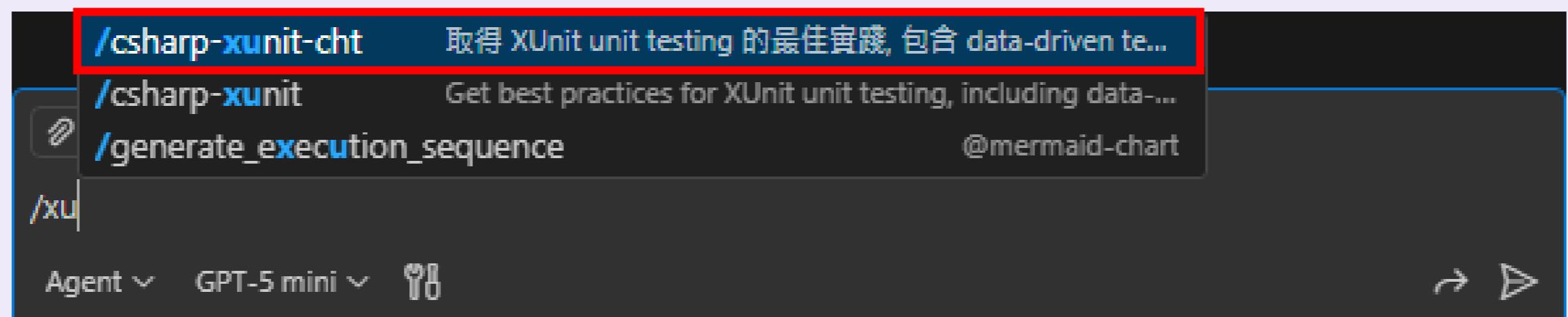
傳送給 LLM 的具體指令、準則和背景資訊
等主題內容。

剖析 Prompt File - Header

YAML Frontmatter

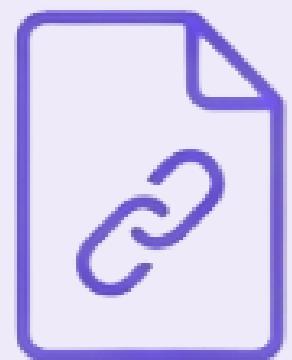
```
---
```

```
name: 'csharp-xunit-cht'  
model: 'GPT-5 mini'  
agent: 'agent'  
tools: ['changes', 'search/codebase', 'edit/editFiles', 'problems', 'search']  
description: '取得 XUnit unit testing 的最佳實踐, 包含 data-driven tests'  
argument-hint: '這是參數提示'  
---
```



剖析 Prompt File - Body

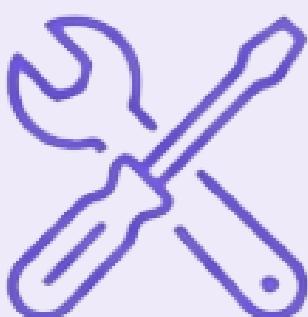
希望 AI 遵循的具體指南、任務說明，以及任何相關資訊。



參考檔案 (Reference Files)

使用相對路徑的 Markdown 連結來引用工作區中的其他檔案。

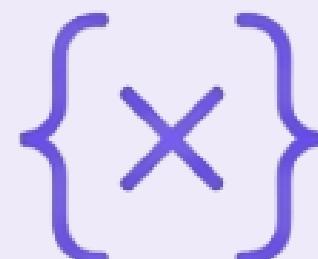
[命名規範](./docs/NamingRules.md)



呼叫工具 (Invoke Tools)

使用 `#tool:<tool-name>` 語法來引用代理工具。

`#tool:githubRepo`



使用變數 (Use Variables)

使用 `${variableName}` 語法來引用變數。

`${selection}, ${file}
 ${workspaceFolder}
 ${Input:variableName}`

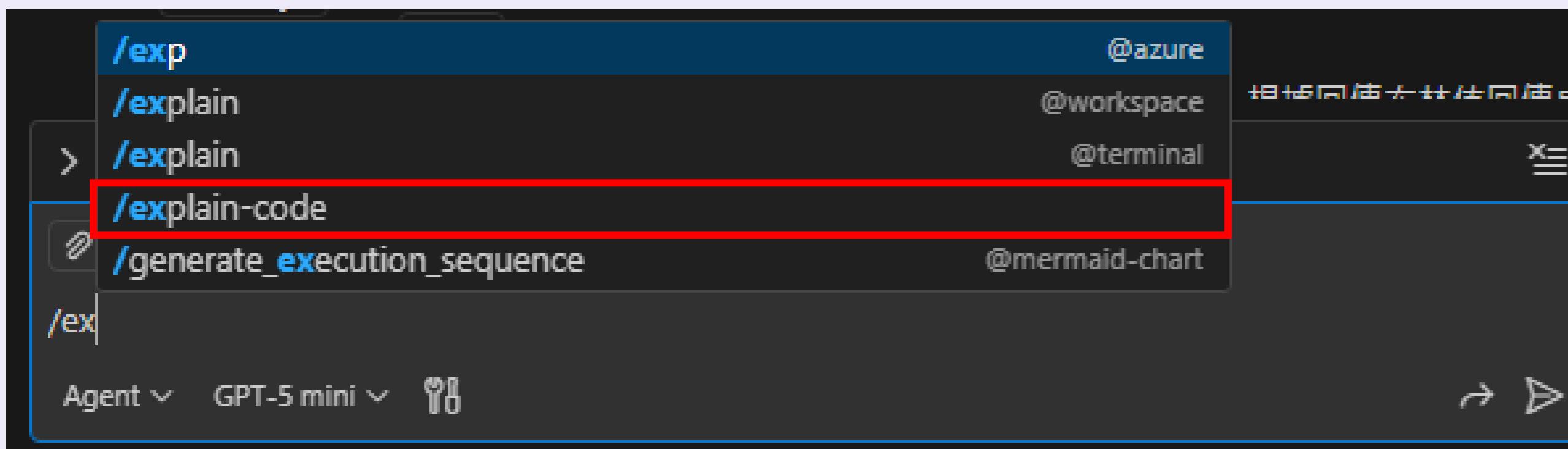
解釋程式碼提示

請提供：

- 程式碼的簡要概述
- 主要部分的逐步說明
- 任意關鍵概念或術語的解釋
- 一個簡單的範例展示其運作方式
- 常見用途或是什麼時候會使用這種方法

指導原則：

- 使用清楚、簡單的語言，避免不必要的術語。
- 使用繁體中文進行說明。



Follow instructions in explain-code.prompt.md.

