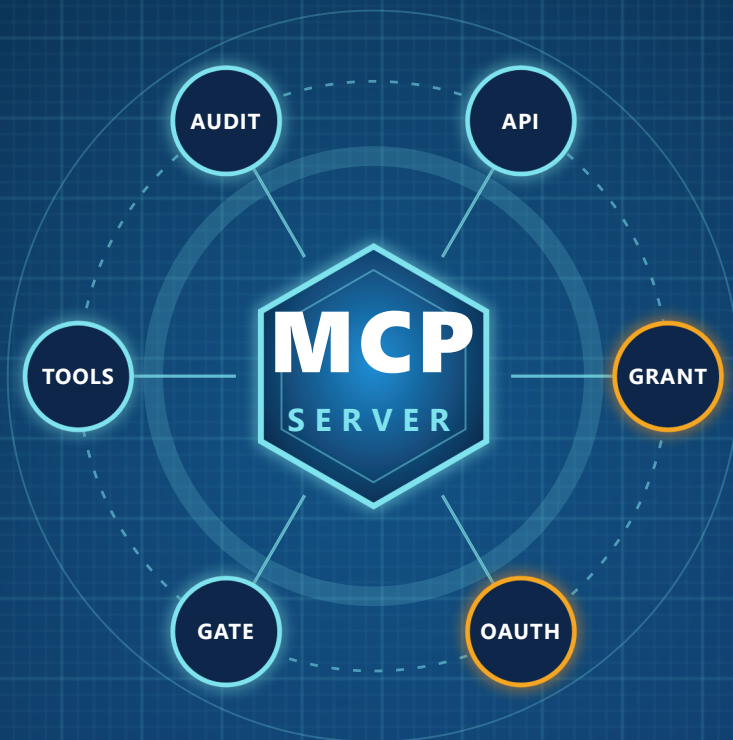




MCP Server 治理平台實戰手冊

30 小時 • 15 章 • 4 篇

ASP.NET Core MVC × EF Core × OAuth × MCP Tools API 上架與授權管理平台專題實作

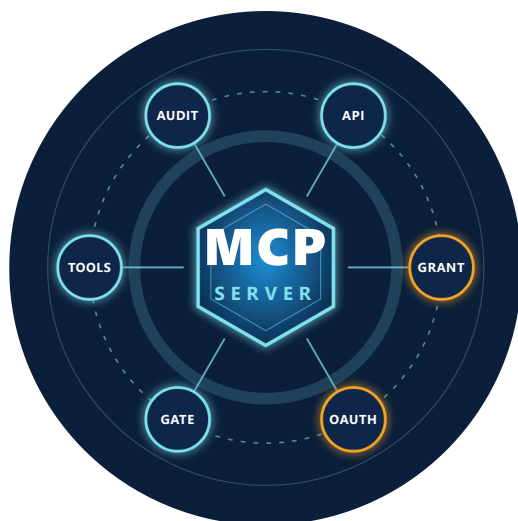
從會員、API 上架審核、使用授權到 MCP 工具呼叫，打造每一步都可檢查、可追查的 API 治理平台

第一篇
平台基礎與會員

第二篇
API 上架與授權審核

第三篇
Gateway、MCP 與 OAuth

第四篇
營運治理與交付



MCP Server 治理平台實戰手冊

30 HRS

ASP.NET Core MVC × EF Core × OAuth × MCP Tools API 上架與授權管理平台專題實作

對應課程	ASP.NET Core MVC × MCP Server API 上架與授權管理平台（30 小時，15 個 Section × 2 小時）
導引案例	MCP Server API 上架與授權管理平台（資料庫 MCPServerManag ）
技術棧	.NET 10、ASP.NET Core MVC、EF Core、SQL Server、Identity、OpenIddict、MCP C# SDK
全書結構	四篇 十五章 + 索引
範例專案	範例/01~15 （每章對應一個範例資料夾，逐章累加）
版次	第 1 版 2026 年 9 月

如何使用本書

本書由 15 個 Section 的課程講義依授課時序編排而成：**一個 Section 就是一章**，章號與 Section 編號相同。每章的範例專案都從上一章複製後往上加，最後累積成一個完整的平台。書中的**預期結果**、**驗收標準**與**反向驗證**標出每一步該看到什麼、以及如何證明權限真的會擋。

每一章都依相同結構撰寫，對應每單元 120 分鐘的課堂時間配置：

- 1. **學習目標**：本章完成後應具備的能力
- 2. **先備知識**：進入本章前需要的概念與前一章的成果
- 3. **觀念說明 (20 分鐘)**：設計理由、流程圖與資料模型
- 4. **教師示範 (25 分鐘)**：講師現場操作的步驟與畫面
- 5. **學員實作 (60 分鐘)**：逐步實作、完整程式碼與預期結果
- 6. **常見錯誤**：實作時最常遇到的問題與修正方式
- 7. **單元驗收 (15 分鐘)**：驗收標準、反向驗證與當堂成果

篇	主題	章
第一篇	平台基礎與會員	第 1～4 章 (S01～S04)
第二篇	API 上架與授權審核	第 5～7 章 (S05～S07)
第三篇	Gateway、MCP 與 OAuth	第 8～11 章 (S08～S11)
第四篇	營運治理與交付	第 12～15 章 (S12～S15)

書末附有**索引**：依關鍵詞列出討論最集中的章節頁碼。

目錄

第一篇 平台基礎與會員	1
第 1 章 專題需求MCP概念與開發環境	2
學習目標	3
先備知識	4
觀念說明 (20 分鐘)	5
教師示範 (25 分鐘)	9
學員實作 (60 分鐘)	11
常見錯誤	16
單元驗收 (15 分鐘)	17
第 2 章 MVC分層與古典RWD版型	18
學習目標	19
先備知識	20
觀念說明 (20 分鐘)	21
教師示範 (25 分鐘)	24
學員實作 (60 分鐘)	28
常見錯誤	33
單元驗收 (15 分鐘)	34
第 3 章 MSSQL資料模型與EFCore	36
學習目標	37
先備知識	38
觀念說明 (20 分鐘)	39
教師示範 (25 分鐘)	42
學員實作 (60 分鐘)	44
常見錯誤	48
單元驗收 (15 分鐘)	49
第 4 章 會員註冊登入與角色授權	50
學習目標	51
先備知識	52
觀念說明 (20 分鐘)	53
教師示範 (25 分鐘)	56
學員實作 (60 分鐘)	60
常見錯誤	64
單元驗收 (15 分鐘)	65
第二篇 API 上架與授權審核	66
第 5 章 API目錄規格與版本管理	67
學習目標	68
先備知識	69
觀念說明 (20 分鐘)	70
教師示範 (25 分鐘)	72
學員實作 (60 分鐘)	76
常見錯誤	81

單元驗收 (15 分鐘)	82
第 6 章 API上架送審與管理者審核	83
學習目標	84
先備知識	85
觀念說明 (20 分鐘)	86
教師示範 (25 分鐘)	90
學員實作 (60 分鐘)	94
常見錯誤	97
單元驗收 (15 分鐘)	98
第 7 章 API使用申請與授權審核	99
學習目標	100
先備知識	101
觀念說明 (20 分鐘)	102
教師示範 (25 分鐘)	104
學員實作 (60 分鐘)	109
常見錯誤	112
單元驗收 (15 分鐘)	113
第三篇 Gateway、MCP 與 OAuth	114
第 8 章 APIGateway與安全呼叫	115
學習目標	116
先備知識	117
觀念說明 (20 分鐘)	118
教師示範 (25 分鐘)	120
學員實作 (60 分鐘)	123
常見錯誤	127
單元驗收 (15 分鐘)	128
第 9 章 MCPServer與Tools映射	129
學習目標	130
先備知識	131
觀念說明 (20 分鐘)	132
教師示範 (25 分鐘)	136
學員實作 (60 分鐘)	139
常見錯誤	145
單元驗收 (15 分鐘)	146
第 10 章 ConnectorOAuth與會員授權	147
學習目標	148
先備知識	149
觀念說明 (20 分鐘)	150
教師示範 (25 分鐘)	153
學員實作 (60 分鐘)	156
常見錯誤	162
單元驗收 (15 分鐘)	163
第 11 章 動態工具權限與呼叫管制	164
學習目標	165
先備知識	166

觀念說明 (20 分鐘)	167
教師示範 (25 分鐘)	170
學員實作 (60 分鐘)	172
常見錯誤	176
單元驗收 (15 分鐘)	177

第四篇 營運治理與交付 178

第 12 章 會員與API全生命週期管理 179

學習目標	180
先備知識	181
觀念說明 (20 分鐘)	182
教師示範 (25 分鐘)	185
學員實作 (60 分鐘)	188
常見錯誤	191
單元驗收 (15 分鐘)	192