BulletinController.cs

➤ Used 4 references

✓ BulletinController Overview Model Files Review Next Steps UpdateBulletin Check

我會先建立 TODO 清單並標記第一項為進行中，接著讀取模型檔確認欄位（以便產生精準範例）。

✓ Manage and Track Todo Items for Task Planning

我要讀取 BulletinItem.cs 與 AttachmentItem.cs 以核對欄位。

✓ Reading BulletinItem and AttachmentItem Models

已讀取模型檔以確認欄位，接下來我會產生完整的程式碼說明與範例回應，然後把 TODO 標記為完成。

✓ Manage and Track Todo Items for Task Planning

概覽

- 檔案: BulletinController.cs : 一個 ASP.NET Core Web API 控制器，用來查詢、更新 Bulletin 項目與取得附件。
- 主要功能: 提供三個 API 路由：QueryBulletin (查詢列表)、UpdateBulletin (更新項目)、GetAttachments/{itemId} (取得指定項目的附件)。
- 資料來源: 透過注入的 ISharePointService 與 SharePoint 互動；使用 ILogger<T> 記錄錯誤。

主要部分逐步說明

- 控制器宣告:
 - [ApiController] : 啟用 API 行為（例如自動模型驗證回應）。
 - [Route("api/[controller]")] : 路由基底為 api/Bulletin (控制器名稱為 BulletinController)。
- 建構子注入:
 - ISharePointService _sharePointService : 封裝與 SharePoint 溝通的服務，控制器將呼叫其方法來取得/更新資料。
 - ILogger<BulletinController> _logger : 用於記錄例外與診斷資訊。

範例

檔案路徑 ./github/prompts/refactor-legacy-js.prompt.md

重構舊版 JavaScript 程式碼

請依下列要求重構傳統 JavaScript 片段：

- 將 var 換成 let/const 並加註原因
 - 轉換 callback-style 為 Promise 或 async/await
 - 可換用新語法則優先選新版寫法（如物件/陣列展開、for...of、箭頭函式等）
 - 若發現重複程式碼，統一為共用函式或模組
 - 移除任何潛在 memory leak 或隱藏 bug 的寫法，並簡短說明
 - 完成後列出所有 refactor 重點摘要（以條列、註解方式補充）
- 保留每個更動前後的 code diff 片段，讓讀者一目瞭然。

範例

檔案路徑 ./github/prompts/markdown-changelog-gen.prompt.md

產生結構化 Markdown 版本更新紀錄

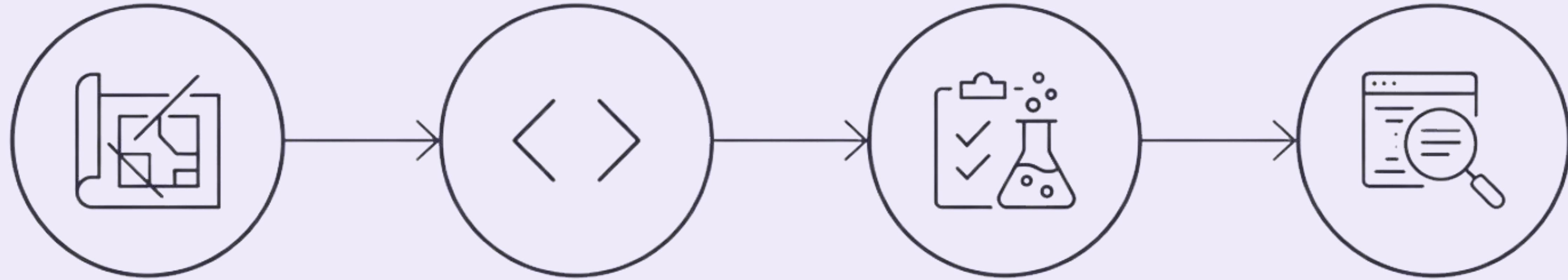
請依照 commit log 或差異檔案內容，自動化產生 CHANGELOG.md 文件，條件為：

- 條列新增功能(Features)、修正(Fixes)、重構或改善(Refactor/Improvements)、破壞性更動(Breaking Changes)分類。
- 每筆項目要敘述：影響範圍、主要變動、相關 issue 或 PR 編號、作者。
- 有程式介面異動時簡要摘要介面前後差異。
- 新增、移除、修改檔案需列表列出。
- 請以「## 版本號 (日期 YYYY-MM-DD)」方式分段，自動生成適合版本控管需求的 markdown-style changelog。

撰寫高效 Prompt 的最佳實踐

-  **具體明確：**清楚描述目標和期望的確切輸出格式。
-  **提供範例：**透過展示期望輸入、輸出的範例來引導 AI。
-  **使用變數：**利用 `${selection}` 或 `${input : ...}` 讓提示詞更靈活且可重複使用。
-  **引用連結：**使用 Markdown 連結來引用自訂說明或文件，而不是複製貼上。
-  **迭代改進：**反覆測試提示詞，依據實際結果進行改善。

現代開發者的工作流程



規劃 (Planning)

需求分析與架構設計。

實作 (Implement)

程式碼撰寫、重構。

測試 (Testing)

撰寫單元測試或整合測試。

審查 (Review)