第 13 章 數位儀表板與稽核查詢 193

學習目標	194
先備知識	195
觀念說明 (20 分鐘)	196
教師示範 (25 分鐘)	198
學員實作 (60 分鐘)	201
常見錯誤	205
單元驗收 (15 分鐘)	206

第 14 章 權限安全與整合測試 207

學習目標	208
先備知識	209
觀念說明 (20 分鐘)	210
教師示範 (25 分鐘)	212
學員實作 (60 分鐘)	214
常見錯誤	218
單元驗收 (15 分鐘)	219

第 15 章 部署安裝與成果驗收 220

學習目標	221
先備知識	222
觀念說明 (20 分鐘)	223
教師示範 (25 分鐘)	225
學員實作 (60 分鐘)	227
常見錯誤	230
單元驗收 (15 分鐘)	231

索引 232

PART 1

第一篇

平台基礎與會員

S01~S04

先釐清 API 提供者、使用者、管理者與 MCP Client 的責任，再建立 MVC 分層與古典 RWD 版型、以 EF Core 在 SQL Server 上落實可追溯的資料模型，最後用 Identity 完成註冊登入與角色授權。

第 1 章 專題需求MCP概念與開發環境

四種角色責任、MCP 與 REST 的關係、API 上架與使用授權兩條流程、30 小時範圍與版本鎖定、建立 Solution、驗證 SQL Server 連線、三個示範 API 呼叫清單

第 2 章 MVC分層與古典RWD版型

Controller、ViewModel／DTO、Service、資料層分層；Admin Area；Razor Layout、Partial 與 Tag Helper 元件；米白墨綠古銅金古典 RWD；首頁、API 目錄、會員中心與後臺版型

第 3 章 MSSQL資料模型與EFCore

Identity 關聯、API 與版本、申請與 Grant 的區別、外鍵、索引、唯一與 CHECK 約束、rowversion、UTC 時間、EF Core Migration、示範資料匯入

第 4 章 會員註冊登入與角色授權

Identity 密碼雜湊、Cookie、角色、Policy、Area 授權慣例、Anti-forgery、登入錯誤與帳號鎖定、Admin 種子、本人資料範圍

第 1 章

專題需求MCP概念與開發環境

四種角色責任、MCP 與 REST 的關係、API 上架與使用授權兩條流程、30 小時範圍與版本鎖定、建立 Solution、驗證 SQL Server 連線、三個示範 API 呼叫清單



課程時數 2 小時（120 分鐘） 當堂成果 專題規格、流程與可執行網站骨架 範例專案 範例/01_專題需求MCP概念與開發環境

學習目標

上完這個單元，你能夠：

- 用一張表說出**API 提供者**、**API 使用者**、**後臺管理者**、**MCP Client** 四種角色各自能做什麼、不能做什麼。
- 畫出「MCP Client → `/mcp` → API Gateway → 上游 REST API」的呼叫路徑，並指出 `tools/List` 與 `tools/call` 各在哪一段發生。
- 寫出 **API 上架**（草稿 → 送審 → 核准發布／退回 → 下架）與 **使用授權**（申請 → 審核 → 建立 `ApiGrant`）兩條流程的每一個狀態。
- 用 `dotnet new` 建立 `McpPlatform.slnx`，以 `global.json` 鎖定 SDK `10.0.401`、以 `Directory.Packages.props` 鎖定套件版本，並讓 `dotnet build` 顯示 `0` 個錯誤。
- 打開 `http://localhost:5199/SetupCheck`，看到 SQL Server「✓ 正常」、兩個唯讀示範 API「HTTP 200 OK」。
- 用一句話說明並舉證：「**註冊成功不代表取得 API 呼叫權**」。

先備知識

- 這是第一個單元，沒有前一單元的產出。
- 會寫 C# 類別與 `async` / `await`；看得懂 HTML 與 SQL `SELECT`；知道 HTTP `GET` / `POST`、狀態碼 200 / 201 / 400 與 JSON。
- 本機已安裝下列工具（括號內為本單元實測版本）：
- .NET SDK（`10.0.401`）：`dotnet --version` 應印出 `10.0.401`。
- SQL Server 預設執行個體（`16.0.1200.5` Developer Edition）：SSMS 能以 `localhost` 登入。
- Visual Studio 2026，或 VS Code 加裝 C# Dev Kit 與 REST Client 擴充：能開啟 `.http` 檔並按「Send Request」。
- 教室 SQL Server 版本不一定和上面相同；本單元只用到 `SERVERPROPERTY` 與 `DB_ID`，2019 以上都能跑，但**同一班要用同一個版本**（理由見〈30 小時範圍與版本鎖定〉）。

觀念說明（20 分鐘）

為什麼需要這個平台

一家公司已經有「商品查詢」「課程查詢」「服務工單建立」三支 HTTP JSON API，現在想讓 AI 助理（MCP Client）也能呼叫。若直接把網址與金鑰貼給 AI 助理，會立刻出現三個問題：

1. **誰都能叫**：拿到金鑰就能呼叫，無法做到「這位會員可以查商品、但不能建工單」。
2. **無法收回**：員工離職或 API 下架時，流出去的金鑰仍然有效，除非換金鑰讓所有人一起斷線。
3. **無從追查**：出事時查不到是哪位會員、透過哪個 Client、幾點、呼叫了哪個版本。

本課程的平台就是夾在中間的那一層：**提供者**登錄 API、**管理者**審核後才發布、**使用者**申請並獲准後才能呼叫，而且**每一次呼叫都在平台上重新檢查權限並留下紀錄**。

四種角色與責任

下圖回答「一支 API 從登錄到被 AI 呼叫，經過哪些人的手」。注意：**提供者和管理者都是一般會員**，同一個人可以兩者都是。

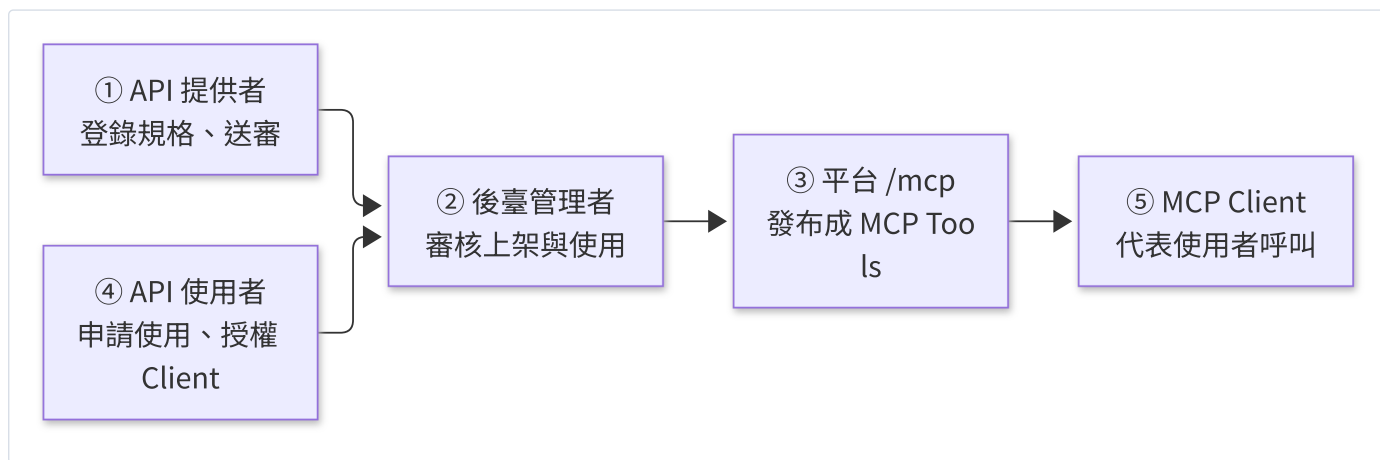


圖 01-1 四種角色在平台上的分工

編號	角色	能做什麼	不能做什麼（為什麼）
①	API 提供者（會員）	建立 API 草稿、填 URL／Method／參數 Schema、建版本、送審	不能核准自己的 API：送審與審核必須分離
②	後臺管理者（Admin）	審核上架與使用申請、停權、下架、撤銷授權	不因管理角色自動取得呼叫權 ；不得核准本人案件
③	平台 /mcp	把已核准版本公開成 MCP Tool，每次呼叫前檢查權限	不執行使用者上傳的程式，只代為呼叫已登錄的 API
④	API 使用者（會員）	瀏覽已發布 API、對某版本提出使用申請、授權自己的 Client	核准前看不到也叫不到該工具
⑤	MCP Client	代表某位使用者列出工具並呼叫	只拿得到那位會員「此刻有效」的權限

MCP 與 REST 的關係

MCP（Model Context Protocol）是讓 AI 應用程式用統一格式「探索並呼叫外部工具」的協定，訊息是 JSON-RPC 2.0（每則有 `method`、`params`、`id`）。本課程用它的 **Streamable HTTP** 傳輸：所有訊息都送到同一個端點 `/mcp`。**REST API** 則是本來就存在的上游服務。兩者的關係是：**MCP 不取代 REST，而是包在 REST 外面、給 AI 用的目錄與呼叫介面**；AI 只需知道工具名稱與輸入 Schema，不必知道上游網址或金鑰。

下圖是一次完整呼叫。注意第 ④ 步：**平台在轉呼叫上游之前一定先檢查權限**。

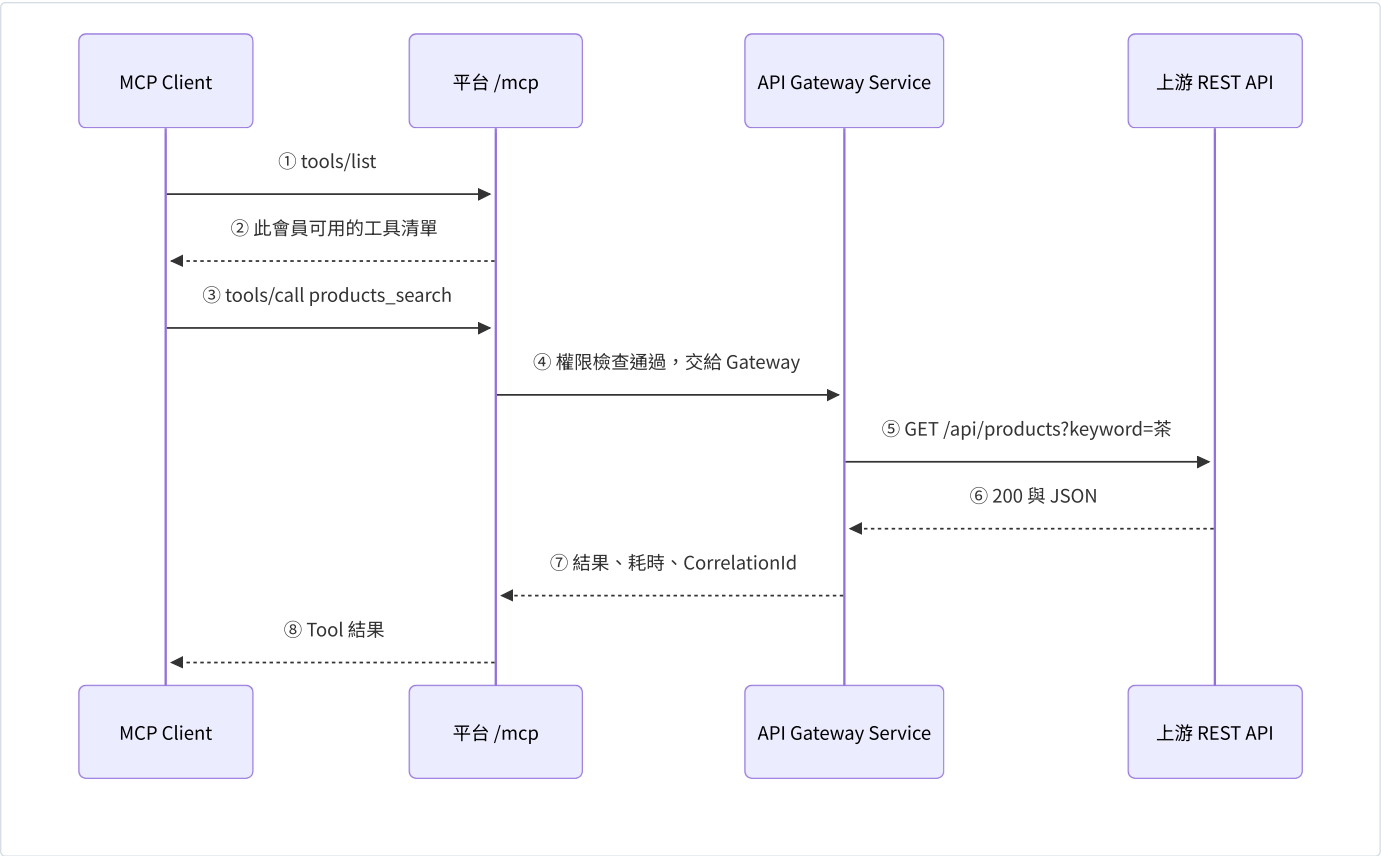


圖 01-2 一次 MCP 工具呼叫如何轉成 REST 呼叫

訊息	送出／拿回的內容	實作單元
① → ②	送 <code>tools/list</code> ；拿回只含這位會員有效授權的工具（名稱、描述、 <code>inputSchema</code> ）	09、11
③	送 <code>tools/call</code> ， <code>params</code> 帶工具名稱與參數 <code>{"keyword": "茶"}</code>	09
④	重新檢查會員、Grant、版本、Scope、額度、參數；不通過就在這裡回錯誤， 不會 打到上游	11
⑤ → ⑥	Gateway 依已核准版本組出網址，只允許登錄過的 Host／Path；拿回上游狀態碼與 JSON	08
⑦ → ⑧	結果連同耗時、 <code>CorrelationId</code> 寫入 <code>ApiCallLogs</code> 後回給 Client	08、13

兩個最常搞混的地方：**Client 快取了工具清單，不代表之後都能呼叫**（清單只是 ② 當下的快照，撤銷後下一次 ③ 會在 ④ 被拒絕）；**網站登入 Cookie 不等於 MCP Access Token**（Cookie 給瀏覽器用，Token 由 OAuth 簽發給 Client 用，Section 10）。

兩條審核流程：API 上架與使用授權

平台上有兩種完全不同的申請。第一條是 API 提供者讓自己的 **API 版本** 出現在目錄上：

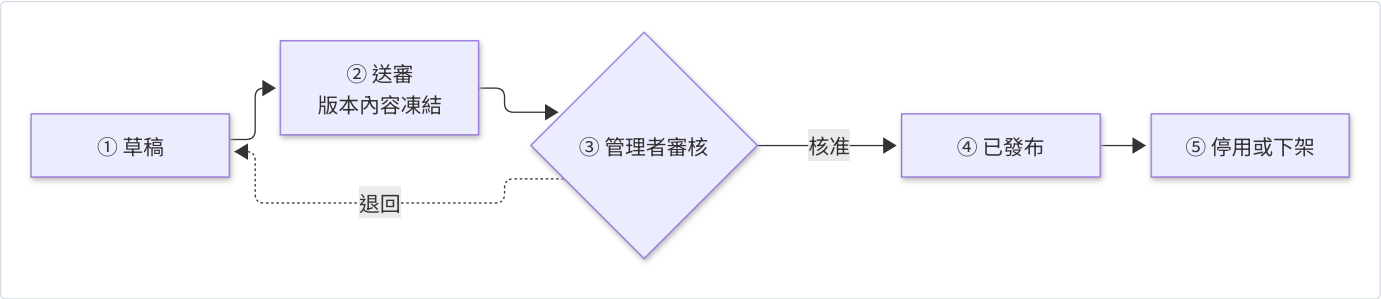


圖 01-3 API 上架流程（對象是 API 版本）

編號	發生什麼事	為什麼這樣設計
① 草稿	可任意修改 URL、Method、參數 Schema	還沒有人依賴它
② 送審	該版本內容 凍結	審的內容必須等於發布的內容，否則可以「審 A 發 B」
③ → ④ 核准	進入可使用目錄，寫一筆審核紀錄（操作者、時間、理由）	只有核准版本能被申請與呼叫
③ → ① 退回	回到草稿修訂後 重新送審 ，原紀錄保留	歷程可追查
⑤ 下架	新呼叫一律拒絕，歷史紀錄不刪除	下架是停止服務，不是抹掉證據

變更 URL、Method、Schema 或權限需求時要**建立新版本並重新送審**，新版本不繼承舊版核准。第二條是 API 使用者取得某個**已發布版本的呼叫權**：