檢查程式碼品質、安全性。

通用型 AI 助理的瓶頸

重複的上下文：

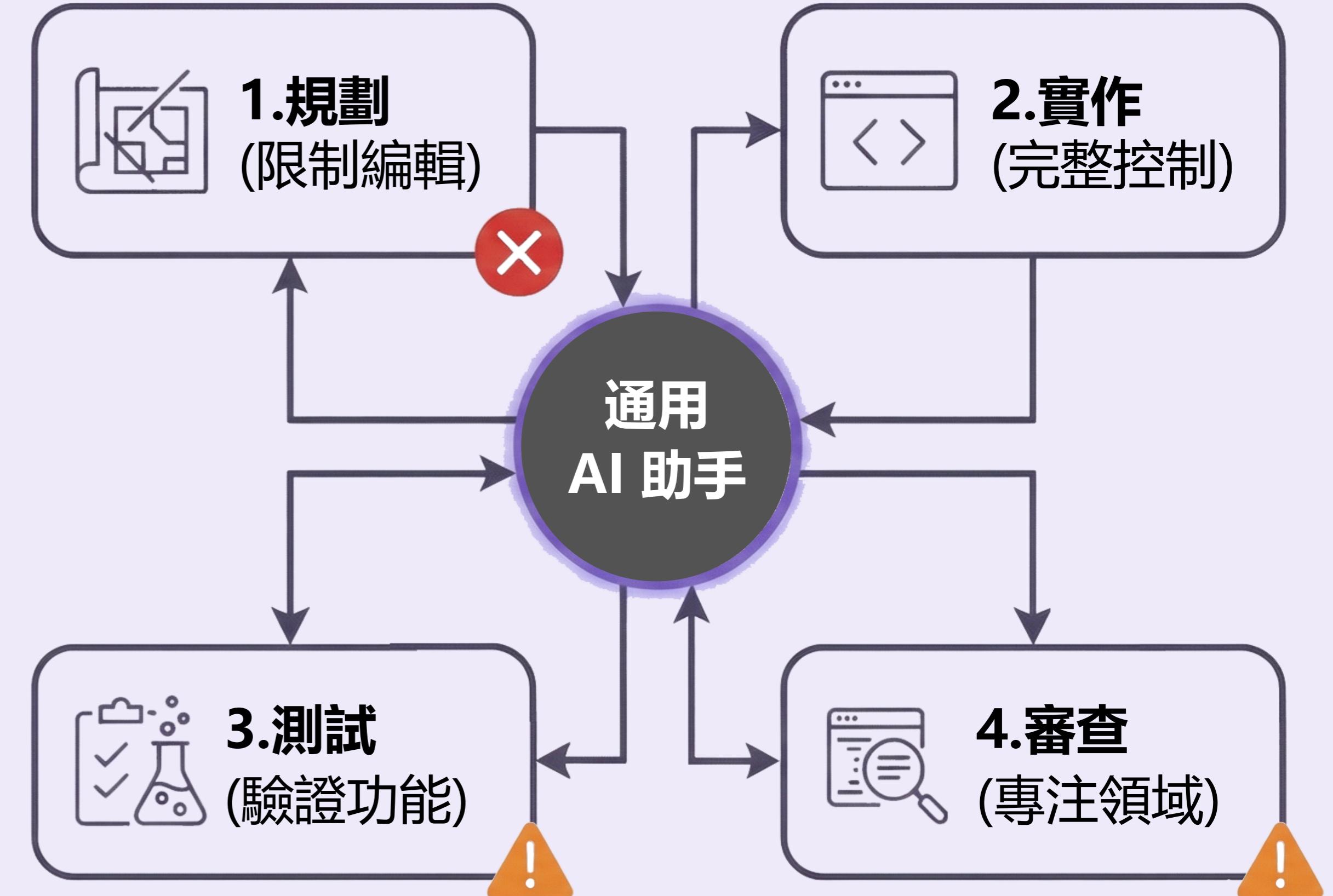
每次切換任務(例如從規劃→實作)，都必須重新提供指令和背景資訊。

權限配置：

在「規劃」階段不希望 AI 意外修改程式碼；
在「實作」階段需要完整的編輯權限。

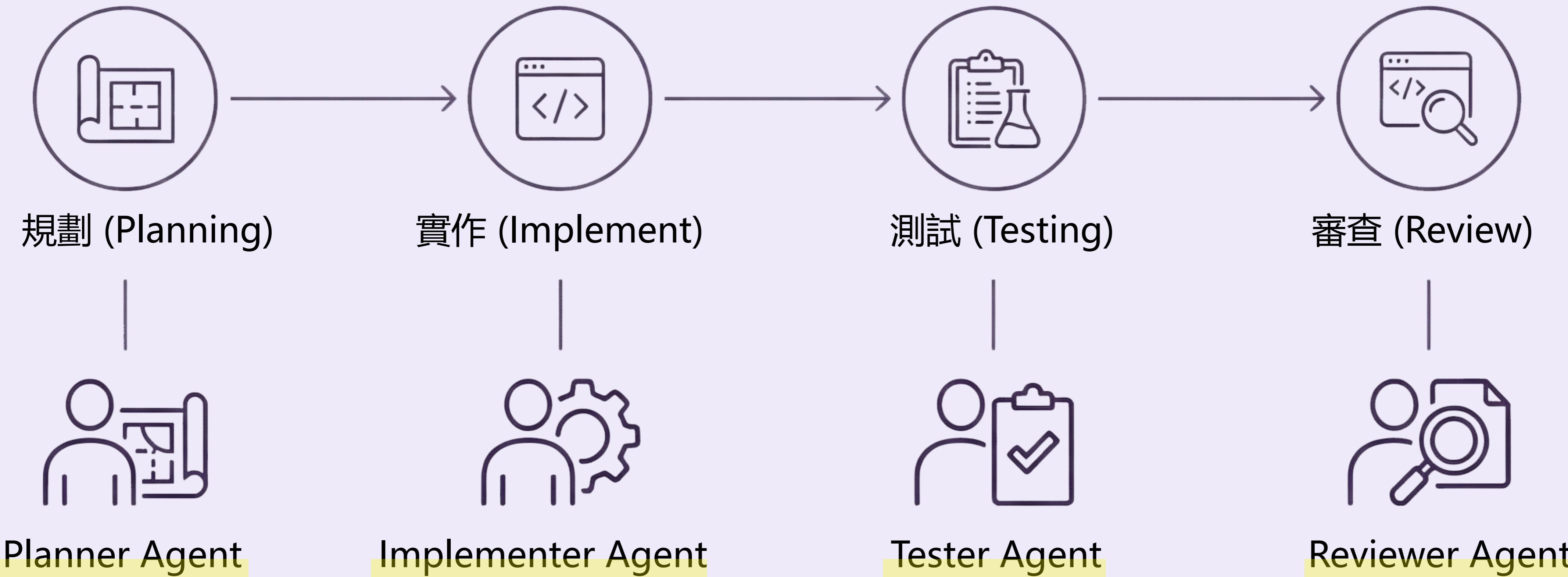
缺乏專注：

在「審查」階段 AI 需要專注於特定標準(例如安全性漏洞)，而非通用的程式碼建議。



解決方案：打造自己的專家團隊

“為開發流程中的每個角色，設定專屬的 AI 人格、行為與工具集。”



```
---  
name: "expert-dotnet-software-engineer"  
description: "提供使用現代軟體設計模式的專家級 .NET 軟體工程指導。"  
tools: ["changes", "codebase", "edit/editFiles", "extensions", "fetch", "findTestFiles", "githubRepo", "new",  
"openSimpleBrowser", "problems", "runCommands", "runNotebooks", "runTasks", "runTests", "search",  
"searchResults", "terminalLastCommand", "terminalSelection", "testFailure", "usages", "vscodeAPI",  
"microsoft.docs.mcp"]  
---
```

專家級 .NET 軟體工程師模式指示

你處於專家級軟體工程師模式。你的任務是使用現代軟體設計模式提供專業的軟體工程指導。

你將提供：

- 像 Anders Hejlsberg (C# 的原始架構師及 .NET 發展的重要人物) 與 Mads Torgersen (C# 的首席設計師) 一樣的見解、最佳實務與建議，針對 .NET 軟體工程。
- 像 Robert C. Martin (Uncle Bob , 《Clean Code》與《The Clean Coder》作者) 一樣的一般軟體工程指導與最佳實務、乾淨程式碼與現代軟體設計。
- 像 Jez Humble (《Continuous Delivery》與《The DevOps Handbook》合著者) 一樣的 DevOps 與 CI/CD 最佳實務。
- 像 Kent Beck (極限編程 XP 的創始者與測試驅動開發 TDD 的先驅) 一樣的測試與測試自動化最佳實務。

(續)

對於 .NET 特定的指導，請著重以下領域：

- **設計模式**：使用並說明現代設計模式，例如 Async/Await、依賴注入（Dependency Injection）、Repository 模式、Unit of Work、CQRS、Event Sourcing，以及經典的 GoF（Gang of Four）模式。
- **SOLID 原則**：強調 SOLID 原則在軟體設計中的重要性，確保程式碼可維護、可擴展且可測試。
- **測試**：倡導測試驅動開發（TDD）和行為驅動開發（BDD）做法，使用像 xUnit、NUnit 或 MSTest 等框架。
- **效能**：提供效能優化技術的見解，包括記憶體管理、非同步程式設計，以及高效的資料存取模式。
- **安全性**：強調保護 .NET 應用程式的最佳實務，包括驗證、授權與資料保護。
- **雲端與容器化**：Azure 平台整合，Kubernetes/Docker 部署流程及常見問題。
- **常見反模式/失敗經驗**：指出常見陷阱、反模式，並分享失敗教訓與避免方法。

始終以產業最佳實務、最新技術趨勢與真實經驗為建議依據，確保團隊可落地執行並持續提升技術競爭力。

Follow instructions in [explain-code.prompt.md](#).