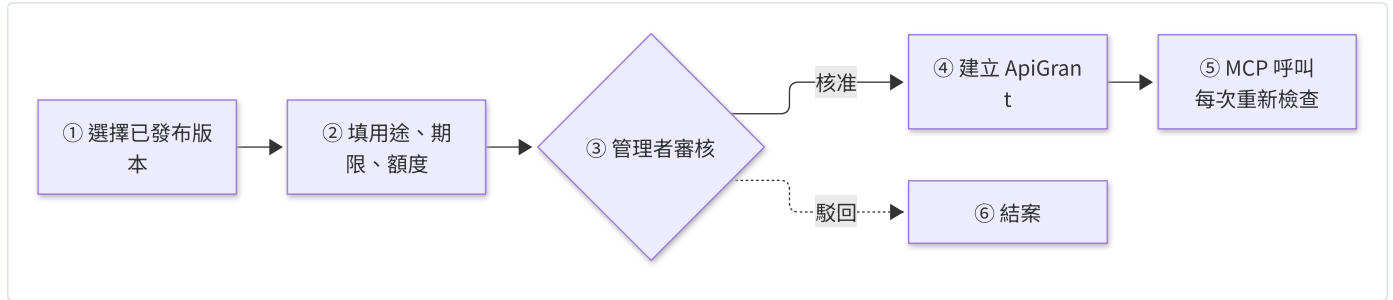


圖 01-4 API 使用授權流程（對象是會員對某版本的使用權）

編號	發生什麼事	資料表 (Section 03 建立)
①	申請一律綁定 明確版本 ，不是綁 API 名稱	ApiVersions
②	記錄用途、希望的期限與額度	ApiAccessRequests
③	管理者核准或駁回，寫審核紀錄	ApprovalRecords
④	建立 使用權 ：有效期間、額度、Scope、狀態	ApiGrants
⑤	每次 tools/call 重新檢查 Grant 是否仍有效	ApiUsageCounters 、 ApiCallLogs
⑥	駁回：不產生任何使用權	ApprovalRecords

「申請被核准」和「授權有效」分開存：**ApiAccessRequests** 記當初的審核結果，永遠不變；**ApiGrants** 記現在能不能用，會因撤銷、到期、額度用完而失效，但核准歷程仍在。

為什麼「註冊成功不代表取得 API 呼叫權」

這句話是本單元的驗收題。下圖是一次 MCP 呼叫要通過的關卡。注意：**註冊只在最左邊**，後面還有三道關卡，**任何一道回答「否」**，呼叫就在那裡被拒絕，不會到達⑤。

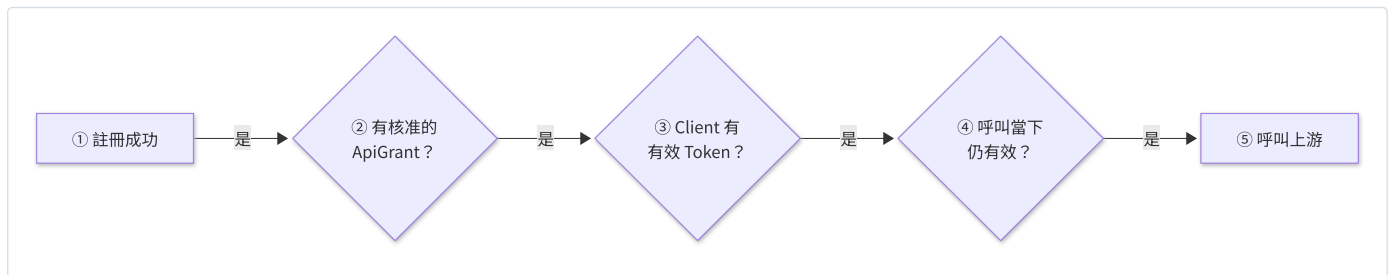


圖 01-5 從註冊到真正呼叫之間的三道關卡

編號	關卡	回答「否」的具體情境（結果：拒絕並記錄一筆授權拒絕事件）	實作單元
①	註冊	註冊只建立一筆 AspNetUsers 會員資料， 不產生任何 Grant	04
②	有沒有 Grant	從沒提出使用申請，或申請被駁回	07
③	Token	Client 沒完成 OAuth 授權、Token 過期、Audience 不是本平台	10
④	即時檢查	Grant 被撤銷、到期、額度用完；會員停權；API 版本下架	11、12
⑤	通過	由 Gateway 代為呼叫上游並記錄耗時與結果	08

驗收時可以這樣回答：「註冊只證明『你是誰』，不證明『你能用什麼』。要經過使用申請、管理者核准並建立 ApiGrant，而且每次呼叫當下 Grant、Token、會員與 API 狀態都還有效，才能呼叫。」

30 小時範圍與版本鎖定

30 小時只夠把一條完整流程做紮實（大綱〈四、專題範圍〉）：

納入核心範圍	不納入（為什麼）
會員前臺、管理後臺、Web API、MCP Server 在同一個網站	多租戶：每張表多一層資料隔離
登錄既有 HTTP JSON API 的呼叫規格，受控的 GET／POST 與明確 Schema	執行使用者上傳的程式、檔案串流、任意腳本：無法控管風險與驗證輸入輸出
一個 API 多個版本，申請與授權都綁版本；三個示範 API	付費計價：金流與對帳不是本課主題
一個課堂 MCP Client 的實測；使用教師提供的 OAuth 骨架	保證任意商用 Connector 可連、從零寫 OAuth 伺服器

版本鎖定是指開課前把 SDK、NuGet 套件、MCP 協定與測試 Client 的版本定下來，整班不混用。MCP C# SDK 與規格都在快速演進，同一段程式在 A 版能編譯、在 B 版可能方法已改名；版本不一致時，老師的示範在學員電腦上會編譯失敗，而且錯誤訊息看不出是版本問題。

鎖定對象	鎖在哪裡	本課程的值
.NET SDK	global.json	10.0.401，rollForward: latestPatch
Microsoft.Data.SqlClient	Directory.Packages.props	7.1.0（本單元加入）
ModelContextProtocol.AspNetCore	Directory.Packages.props	2.2.0（2026-09-26 於 nuget.org 查得的最新穩定版，Section 09 加入）
MCP 授權規格基線	文件記錄	2025-11-25（大綱指定，開課時與 SDK、Client 一併核對）

限制與注意事項

- 雲端 Connector 連不到你電腦的 localhost：本機開發用本機 Client 測；雲端展示需對外可達的 HTTPS 網址（Section 10、15）。
- sa 只用於隔離的教學環境，不當網站會員帳號；正式部署改用最小權限專用帳號。密碼原則若拒絕指定密碼，由教室提供測試密碼。
- 本單元不建資料庫：MCPServerManag 在 Section 03 由 Migration 建立，今天的檢查頁連 master，顯示「尚未建立」是正確結果。

名詞對照

名詞	白話說明	在本課程中是
MCP Client／Connector	會說 MCP 協定、代表使用者呼叫工具的程式	課堂指定的測試 Client
上游 API	平台代為呼叫的原始 REST API	本單元的 McpPlatform.DemoApis
ApiGrant	使用申請核准後產生的使用權，有期限、額度、狀態	ApiGrants 資料表
冪等（Idempotency）	同一請求送一次或多次，效果相同	Idempotency-Key 標頭
User Secrets	開發機上存放機密設定的機制，檔案不在專案裡	ConnectionStrings:DefaultConnection

教師示範（25 分鐘）

示範情境：同時啟動網站與示範上游 API，打開環境檢查頁，證明網站能跑、SQL Server 能連、三支示範 API 列得出來且唯讀的兩支叫得通；再停掉上游，讓學員看到失敗時的畫面。

範例由兩個獨立程式組成，就像真實世界裡平台和 API 提供者的服務架在不同地方：網站 `McpPlatform.Web`（`localhost:5199`）連到 SQL Server 的 `master` 讀版本、並對示範上游 `McpPlatform.DemoApis`（`localhost:5080`）送出探測請求。

畫面背後的檢查邏輯都在 `Services/SetupCheckService.cs`：`CheckSqlServerAsync` 在步驟三逐段說明；`CheckDemoApisAsync` 對呼叫清單中 `Probe` 為 `true` 的端點各送一次 `GET`（3 秒逾時）並計時，2xx 顯示「✓ 正常」、其他狀態碼或連不上顯示「✗ 失敗」，`Probe` 為 `false` 的寫入型端點只列出不呼叫。

示範步驟

1. 開兩個終端機，都切到 `範例/01_專題需求MCP概念與開發環境/McpPlatform`。
2. 第一個執行 `dotnet run --project src/McpPlatform.DemoApis`，等到 `Now listening on: http://localhost:5080`。
3. 第二個執行 `dotnet run --project src/McpPlatform.Web --launch-profile http`，瀏覽器自動打開首頁；點導覽列「環境檢查」。

MCP Server API 平台

首頁

環境檢查

開發環境檢查

重新檢查

檢查時間：2026-09-26 17:26:58（臺灣時間） | 執行環境：Development | 執行階段：.NET 10.0.12

SQL Server

狀態	✓ 正常	已連線；系統資料庫尚未建立（Section 03 以 Migration 建立）。
伺服器	localhost	
版本 / 版別	16.0.1200.5	Developer Edition (64-bit)
系統資料庫	MCPServerManag	尚未建立

示範上游 API 呼叫清單 `http://localhost:5080`

名稱	方法與路徑	用途	檢查結果	耗時
商品查詢 <code>products.search</code>	<code>GET /api/products?</code> <code>keyword=茶</code>	依關鍵字比對商品名稱或分類，回傳價格與庫存。	✓ 正常 HTTP 200 OK	36 ms
課程查詢 <code>courses.search</code>	<code>GET /api/courses?level=</code> <code>進階</code>	依程度（入門 / 中階 / 進階）篩選課程，回傳時數。	✓ 正常 HTTP 200 OK	0 ms
服務工單建立 <code>tickets.create</code>	<code>POST /api/tickets</code>	建立一張服務工單；需 Idempotency-Key 標頭與 <code>confirmed=true</code> 。	— 未呼叫 寫入型 API：不在檢查中自動呼叫，請用 <code>DemoApis.http</code> 手動測試。	—

© 2026 - MCP Server API 平台 - 環境檢查

圖 01-6 環境檢查頁：SQL Server 已連線、兩支唯讀 API 正常

畫面重點：

1. ①標題下的灰字資訊列：檢查時間（UTC 轉臺灣時間）、執行環境 `Development`、執行階段 `.NET 10.0.12`。
2. ②SQL Server「狀態」列：綠色「✓ 正常」與「已連線；系統資料庫尚未建立」。
3. ③「版本／版別」列：`16.0.1200.5 Developer Edition (64-bit)`，由 `SERVERPROPERTY` 讀出，教室機數字可能不同。
4. ④「系統資料庫」列：`MCPServerManag` 旁的黃色「尚未建立」是**正確結果**。
5. ⑤呼叫清單表格：前兩列「✓ 正常／HTTP 200 OK」與耗時毫秒數；第三列灰色「— 未呼叫」，因為是寫入型 API。
6. ⑥右上「重新檢查」：再跑一次全部檢查；檢查只讀不寫，可以一直按。
7. 在第一個終端機按 `Ctrl+C` 停掉示範 API，再按「重新檢查」。頁面約 6 秒後才出現（兩支 API 各等 3 秒逾時）。

開發環境檢查

重新檢查

檢查時間：2026-09-26 17:27:08 (臺灣時間) | 執行環境：Development | 執行階段：.NET 10.0.12

SQL Server

狀態	✔ 正常 已連線；系統資料庫尚未建立 (Section 03 以 Migration 建立) 。
伺服器	localhost
版本 / 版別	16.0.1200.5 Developer Edition (64-bit)
系統資料庫	MCPServerManag 尚未建立

示範上游 API 呼叫清單 http://localhost:5080

名稱	方法與路徑	用途	檢查結果	耗時
商品查詢 products.search	GET /api/products? keyword=茶	依關鍵字比對商品名稱或分類，回傳價格與庫存。	✖ 失敗 無法連線：The request was canceled due to the configured HttpClient.Timeout of 3 seconds elapsing.	—
課程查詢 courses.search	GET /api/courses? level=進階	依程度 (入門 / 中階 / 進階) 篩選課程，回傳時數。	✖ 失敗 無法連線：The request was canceled due to the configured HttpClient.Timeout of 3 seconds elapsing.	—
服務工單建立 tickets.create	POST /api/tickets	建立一張服務工單；需 Idempotency-Key 標頭與 confirmed=true。	— 未呼叫 寫入型 API：不在檢查中自動呼叫，請用 DemoApis.http 手動測試。	—

圖 01-7 停掉示範 API 後的環境檢查頁

畫面重點：

- 1. ①SQL Server 區塊仍是「✔ 正常」：兩項檢查彼此獨立，一項失敗不拖垮另一項。
- 2. ②兩支唯讀 API 變紅色「✖ 失敗」：訊息 無法連線：The request was canceled due to the configured HttpClient.Timeout of 3 seconds elapsing.，「耗時」變成「—」。
- 3. ③服務工單建立仍是「— 未呼叫」：寫入型 API 不論上游狀態都不會自動呼叫。

學員實作（60 分鐘）

範例專案：範例/01_專題需求MCP概念與開發環境/McpPlatform/（第一個單元，從 `dotnet new` 範本開始）。本單元新增方案層的 `global.json`、`Directory.Packages.props`，網站的環境檢查頁（`SetupCheck` 的 Controller、Service、ViewModel、View 與 `Options/DemoApiOptions.cs`），以及教師提供的示範 API 專案；各檔用途見下列步驟。

步驟一 建立 Solution 並鎖定版本（15 分鐘）

目的：建立包含「網站」與「示範 API」兩個專案的方案，並在第一天就鎖住 SDK 與套件版本。

```
# 在你要放專題的資料夾執行（PowerShell 與 Git Bash 相同）
mkdir McpPlatform
cd McpPlatform
dotnet new globaljson --sdk-version 10.0.401 --roll-forward latestPatch
dotnet new sln -n McpPlatform
dotnet new mvc -n McpPlatform.Web -o src/McpPlatform.Web
dotnet new web -n McpPlatform.DemoApis -o src/McpPlatform.DemoApis
dotnet sln add src/McpPlatform.Web src/McpPlatform.DemoApis
```

預期結果：最後兩行是 專案 `src\McpPlatform.Web\McpPlatform.Web.csproj` 已新增至解決方案。與 `DemoApis` 的同一句。`.NET 10` 的 `dotnet new sln` 產生新格式 `McpPlatform.slnx`，不是舊的 `.sln`。

接著在根目錄新建 `Directory.Packages.props`，先只寫下方的 `<PropertyGroup>` 部分，再執行 `dotnet add src/McpPlatform.Web package Microsoft.Data.SqlClient --version 7.1.0`。指令偵測到中央套件管理，會自動補上 `<ItemGroup>` 那段，並在 `McpPlatform.Web.csproj` 只寫 `<PackageReference Include="Microsoft.Data.SqlClient" />`（沒有版本號）。完成後的檔案：

```
<!-- 範例/01_專題需求MCP概念與開發環境/McpPlatform/Directory.Packages.props -->
<Project>
  <!-- 全方案共用的 NuGet 版本清單：每個套件只在這裡寫一次版本號，各 .csproj 只寫套件名稱。 -->
  <PropertyGroup>
    <ManagePackageVersionsCentrally>true</ManagePackageVersionsCentrally>
  </PropertyGroup>
  <ItemGroup>
    <PackageVersion Include="Microsoft.Data.SqlClient" Version="7.1.0" />
  </ItemGroup>
</Project>
```

程式說明（兩個版本鎖定檔）：

檔案與鍵	值	說明
<code>global.json</code> <code>sdk.version</code>	<code>10.0.401</code>	本方案要用的 SDK；電腦上沒有可接受的版本時，所有 <code>dotnet</code> 指令直接失敗並提示
<code>global.json</code> <code>sdk.rollForward</code>	<code>latestPatch</code>	只接受 <code>10.0.4xx</code> 中 ≥ 401 的修補版；寫成 <code>latestMajor</code> 會讓 <code>.NET 11</code> SDK 也被採用，等於沒鎖
<code>ManagePackageVersionsCentrally</code>	<code>true</code>	開啟中央套件管理；之後 <code>.csproj</code> 的 <code>PackageReference</code> 寫 <code>Version</code> 會編譯錯誤 <code>NU1008</code>
<code>PackageVersion</code>	<code>Microsoft.Data.SqlClient</code> <code>7.1.0</code>	整個方案只有這裡決定版本號

預期結果：`dotnet build McpPlatform.slnx` 印出 建置成功。、0 個警告、0 個錯誤。

常見改錯：先 `dotnet add package` 才建 props 檔——版本號留在 `.csproj`，建好 props 後出現 `NU1008`。把 `.csproj` 裡的 `Version="..."` 刪掉、改寫到 props 檔即可。

步驟二 固定埠號並啟動 MVC 網站（10 分鐘）

目的：網站固定 `5199`、示範 API 固定 `5080`，全班的網址與設定檔才能一致。

範本的埠號是隨機的（例如 `5086`）。在 `src/McpPlatform.Web/Properties/LaunchSettings.json` 把 `http` 設定檔的 `applicationUrl` 改成 `http://localhost:5199`（`https` 設定檔改成 `https://localhost:7199;http://localhost:5199`）；在 `src/McpPlatform.DemoApis/Properties/LaunchSettings.json` 只留 `http` 設定檔，`applicationUrl` 改成 `http://localhost:5080`、`launchBrowser` 改成 `false`（上游沒有畫面）。

環境檢查頁的 `SetupCheckController`、`SetupCheckService`、`SetupCheckViewModel`、`DemoApiOptions`、`Views/SetupCheck/Index.cshtml` 與導覽列連結直接從範例複製（Controller 只做一件事：呼叫 `SetupCheckService.RunAsync` 並把結果交給 View）。然後啟動：`dotnet run --project src/McpPlatform.Web --launch-`

profile http。

預期結果：終端機出現 `Now listening on: http://localhost:5199`，瀏覽器打開首頁。此時點「環境檢查」，SQL Server 會顯示紅色 `SQL 錯誤 18456：使用者 'sa' 的登入失敗。`——還沒設定密碼，這是預期的，下一步修好。