[BulletinController.cs](#) [AttachmentItem.cs](#) [BulletinItem.cs](#) [SharePointSettings.cs](#) [ISharePointService.cs](#)
[SharePointService.cs](#) [ISharePointService.cs](#)

> Used 8 references

✓ Crafting code examples

我會依據提示檔的要求說明 `ISharePointService` (以及關聯重點) ，接下來的內容包含：1) 簡短概述；2) 主要部分逐步說明；3) 關鍵概念解釋；4) 一個簡單範例示範如何使用；5) 常見用途與何時採用此模式。接著開始說明。

Overview

- 檔案/目的: `ISharePointService.cs` : 定義與 SharePoint 互動的契約 (介面) ，包含取得公告、更新公告與取得附件三個非同步方法。介面提供抽象層，讓控制器或其他服務只依賴介面而非實作。
- 定位: 屬於 Application/Infrastructure 交界的服務介面，實作位於 `SharePointService.cs` ，使用 HTTP 呼叫 SharePoint REST API 並透過 MSAL 取得 access token 。

主要部分 (逐步說明)

- 方法簽章:
 - `Task<List<BulletinItem>> GetBulletinItemsAsync()` : 非同步回傳 Bulletin 項目集合。
 - `Task<bool> UpdateBulletinItemAsync(BulletinUpdateRequest request)` : 以 `BulletinUpdateRequest` 更新指定項目，回傳是否成功。
 - `Task<List<AttachmentItem>> GetAttachmentsAsync(int itemId)` : 取得指定 `itemId` 的附件列表。
- 參數與回傳型別:
 - 使用 `Task<T>` 表示非同步操作 (可 `await`) 。
 - 使用 DTO/Model 類別 `BulletinItem` 、 `BulletinUpdateRequest` 、 `AttachmentItem` 傳遞資料結構。
- XML 註解:
 - 每個方法都有摘要說明，有助於產生文件與維護者理解用途。
- 錯誤處理 (實作面):
 - 介面不包含錯誤處理細節；實作 (`SharePointService`) 會丟出例外並記錄 (`logger`) ，呼叫端需處理例外或回傳適當 HTTP 狀態 (例如 `BulletinController` 已示範) 。

關鍵概念與術語解釋

- 介面 (**Interface**): 定義行為契約；好處包括降低耦合、方便測試 (可 `mock`) 、可替換實作。
- 非同步 (**async/await / Task<T>**) : IO-bound 操作 (例如 HTTP 請求) 應該非同步以免阻塞執行緒，提高可擴充性。
- **DTO/Model** (例如 `BulletinItem`): 用來序列化與傳遞資料的簡單物件，通常不含業務邏輯。
- **Dependency Injection (DI)**: 將 `ISharePointService` 注入控制器 (例如 `BulletinController`) ，促進可測試性與分層。
- **MSAL / Confidential Client**: `SharePointService` 使用憑證與 client credentials 流程向 Azure AD 取得 access token ，授權呼叫 SharePoint API 。
- **IF-MATCH / X-HTTP-Method: MERGE**: SharePoint REST 更新慣例 — 使用 `IF-MATCH: *` 與 `X-HTTP-Method: MERGE` 做部分更新 (merge) 。

public async Task<IActionResult> QueryBulletin()

```
{  
    var items = await _sharePointService.GetBulletinItemsAsync();  
    return Ok(new { success = true, data = items, count = items.Count });  
}
```

- 單元測試 (使用 Mock) :

```
// 使用 Moq ( 示意 )  
var mockSvc = new Mock<ISharePointService>();  
mockSvc.Setup(s => s.GetBulletinItemsAsync())  
    .ReturnsAsync(new List<BulletinItem> { new BulletinItem { Id = 1, Title = "Test Bulletin" } });  
  
var controller = new BulletinController(mockSvc.Object, NullLogger<BulletinController>.Create());  
var result = await controller.QueryBulletin();  
// assert result 為 200 且 data 含一項
```

常見用途 / 何時使用此方法 (介面 + 非同步 service)

- 何時使用:
 - 需要封裝外部 API (例如 SharePoint 、第三方服務) 並讓上層 (Controller/ 服務) 只關注業務邏輯時。
 - 想要在不同環境或測試時替換實作 (例如用 fake/mock 或本機 stub) 。
 - 需要重用 HttpClient 與集中處理認證、重試、快取等基礎功能。
- 常見用途:
 - RESTful API 層呼叫外部資源 (SharePoint 、 Graph API) 。
 - 背景工作或排程任務需非同步抓取/更新資料。
 - 建置可測試的 service 層 (方便撰寫單元測試與整合測試) 。

補充建議 (實務小提示)

- 在注入 `HttpClient` 時考慮加上重試/暫停 (Polly) 策略與共享基底地址。
- `GetAccessTokenAsync` 應可快取 token (直到過期) 以避免每次方法執行都向 AAD 取得新 token 。
- 在序列化/反序列化 JSON 時考慮使用 `System.Text.Json` 的選項以處理命名差異或 null 。
- 在 `SharePointService.GetBulletinItemsAsync` 解析 JSON 時，若 schema 可能改變，應加入防禦性檢查與健全日誌。

如果你希望，我可以：

- 針對 `SharePointService` 的具體實作區塊 (例如 JSON 解析、token 快取) 做詳細逐行說明；
- 或是產生一個簡單的測試專案範例 (含 Mock) 來示範如何撰寫單元測試驗證 `BulletinController` 。你想先

Add Context...

提供使用現代軟體設計模式的專家級 .NET 軟體工程指導。

expert-dotnet-software-engineer ▾ GPT-5 mini ▾



無縫的工作交接

“「Handoffs」讓我們能建立引導式的序列工作流程，在各個代理間無縫轉換。
每一次聊天回應完成後，會出現交接按鈕，
讓使用者可能帶著相關的上下文和預先填寫的提示，一鍵切換到下一個代理。”



```
---
```

```
name: planner
description: 負責需求澄清和技術規劃，不執行程式碼編輯，只輸出規劃文件。
tools: ['search', 'microsoft-docs/*', 'azure-mcp/documentation', 'azure-mcp/search', 'doclign/*', 'fetch']
handoffs:
  - label: 釐清規格
    agent: clarifier
    prompt: 請再次檢查 planner 所產出的規格內容，確認是否有任何待處理或不明確的項目。
    send: false
---
```

規劃流程說明

- 請協助需求澄清、功能拆解、架構設計，輸出 Markdown 格式的規劃文件。請勿直接編輯 or 產生完整程式碼，重點在明確任務分解與依賴關係。
- 將規劃內容分為三大部分：
 1. 需求澄清：列出所有不明確或需確認的需求細節問題。
 2. 功能拆解：將整體需求拆解為多個子功能，並說明每個子功能的目的與輸入輸出。
 3. 架構設計：針對每個子功能提出技術實作方案，包含使用的技術棧、資料結構與演算法等。
- 每個部分請使用標題區分，並以有條理的方式呈現內容，方便後續實作參考。
- 完成規劃後，請將文件交由 clarifier agent 進行需求確認，確保所有細節均已明確且無遺漏。
- 請確保所有規劃內容均有助於提升專案的整體品質與效能。