步驟三 驗證 SQL Server 連線（20 分鐘）

目的：把連線字串放進 User Secrets，讓網站連上 SQL Server，並確認 `MCPServerManag` 目前「尚未建立」。

`appsettings.Development.json` 只放佔位值（會提交到 Git）：`"DefaultConnection": "Server=localhost;Database=MCPServerManag;User Id=sa;Password=請用 user-secrets 設定;TrustServerCertificate=True"`。真正的密碼用 User Secrets 覆蓋：

```
# 在 McpPlatform/ 根目錄執行
dotnet user-secrets init --project src/McpPlatform.Web
dotnet user-secrets set "ConnectionStrings:DefaultConnection" "Server=localhost;Database=MCPServerManag;User Id=sa;Password=<請填入教室提供的 sa 密碼>;TrustServerCertificate=True" --project src/McpPlatform.Web
```

預期結果：`Successfully saved ConnectionStrings:DefaultConnection to the secret store.`；`McpPlatform.Web.csproj` 多一行 `<UserSecretsId>`（每人 GUID 不同）。機密存在 `%APPDATA%\Microsoft\UserSecrets\<UserSecretsId>\secrets.json`，不在專案裡，不會被 `git add` 帶走；`Development` 環境下它優先於 `appsettings.Development.json`。

連線字串片段	說明
<code>Server=localhost</code>	本機預設執行個體（埠 1433）；具名執行個體寫 <code>localhost\名稱</code>
<code>Database=MCPServerManag</code>	照大綱拼寫，結尾是 <code>Manag</code> 不是 <code>Manager</code>
<code>TrustServerCertificate=True</code>	本機 SQL Server 用自簽憑證；少了它， <code>Microsoft.Data.SqlClient</code> 預設的加密連線會因憑證不受信任而失敗。正式環境改用有效憑證並拿掉

檢查程式是 `Services/SetupCheckService.cs` 的 `CheckSqlServerAsync`，分兩段看。前半段準備連線字串：

```
// 範例/01_專題需求MCP概念與開發環境/McpPlatform/src/McpPlatform.Web/Services/SetupCheckService.cs (CheckSqlServerAsync 前半)
public async Task<SqlServerCheckResult> CheckSqlServerAsync(CancellationToken ct)
{
    var connectionString = configuration.GetConnectionString("DefaultConnection"); // ①
    if (string.IsNullOrEmpty(connectionString))
        return new() { Status = CheckStatus.Failed, Message = "尚未設定 ConnectionStrings:DefaultConnection (請用 dotnet user-secrets set 設定)。" };

    var builder = new SqlConnectionStringBuilder(connectionString);
    var targetDatabase = builder.InitialCatalog; // ② 記下目標：MCPServerManag
    builder.InitialCatalog = "master"; // ③ 改連 master
    builder.ConnectTimeout = 5; // ④ 最多等 5 秒
```

後半段（同一方法，接上段）連線、查詢並處理錯誤：

```
// 範例/01_專題需求MCP概念與開發環境/McpPlatform/src/McpPlatform.Web/Services/SetupCheckService.cs (CheckSqlServerAsync 後半)
try
{
    await using var connection = new SqlConnection(builder.ConnectionString);
    await connection.OpenAsync(ct);

    await using var command = new SqlCommand(
        """
        SELECT CAST(SERVERPROPERTY('ProductVersion') AS nvarchar(128)),
               CAST(SERVERPROPERTY('Edition') AS nvarchar(128)),
               DB_ID(@db)
        """, connection);
    command.Parameters.Add(new SqlParameter("@db", System.Data.SqlDbType.NVarChar, 128) { Value = targetDatabase }); // ⑥

    await using var reader = await command.ExecuteReaderAsync(ct);
    await reader.ReadAsync(ct);
    var exists = !reader.IsDBNull(2); // ⑥ DB_ID 查不到會回 NULL

    return new()
    {
        Status = CheckStatus.Ok,
        Message = exists ? "已連線，系統資料庫已存在。" : "已連線；系統資料庫尚未建立（Section 03 以 Migration 建立）。",
        DataSource = builder.DataSource,
        ProductVersion = reader.GetString(0),
        Edition = reader.GetString(1),
        TargetDatabase = targetDatabase,
        TargetDatabaseExists = exists
    };
}
catch (SqlException ex) // ⑦
{
    return new() { Status = CheckStatus.Failed, Message = $"SQL 錯誤 {ex.Number} : {ex.Message}", DataSource = builder.DataSource, TargetDatabase = targetDatabase };
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	讀 <code>ConnectionStrings:DefaultConnection</code> ，沒設定就回傳提示	給一句能照做的提示，而不是丟例外
② ③	記下 <code>MCPServerManag</code> ，改建 <code>master</code>	直接連不存在的資料庫會得到錯誤 4060，分不出是密碼錯還是資料庫不存在
④	連線逾時 5 秒（預設 15 秒）	SQL Server 沒開時檢查頁很快就回報
⑤	用參數 <code>@db</code> 傳資料庫名稱	不把字串拼進 SQL，養成防 SQL Injection 的習慣
⑥	<code>DB_ID(...)</code> 回 <code>NULL</code> 表示資料庫不存在	只讀系統資訊，不建立任何東西
⑦	只攔 <code>SqlException</code> ，連同錯誤代碼 <code>ex.Number</code> 顯示	代碼好查：18456 登入失敗、4060 資料庫不存在、-2 逾時

重新啟動網站（`Ctrl+C` 後再 `dotnet run`）後打開環境檢查頁。

預期結果（圖 01-6 的②③④）：

狀態	✓ 正常	已連線；系統資料庫尚未建立（Section 03 以 Migration 建立）。
版本／版別	16.0.1200.5	Developer Edition (64-bit)
系統資料庫	MCPServerManag	尚未建立

常見改錯：改完 User Secrets 沒重啟網站，頁面仍是 18456。本單元實測就遇到：改完重新整理仍失敗，重啟後才正常。

步驟四 建立三個示範 API 的呼叫清單（15 分鐘）

目的：啟動教師提供的三個示範上游 API，並用兩種形式寫下呼叫清單：給人手動測試的 `DemoApis.http`，與給網站讀取的 `appsettings.json` `DemoApis` 節點。

名稱	方法與路徑	輸入	成功回應
商品查詢	GET <code>/api/products</code>	<code>keyword</code> （選填）比對名稱或分類	200
課程查詢	GET <code>/api/courses</code>	<code>level</code> （選填）入門／中階／進階	200
服務工單 建立	POST <code>/api/tickets</code>	標頭 <code>Idempotency-Key</code> （必填）；本文 <code>subject</code> （1~100 字）、 <code>description</code> （≤1000 字）、 <code>confirmed: true</code>	第一次 201；同 Key 重送 200 並回同一張工單

服務工單建立是寫入型 API，有兩道保護：沒有 `Idempotency-Key` 或 `confirmed` 不是 `true` 都回 `400`。前者讓網路斷線後重送不會多建一張工單（幕等），後者是日後 AI 透過 MCP 呼叫時「先問使用者、確認後才執行」的依據。實作在 `src/McpPlatform.DemoApis/Program.cs` 與 `TicketStore.cs`，Section 08 講 Gateway 寫入幕等時再逐行說明。

給人用的呼叫清單 `src/McpPlatform.DemoApis/DemoApis.http`（節錄；完整檔另有「0. 健康檢查」「2. 課程查詢」「3-1 同 Key 重送」「3-2 沒有確認」四個請求，見範例檔）：

```
@DemoApis = http://localhost:5080

### 1. 商品查詢（唯讀）：keyword 比對商品名稱或分類
GET {{DemoApis}}/api/products?keyword=茶

### 3. 服務工單建立（寫入）：必須帶 Idempotency-Key，且 confirmed=true
POST {{DemoApis}}/api/tickets
Content-Type: application/json
Idempotency-Key: demo-0001

{
  "subject": "茶葉禮盒缺件",
  "description": "訂單 A-20260926-01 少了一盒紅玉紅茶",
  "confirmed": true
}
```

給網站用的呼叫清單：在 `src/McpPlatform.Web/appsettings.json` 最外層加入 `DemoApis` 節點。三筆端點格式相同，下面列出第一筆：

```
"DemoApis": {
  "BaseUrl": "http://localhost:5080",
  "Endpoints": [
    {
      "Key": "products.search",
      "Name": "商品查詢",
      "Method": "GET",
      "Path": "/api/products?keyword=茶",
      "Description": "依關鍵字比對商品名稱或分類，回傳價格與庫存。",
      "Probe": true
    }
  ]
}
```

鍵	說明
BaseUrl	必須等於示範 API 的 <code>applicationUrl</code> ；寫錯時兩支唯讀 API 逾時失敗（圖 01-7）
Key	穩定識別碼，例如 <code>products.search</code> ；Section 09 映射 MCP Tool 名稱時以它為基礎， 上線後不要改
Method / Path	目前用於顯示與 <code>GET</code> 探測；Section 08 起 Gateway 會嚴格比對
Probe	環境檢查時是否實際呼叫；工單建立這筆寫 <code>false</code> ， 寫入型一律 false

網站在 `Program.cs` 用 `builder.Services.Configure<DemoApiOptions>(... GetSection("DemoApis"))` 綁定這個節點，並以 `AddHttpClient("DemoApis", ...)` 註冊一個 `BaseAddress` 取自 `DemoApis:BaseUrl`、逾時 3 秒的具名 `HttpClient`；`BaseUrl` 缺漏時網站**一啟動就丟例外** `appsettings.json` 缺少 `DemoApis:BaseUrl`，不會等到打開檢查頁才發現。

啟動示範 API（另開終端機）：`dotnet run --project src/McpPlatform.DemoApis`，再依序送出 `.http` 的請求。

預期結果（本單元實測）：

```
1. 商品查詢 → 200 {"count":3,"items":[{"sku":"P-1001","name":"阿里山高山烏龍茶 150g", ...}]}
```

```
2. 課程查詢 → 200 {"count":2,"items":[{"code":"C-301","title":"MCP Server 與 AI 工具串接", ...}]}
```

```
3. 工單建立 → 201 {"ticketNo":"T-00001","subject":"茶葉禮盒缺件", ... }
```

```
3-1. 同 Key 重送 → 200, ticketNo 仍是 T-00001
```

```
3-2. confirmed=false → 400 "尚未確認：請在本文送出 \"confirmed\": true 才會建立工單。"
```

最後重新整理環境檢查頁，應與圖 01-6 一致。

檢查點

做到這裡，你應該可以：

- ☐ `dotnet build McpPlatform.slnx` 顯示 0 個警告、0 個錯誤；`McpPlatform.Web.csproj` 的 `PackageReference` 沒有 `Version`
- ☐ `http://localhost:5199/SetupCheck` 的 SQL Server「✓ 正常」、`MCPServerManag`「尚未建立」

- 同頁商品查詢、課程查詢「✓ 正常／HTTP 200 OK」，服務工單建立「— 未呼叫」
- 同一把 `Idempotency-Key` 送兩次工單：第一次 `201`、第二次 `200` 且 `ticketNo` 相同
- `git status` 看不到任何含真實密碼的檔案

常見錯誤

以下訊息都是撰寫本單元時實際遇到的原文。

症狀（錯誤訊息原文）	原因	處理
檢查頁： <code>SQL 錯誤 18456：使用者 'sa' 的登入失敗。</code>	密碼錯，或仍是 <code>appsettings.Development.json</code> 的佔位值	重跑步驟三的 <code>dotnet user-secrets set</code> ，用 <code>dotnet user-secrets list --project src/McpPlatform.Web</code> 確認，然後重啟網站
檢查頁： <code>無法連線：The request was canceled due to the configured HttpClient.Timeout of 3 seconds elapsing.</code>	示範 API 沒啟動，或 <code>DemoApis:BaseUrl</code> 與其 <code>applicationUrl</code> 埠號不一致	另開終端機執行 <code>dotnet run --project src/McpPlatform.DemoApis</code> ；確認兩邊都是 <code>5080</code>
啟動時： <code>System.IO.IOException: Failed to bind to address http://127.0.0.1:5199: address already in use.</code>	上一次的 <code>dotnet run</code> 沒關，仍佔著 <code>5199</code>	回原終端機按 <code>Ctrl+C</code> ；找不到就用 PowerShell <code>Get-NetTCPConnection -LocalPort 5199 -State Listen</code> 查 <code>OwningProcess</code> 再結束它
啟動時警告： <code>Failed to determine the https port for redirect.</code>	用 <code>http</code> 設定檔啟動， <code>UseHttpsRedirection</code> 找不到 HTTPS 埠	開發時可忽略；要消除就改用 <code>--launch-profile https</code> 。部署一律走 HTTPS（Section 15）
Git Bash 用 <code>curl -d '{"subject": "茶葉..."}'</code> 呼叫工單，回 <code>400</code> 且含 <code>Cannot transcode invalid UTF-8 JSON text to UTF-16 string.</code>	繁中 Windows 的 Git Bash 把命令列中文以 Big5（cp950）送出，不是 UTF-8	改用 <code>.http</code> 檔；或把 JSON 存成 UTF-8 檔案，用 <code>curl --data-binary @t1.json</code> 送出

單元驗收（15 分鐘）

驗收標準：網站可啟動；學員能說明「註冊成功不代表取得 API 呼叫權」。

驗收方式：

1. **網站可啟動**：`dotnet build McpPlatform.slnx`（0 個錯誤）→ 啟動示範 API → 啟動網站 → 開 `http://localhost:5199/SetupCheck`，畫面與圖 01-6 一致。
2. **能說明「註冊成功不代表取得 API 呼叫權」**：對著圖 01-5 說明三個重點——(a) 註冊只建立會員資料、不產生 `ApiGrant`；(b) 要經使用申請與管理者核准才建立 `ApiGrant`；(c) 有 Grant 之後，每次 `tools/call` 當下仍要重新檢查 Token、Grant 期限與額度、會員與 API 狀態。只講到 (a) 不算通過。
3. **反向驗證一**：停掉示範 API 後按「重新檢查」，兩支唯讀 API 必須變紅色「✕ 失敗」、SQL Server 仍「✓ 正常」（圖 01-7）；整頁錯誤或卡住不動就是錯誤處理沒寫對。
4. **反向驗證二**：`.http` 把工單的 `confirmed` 改成 `false` 送出，必須得到 `400`「尚未確認」，而且**不會**產生新的 `ticketNo`。

回顧三句話：平台夾在 MCP Client 與上游 REST API 中間，**每次呼叫都重新檢查權限**；上架審「版本能不能公開」、使用授權審「這位會員能不能用這個版本」，兩條流程兩種資料；版本第一天就鎖住。下一單元（Section 02）把網站拆成 Controller／Service／資料層，並換上米白底、墨綠導覽列、古銅金線框的古典 RWD 版型。

延伸閱讀

- [官方 MCP C# SDK](#) —— Section 09 建立 `/mcp` 時使用的套件原始碼與範例。
- [MCP 2025-11-25 授權規範](#) —— 本課程授權設計的參考基線，Section 10 會用到。
- [ASP.NET Core Identity](#) —— Section 04 會員註冊登入的依據。
- [.NET 官方支援政策](#) —— 為什麼選 .NET 10 LTS。

第 2 章

MVC分層與古典RWD版型

Controller、ViewModel／DTO、Service、資料層分層；Admin Area；Razor Layout、Partial 與 Tag Helper 元件；米白墨綠古銅金古典 RWD；首頁、API 目錄、會員中心與後臺版型



課程時數 2 小時（120 分鐘） 當堂成果 前臺、會員中心與後臺共用版型 範例專案 範例/02_MVC分層與古典RWD版型

學習目標

上完這個單元，你能夠：

- 把網站拆成 `McpPlatform.Web`、`McpPlatform.Core`、`McpPlatform.Infrastructure` 三個專案，並說出為什麼 **Core** 不可以參考任何其他專案。
- 讓一個請求依序經過 Controller → Service → Repository 介面 → Repository 實作，並指出「前臺只看得到已發布 API」這條規則寫在哪一層。
- 說明 DTO 與 ViewModel 的差別，並用「表單 ViewModel 沒有 `OwnerId`、`Status` 欄位」防止使用者偷塞欄位。