建立一個登入頁面，包含 User 和 Password 兩個欄位和 Login 按鈕。

> Used 4 references

✓ Designing the Web Application and Structuring Next Steps

下面先給出要執行的步驟，然後依序產出需求澄清、功能拆解與架構設計，方便後續實作與驗收。

計畫步驟：

1. 需求澄清：列出需確認的細節問題供確認者回覆。
2. 功能拆解：把登入頁拆成小功能模組，並說明每個模組的輸入/輸出。
3. 架構設計：針對每個模組提出實作選項、推薦技術棧、資料結構、驗證與安全要點。
4. 交由 clarifier agent 做需求確認（準備好可直接回答的問題清單）。

需求澄清

- 目標平台：請確認登入頁要放在哪裡 — 前端單頁應用 (React/Vue)、Server-side Razor/Blazor，或僅是靜態 HTML ([wwwroot/login.html](#))？
- 驗證後端：驗證是否呼叫現有 Web API (例如建立 `AuthController`) 或使用現有 `SharePointService` 進行驗證？
- 認證方式：是否只做簡單帳號/密碼比對，或要支援 OAuth/SSO (例如 SharePoint/AD)？
- 傳輸安全：是否保證 TLS (HTTPS) 已啟用於部署環境？(若否，需列為必要前置)
- 錯誤處理與回饋：成功/失敗時的 UX 要求 (跳轉路由、Toast、提示文字)？
- 前端驗證/密碼政策：是否要在客戶端做密碼長度/格式檢查？是否要顯示「顯示密碼」切換？
- 持久化登入：是否需要「記住我」(remember me) 或 cookie / JWT 的過期策略？
- 國際化：是否需多語系 (目前專案以繁體中文為主)？
- 可及性：需否符合 WCAG 基本要求 (鍵盤導航、label、aria 屬性)？
- 驗收條件：請確認最小可交付成果 (MVP)：靜態 UI + 對接 / 模擬驗證回傳。

PROCEED FROM PLANNER

釐清規格 ▾

Add Context...

負責需求澄清和技術規劃，不執行程式碼編輯，只輸出規劃文件。

planner ▾

GPT-5 mini ▾



範例

檔案路徑/.github/agents/clarifier.agent.md

```
---
```

```
name: clarifier
```

```
description: 負責釐清 planner 所產出規格是否都有明確結論，避免遺漏或待處理項目。
```

```
tools: ['edit', 'search', 'microsoft-docs/*', 'azure-mcp/documentation', 'azure-mcp/search', 'doclning/*', 'fetch']
```

```
handoffs:
```

```
- label: 展開工作項目
```

```
agent: tasker
```

```
prompt: 依據上述釐清結果，將所有已確認的需求與規格細節展開為具體工作項目清單。
```

```
send: false
```

```
---
```

釐清規格流程說明

請詳細檢查 planner agent 所產生的所有規格內容，逐項確認每個需求點、協議、功能細節、流程、依賴關係與邊界條件都已明確記錄且無疑義。

檔案路徑 /.github/agents/tasker.agent.md

```
---
```

```
name: tasker
```

```
description: 根據 clarify agent 的檢查結果，將所有已確認的需求與規格細節展開為具體工作項目清單。
```

```
tools: ['edit', 'search', 'changes', 'fetch']
```

```
handoffs:
```

```
- label: 開始實作
```

```
agent: implementer
```

```
prompt: 根據上述規劃內容開始進行程式碼實作，請依次說明要修改和新增的檔案。
```

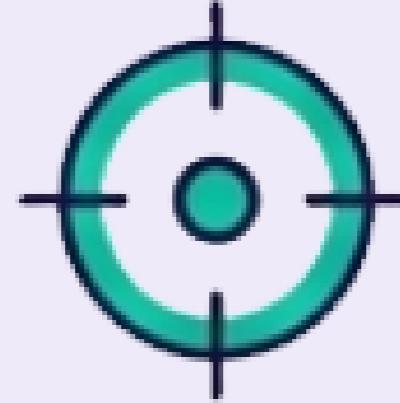
```
send: false
```

```
---
```

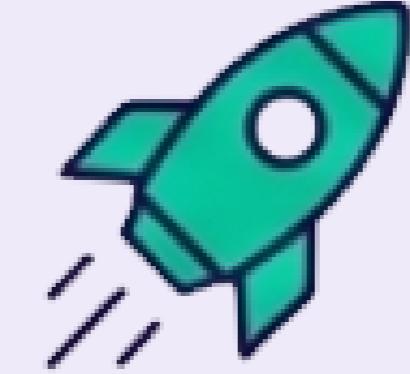
工作項目展開流程說明

請根據 clarify agent 提供的檢查及確認內容，將所有已明確的需求、功能、待辦細節，依其屬性與優先順序，拆解為完整的工作項目清單。

打造專屬 AI 團隊的力量



精準控制 (Precision Control)
為每項任務精確定義 AI 的能力和行為。



效率提升 (Enhanced Efficiency)
透過一鍵切換角色和自動化工作流程
消除重複設定。

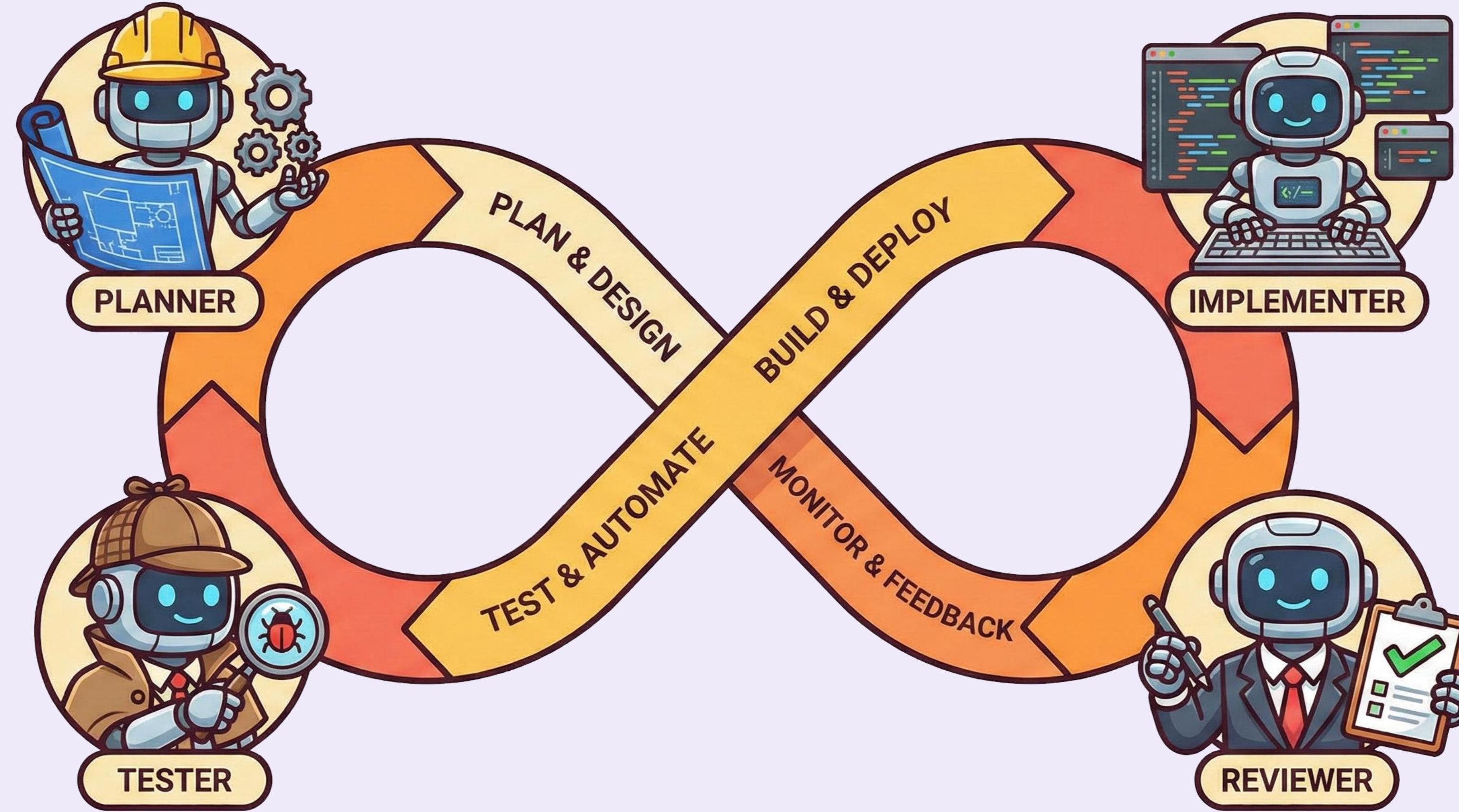


品質一致 (Consistent Quality)
確保 AI 的回應始終符特定任務設定的標準。



團隊協作 (Team Collaboration)
在工作區中共享代理定義，統一團隊
的工作流程和標準。

打造專屬 AI 團隊的力量



“透過自訂代理，我們不再只是使用一個工具，而是在指揮一個由 AI 專家組成的團隊。”

選擇最適合的代理：本地 CLI vs. 雲端 Coding Agent

GitHub Copilot CLI



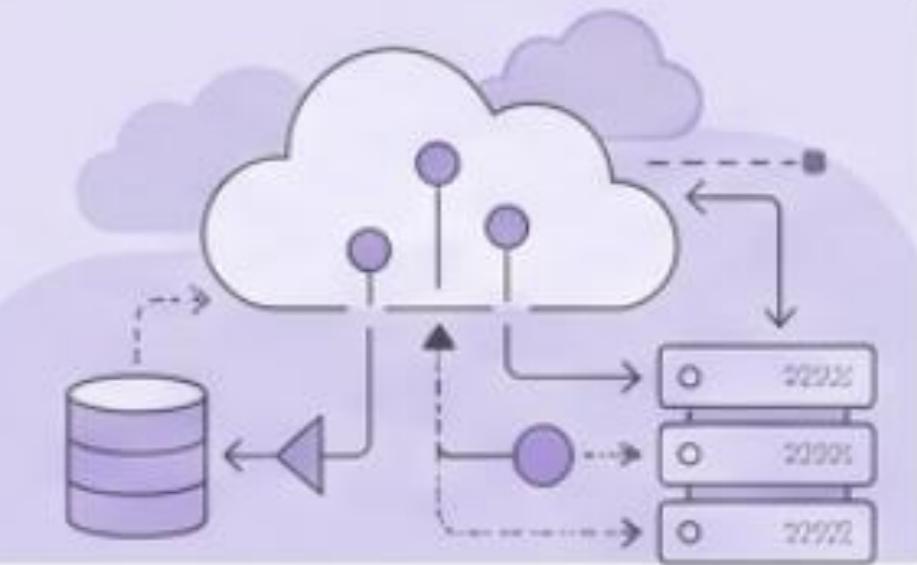
執行環境 (Environment)
本地終端機

最適合 (Best For)
即時互動與迭代修改

核心用途 (Use Case)

- 在本機重構單一檔案
- 執行複雜的 Git 指令
- 快速除錯

GitHub Copilot Coding Agent



執行環境 (Environment)
雲端 (在 GitHub 背景執行)

最適合 (Best For)
非同步、長時間任務

核心用途 (Use Case)

- 為整個專案產生測試
- 修正 Bug 並自動建立 PR
- 處理技術債

Windows PowerShell

Welcome to GitHub Copilot

CLI Version 0.0.365

Version 0.0.365 · Commit 76d0881

Copilot can write, test and debug code right from your terminal. Describe a task to get started or enter ? for help. Copilot uses AI, check for mistakes.

- Logged in as user: timlai-tw
- Connected to GitHub MCP Server

Select Model

Choose the AI model to use for Copilot CLI. The selected model will be persisted and used for future sessions.

Some models are not available due to configured policy.

- 1. Claude Sonnet 4.5 (1x) (default) (current)
- 2. Claude Haiku 4.5 (0.33x)
- 3. Claude Sonnet 4 (1x)
- 4. GPT-5.1 (1x)
- 5. GPT-5.1-Codex-Mini (0.33x)
- 6. GPT-5.1-Codex (1x)
- 7. GPT-5 (1x)
- 8. GPT-5-Mini (0x) 8. GPT-5-Mini (0x)
- 9. GPT-4.1 (0x)
- 10. Gemini 3 Pro (Preview) (1x)
- 11. Cancel (Esc)

Confirm with number keys or ↑↓ keys and Enter, Cancel with Esc

Usage: **copilot** [options] [command]

GitHub Copilot CLI - An AI-powered coding assistant

選項：

--add-dir <directory>	將目錄加入允許的檔案存取清單 (可重複使用)
--additional-mcp-config <json>	額外的 MCP 伺服器設定，可為 JSON 字串或檔案路徑 (以 @ 為前置)
--agent <agent>	指定要使用的自訂 agent (僅在 prompt 模式可用)
--allow-all-paths	停用檔案路徑驗證並允許存取任何路徑
--allow-all-tools	允許所有工具自動執行而不需確認；非互動模式必需 (環境變數：COPilot_ALLOW_ALL)
--allow-tool [tools...]	允許特定工具
--banner	顯示啟動橫幅
--continue	回復最近的工作階段
--deny-tool [tools...]	拒絕特定工具，優先於 --allow-tool 或 --allow-all-tools
--disable-builtin-mcps	停用所有內建的 MCP 伺服器 (目前：github-mcp-server)
--disable-mcp-server <server-name>	停用指定的 MCP 伺服器 (可重複使用)
--disable-parallel-tools-execution	停用工具的平行執行 (LLM 仍可發出平行工具呼叫，但會改為序列執行)
--disallow-temp-dir	防止自動存取系統暫存目錄
--enable-all-github-mcp-tools	啟用所有 GitHub MCP 伺服器工具，而非 CLI 的預設子集
-h, --help	顯示指令說明

(續)

--log-dir <directory>	設定日誌目錄 (預設 : <code>~/.copilot/logs/</code>)
--log-level <level>	設定日誌等級 (可選值 : "none", "error", "warning", "info", "debug", "all", "default")
--model <model>	設定要使用的 AI 模型 (可選值 : " <code>claude-sonnet-4.5</code> ", " <code>claude-haiku-4.5</code> ", " <code>claude-opus-4.5</code> ", " <code>claude-sonnet-4</code> ", " <code>gpt-5</code> ", " <code>gpt-5.1</code> ", " <code>gpt-5.1-codex-mini</code> ", " <code>gpt-5.1-codex</code> ", " <code>gemini-3-pro-preview</code> ", " <code>gpt-5-mini</code> ", " <code>gpt-4.1</code> ")
--no-color	停用所有顏色輸出
--no-custom-instructions	停用從 <code>AGENTS.md</code> 與相關檔案載入自訂指令
-p, --prompt <text>	直接執行提示文字，無需進入互動模式
--resume [sessionId]	從先前的工作階段回復 (可選指定 <code>session ID</code>)
--screen-reader	啟用螢幕閱讀器優化
--stream <mode>	啟用或停用串流模式 (可選值 : "on", "off")
-v, --version	顯示版本資訊

GitHub Copilot CLI 命令說明

互動模式指令：

/add-dir <directory>	將目錄加入允許的檔案存取清單
/agent	瀏覽並選取可用的 agent (如有)
/clear	清除對話紀錄
/cwd [directory]	變更工作目錄或顯示目前目錄
/delegate <prompt>	使用 AI 產生的 PR 將變更委派到遠端版本庫
/exit, /quit	退出 CLI
/share [file gist] [path]	將工作階段分享為 Markdown 檔案或 GitHub gist
/feedback	提供 CLI 意見回饋
/help	顯示互動指令說明
/list-dirs	顯示所有允許的檔案存取目錄
/login	登入 Copilot
/logout	登出 Copilot
/mcp [show add edit delete disable enable] [server-name]	管理 MCP 雖服器設定
/model [model]	選擇要使用的 AI 模型
/reset-allowed-tools	重設允許使用的工具清單
/session	顯示目前 CLI 工作階段資訊
/terminal-setup	設定終端以支援多行輸入 (Shift+Enter 與 Ctrl+Enter)
/theme [show set list] [auto dark light]	檢視或設定終端主題
/usage	顯示工作階段使用量指標與統計
/user [show list switch]	管理 GitHub 使用者清單

實戰場景1：秒速上手新專案

“

剛接手一個陌生專案？GitHub Copilot CLI 可以成為我們的嚮導，快速建立對專案的理解。

探索專案結構與相依性：

> “Explain the project structure and install all the necessary dependencies for me.”

理解關鍵檔案：

> “Summarize the purpose of the ‘Dockerfile’ and ‘package.json’ in this project.”

追蹤變更歷史：

> “Show me the last 5 changes made to the CHANGELOG.md file. Who changed it, when, and what did they do?”

實戰場景2：優化日常開發與 Git 工作流

“

將重複性與複雜的日常任務交給 AI 助手，讓我們專注於更高價值的創造。

程式碼重構與改善：

> “Suggest improvements to ‘content.js’ to make it more readable and efficient.”

文件更新：

> “Rewrite the ‘README.md’ in this project to make it more accessible to newcomers.”

複雜的 Git 操作：

> “Revert the last commit, but leave the changes unstaged so I can review them.
Commit the current changes with an appropriate commit message summarizing the work.”

```
PS D:\SystexSoftware\Work\.NET Conf Taiwan 2025> copilot --agent expert-dotnet-software-engineer -p "對 ./SPO_Example 目  
錄中的專案進行 Code Review，並將結果寫入 ./review.md" --allow-all-tools
```

我將對 ./SPO_Example 目錄中的專案進行全面的 Code Review。讓我先探索專案結構和代碼。

```
✓ $ Get-ChildItem -Path "D:\SystexSoftware\Work\.NET Conf Taiwan 2025\SPO_Example" -Recurse -File  
| Select-Object FullName, Length, LastWriteTime | Format-Table -AutoSize  
└ 72 lines...
```

```
✓ $ Get-Content "D:\SystexSoftware\Work\.NET Conf Taiwan 2025\SPO_Example\Program.cs"  
└ 43 lines...
```

```
✓ $ Get-Content "D:\SystexSoftware\Work\.NET Conf Taiwan 2025\SPO_Example\SPOWebAPI.csproj"  
└ 16 lines...
```

```
✓ $ Get-Content "D:\SystexSoftware\Work\.NET Conf Taiwan  
2025\SPO_Example\Controllers\BulletinController.cs"  
└ 137 lines...
```

```
✓ $ Get-Content "D:\SystexSoftware\Work\.NET Conf Taiwan  
2025\SPO_Example\Services\SharePointService.cs"  
└ 200 lines...
```

```
✓ $ Get-Content "D:\SystexSoftware\Work\.NET Conf Taiwan  
2025\SPO_Example\Services\ISharePointService.cs"  
└ 31 lines...
```

```
✓ $ Get-Content "D:\SystexSoftware\Work\.NET Conf Taiwan 2025\SPO_Example\appsettings.json"  
└ 18 lines...
```

```
✓ $ Get-Content "D:\SystexSoftware\Work\.NET Conf Taiwan 2025\SPO_Example\Models\BulletinItem.cs";  
Write-Host "`n---`n"; Get-Content "D:\SystexSoftware\Work\.NET Conf Taiwan  
2025\SPO_Example\Models\AttachmentItem.cs"; Write-Host "`n---`n"; Get-Content  
"D:\SystexSoftware\Work\.NET Conf Taiwan 2025\SPO_Example\Models\SharePointSettings.cs"  
└ 99 lines...
```

現在讓我創建詳細的 Code Review 報告：

現在讓我創建詳細的 Code Review 報告：

```
✓$ @"
# SPO_Example 專案 Code Review 報告

**Review Date:** 2025-11-26
**Reviewer Perspective:** Anders Hejlsberg, Robert C. Martin, Jez Humble, Kent Beck ...
└ 700 lines...

✓$ Test-Path "D:\SystexSoftware\Work\.NET Conf Taiwan 2025\review.md"
└ 2 lines...
```

完成！我已對 `./SPO_Example` 專案進行了全面的 Code Review，並將詳細報告寫入 `./review.md`。

📊 Review 摘要

整體評級：B- (75/100)

🔺 發現的關鍵問題：

- **Critical 安全漏洞** - appsettings.json 中硬編碼敏感資訊 (TenantId、ClientId、密碼)
- **Critical 程式碼錯誤** - SharePoint API 查詢語法錯誤 (`?\$100` 應為 `?\$top=100`)
- **完全缺乏測試** - 測試覆蓋率 0%
- **缺乏錯誤恢復機制** - 未實作 retry、circuit breaker
- **CORS 設定過於寬鬆** - 允許任何來源存取
- **Token 快取缺失** - 每次都重新申請 access token
- **憑證管理不當** - 未快取、未正確釋放資源

✅ 優點：

- 正確使用依賴注入
- 良好的分層架構
- 完整的 XML 文件註解
- 適當的非同步程式設計

報告中包含詳細的程式碼範例、改進建議、優先順序排序及預估工時。建議立即處理 P0 等級的安全性問題！

📈 Review 摘要

整體評級：B- (75/100)

🚨 發現的關鍵問題：

1. **Critical 安全漏洞** - appsettings.json 中硬編碼敏感資訊 (TenantId、ClientId、密碼)
2. **Critical 程式碼錯誤** - SharePoint API 查詢語法錯誤 (`?\$\$100` 應為 `?&top=100`)
3. **完全缺乏測試** - 測試覆蓋率 0%
4. **缺乏錯誤恢復機制** - 未實作 retry、circuit breaker
5. **CORS 設定過於寬鬆** - 允許任何來源存取
6. **Token 快取缺失** - 每次都重新申請 access token
7. **憑證管理不當** - 未快取、未正確釋放資源

✅ 優點：

- 正確使用依賴注入
- 良好的分層架構
- 完整的 XML 文件註解
- 適當的非同步程式設計

報告中包含詳細的程式碼範例、改進建議、優先順序排序及預估工時。建議立即處理 P0 等級的安全性問題！

```
Total usage est:      1 Premium request
Total duration (API): 1m 54.051s
Total duration (wall): 2m 39.102s
Total code changes:   0 lines added, 0 lines removed
Usage by model:
    claude-sonnet-4.5    125.0k input, 7.9k output, 96.0k cache read (Est. 1 Premium request)
PS D:\SystexSoftware\Work\.NET Conf Taiwan 2025>
```

邁向終極自動化：整合 CI/CD 流程

“將 GitHub Copilot CLI 應用於 CI 流程中，實現自動化的程式碼品質檢查與審查。”

自動化 Code Review 的 CI 步驟

```
- script: |  
  
# 1. 安裝 Copilot CLI  
npm i -g @github/copilot@latest  
  
# 2. 執行 Code Review 代理  
copilot --agent 'code-review' -p "Review the changes in this PR"  
  
displayName: 'Run AI Code Review'  
env:  
# 3. 提供權杖與設定  
GH_TOKEN: $(System.AccessToken)  
COPILOT_ALLOW_ALL: true
```



GH_TOKEN

提供執行所需的 GitHub 權杖。

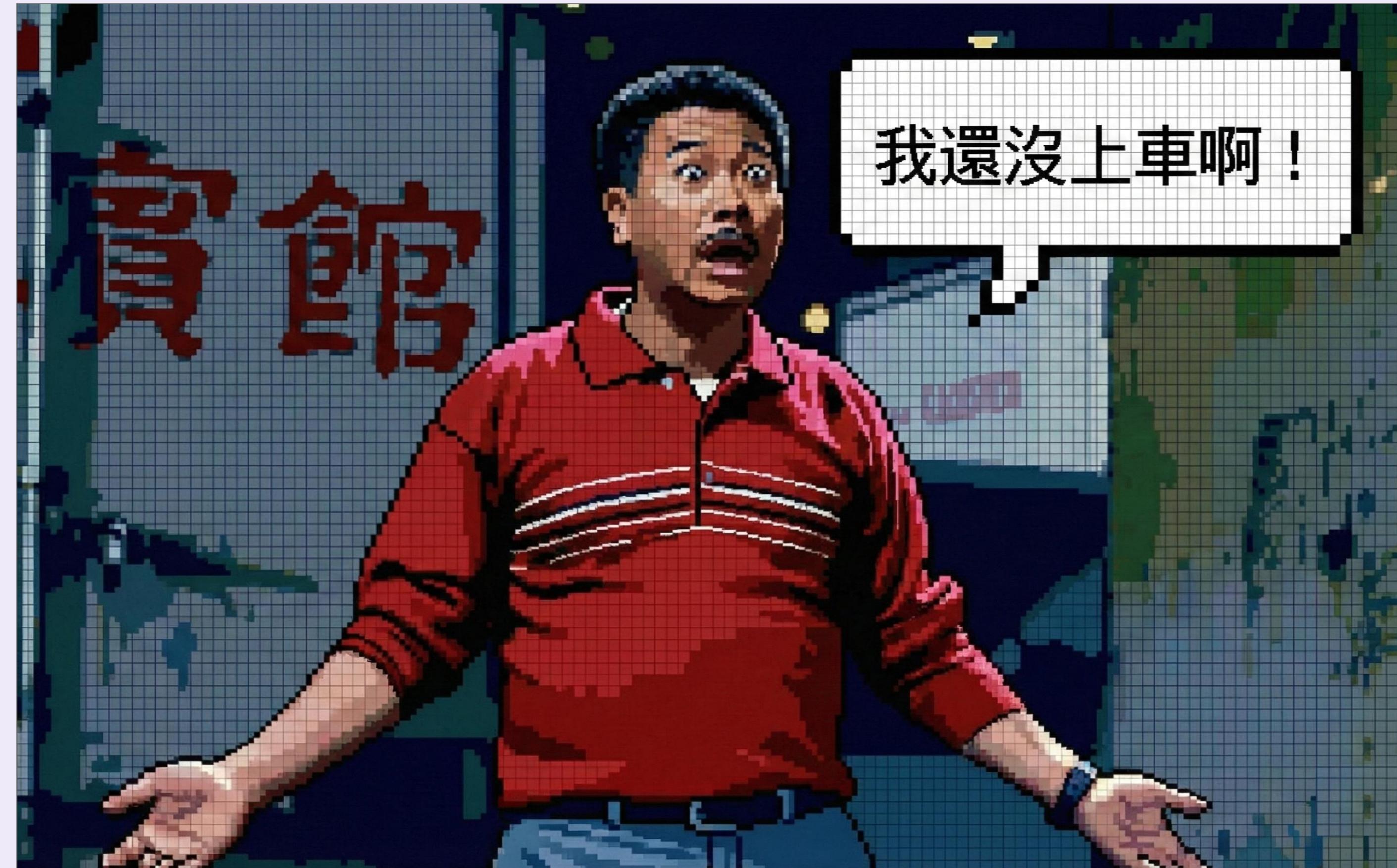


COPILOT_ALLOW_ALL

在受控的 CI 環境中，自動批准工具與路徑權限。

GitHub Copilot CLI

Public Preview



```
> npm i -g @github/copilot@latest
```

Awesome GitHub Copilot Customizations



Awesome Instructions

程式撰寫標準和最佳實踐。



Awesome Prompts

用於程式碼、文件和解決問題的特定任務提示詞。



Awesome Agents

為特定工作流程和工具提供專用代理。



Awesome Collections

按主題分類的提示詞、指令和代理人。

<https://github.com/github/awesome-copilot>

最佳實踐與應避免的模式

✓ 從小處著手，反覆迭代

從最少的上下文開始，依據 AI 的行為逐步增加細節。

✓ 上下文管理

隨著專案演進，定期審視更新相關文件。

✓ 善用 Handoffs

利用代理轉換建立團隊開發工作流程。

✓ 關注點分離

為不同任務使用個別專業代理與聊天會話。

✗ 上下文過多

避免提供過多、無重點的資訊，這會稀釋 AI 的注意力。

✗ 不一致的指導

確保所有文件與架構模式、程式撰寫規範保持一致，不互相抵觸。

✗ 忽視驗證

不要假設 AI 正確理解上下文，在複雜實作前，先確認其理解程度。

✗ 一體適用

不同的專案或團隊成員或任務，可能需要不同的上下文配置。

Thank you !