- 建立 `Admin` 與 `Member` 兩個 Area，讓 `/Admin/Apis`、`/Member/Apis/Create` 正確導到各自的 Controller 與 View。
- 用一個全站 `_Layout`、一個 `_AreaLayout`、一個 Partial `_ApiCard` 與一個 Tag Helper `<status-badge>`，組出前臺、會員中心、後臺三種頁面。
- 在 390px、768px、1440px 三種寬度驗證：頁面不出現水平捲軸、表單不溢出、後臺表格在平板水平捲動、在手機改成卡片。

先備知識

- 完成 Section 01：範例/01_專題需求MCP概念與開發環境/McpPlatform 能 `dotnet build`（0 個錯誤），`/SetupCheck` 能打開。本單元的範例是把它整份複製後往上加。
- 知道 Bootstrap 5 的格線（`row`、`col-12 col-md-6`）與斷點名稱：`md` 是 $\geq 768\text{px}$ 、`lg` 是 $\geq 992\text{px}$ 。
- 會用瀏覽器開發人員工具（F12）的「裝置工具列」切換視窗寬度。
- C# `interface` 與建構子注入（DI）的基本用法；Section 01 的 `SetupCheckController` 就是這樣拿到 Service 的。

觀念說明（20 分鐘）

為什麼要分層

Section 01 所有程式都在 `McpPlatform.Web` 裡，這在只有一頁時沒問題。但這個平台之後會有四個入口讀同一份 API 資料：前臺目錄、會員中心、管理後臺，以及 Section 09 的 MCP `/mcp` 端點。假設「未發布的 API 不能出現在公開目錄」這條規則直接寫在 `CatalogController` 裡，Section 09 寫 MCP 工具清單時就得再寫一次；只要有一處忘了寫，草稿或已下架的 API 就會外流給 AI 用戶端，而且編譯器不會提醒你。

分層的目的就是讓每條規則只寫一次、放在固定的位置：Controller 只處理 HTTP（讀參數、選畫面），業務規則放 Service，資料怎麼存取放 Repository。

四層怎麼分工

下圖是本單元「打開 API 目錄」這一個請求走過的路。注意：每個箭頭都是呼叫一個介面，箭頭左邊的元件不知道右邊的實作是誰。

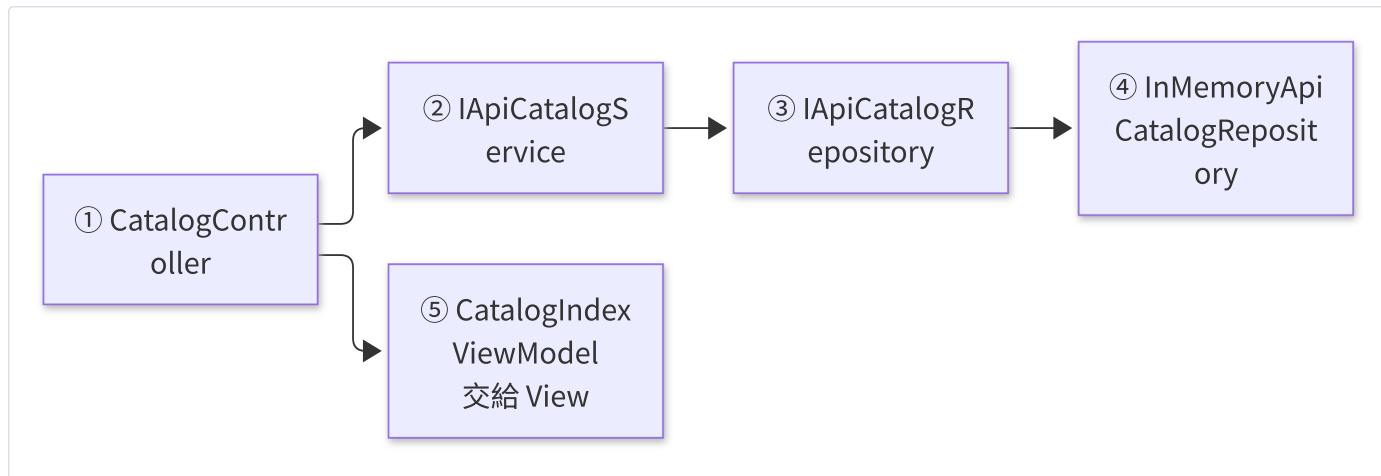


圖 02-1 一個「查詢 API 目錄」請求經過的四層

編號	層	在哪個專案	只做什麼	不可以做什麼
①	Controller	<code>McpPlatform.Web</code>	讀網址參數、呼叫 Service、把結果包成 ViewModel、選 View	寫業務規則、直接查資料
②	Service	<code>McpPlatform.Core</code>	業務規則：公開目錄強制加上「狀態＝已發布」的條件	知道資料存在記憶體還是 SQL Server
③	Repository 介面	<code>McpPlatform.Core</code>	定義「要什麼資料」： <code>SearchAsync(ApiSearchCriteria, ct)</code>	有任何實作程式
④	Repository 實作	<code>McpPlatform.Infrastructure</code>	依條件取資料。本單元用記憶體假資料，Section 03 換成 EF Core	決定誰能看什麼（那是 ② 的事）
⑤	ViewModel	<code>McpPlatform.Web</code>	帶著畫面才需要的東西：搜尋框的值、分類下拉選單	被 Service 或 Repository 使用

最重要的效果在 Section 03：把 ④ 換成 EF Core 版時，① ② ③ 一行都不用改，只改 `Program.cs` 的一行註冊。

專案參考的方向

分層不是靠「資料夾分得整齊」，而是靠專案參考讓錯誤的呼叫編譯不過。下圖的箭頭是「參考」（A → B 表示 A 可以使用 B 的類別）。

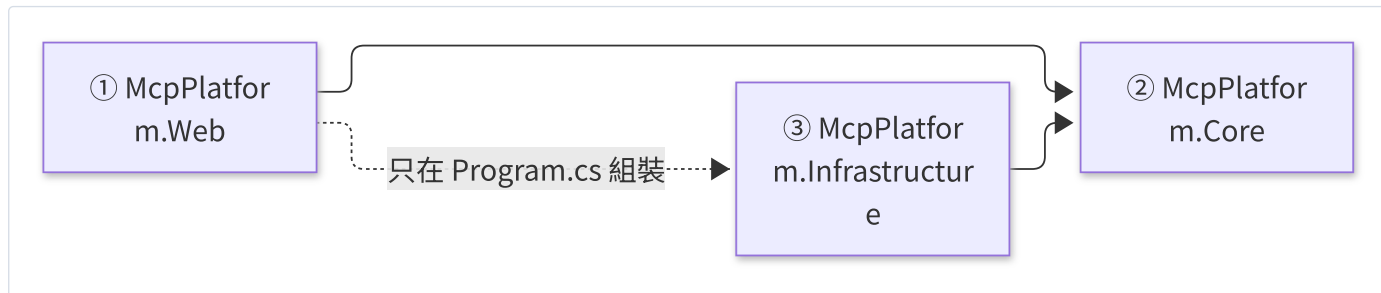


圖 02-2 三個專案的參考方向

編號	專案	參考誰	為什麼
①	<code>McpPlatform.Web</code>	Core、Infrastructure	需要 Core 的介面與 DTO；參考 Infrastructure 只為了在 <code>Program.cs</code> 註冊實作
②	<code>McpPlatform.Core</code>	無	業務規則不能依賴資料庫或網頁技術；少了這條限制，Service 就可能直接 <code>new SqlConnection</code>
③	<code>McpPlatform.Infrastructure</code>	Core	要實作 Core 定義的 <code>IApiCatalogRepository</code>

如果有人 Core 裡寫 `using McpPlatform.Infrastructure;`，會直接編譯失敗——這正是我們要的：違規在編譯時就被擋下，不必靠 Code Review 發現。

DTO 與 ViewModel

	DTO (<code>ApiSummaryDto</code>)	ViewModel (<code>CatalogIndexViewModel</code> 、 <code>ApiDraftFormViewModel</code>)
是什麼	跨層傳遞的純資料	某一個畫面專用的資料模型
放在哪	<code>McpPlatform.Core/Catalog/</code>	<code>McpPlatform.Web/ViewModels/</code>
誰用	Service、Repository、Controller	只有 Controller 與 View
內容	API 名稱、版本、端點、狀態、更新時間	搜尋關鍵字、下拉選單、表單驗證屬性 <code>[Required]</code>

表單 ViewModel 還有一個安全用途：防止過度提交 (overposting)。ASP.NET Core 的模型繫結會把表單欄位依名稱填進物件；如果表單直接繫結到一個有 `OwnerId`、`Status` 的類別，攻擊者只要在送出時多塞一個 `Status=Published`，就可能讓草稿跳過審核直接發布。`ApiDraftFormViewModel` 根本沒有這兩個屬性，多塞的欄位就沒有地方可以放，自然被忽略。擁有者由登入身分決定 (Section 04)，狀態一律從草稿開始 (Section 05)。

Area：把會員中心與後臺分區

Area 是 ASP.NET Core MVC 把一組 Controller 與 View 分成獨立區塊的機制。本單元建立兩個：

Area	網址開頭	資料夾	之後由誰保護
Member	<code>/Member/ ...</code>	<code>Areas/Member/Controllers</code> 、 <code>Areas/Member/Views</code>	Section 04：必須登入
Admin	<code>/Admin/ ...</code>	<code>Areas/Admin/Controllers</code> 、 <code>Areas/Admin/Views</code>	Section 04：必須有 Admin 角色

Area 要成立需要三件事同時存在：Controller 上的 `[Area("Admin")]`、`Program.cs` 的 `{area:exists}` 路由、`Areas/<名稱>/Views/<Controller>/` 的 View。少任何一件都會 404 或找不到 View (見〈常見錯誤〉)。

分區不等於保護。本單元還沒有登入，任何人都能打開 `/Admin`。導覽列裡隱藏「管理後臺」連結也不算授權控制——知道網址的人直接輸入就進得去。真正的保護在 Section 04 用 `[Authorize(Roles = "Admin")]` 加在 Controller 上。

Razor Layout 與元件

下圖是頁面怎麼被「一層一層包起來」。注意：後臺頁面被包兩層，前臺頁面只包一層，兩者的頁首頁尾因此完全一致。

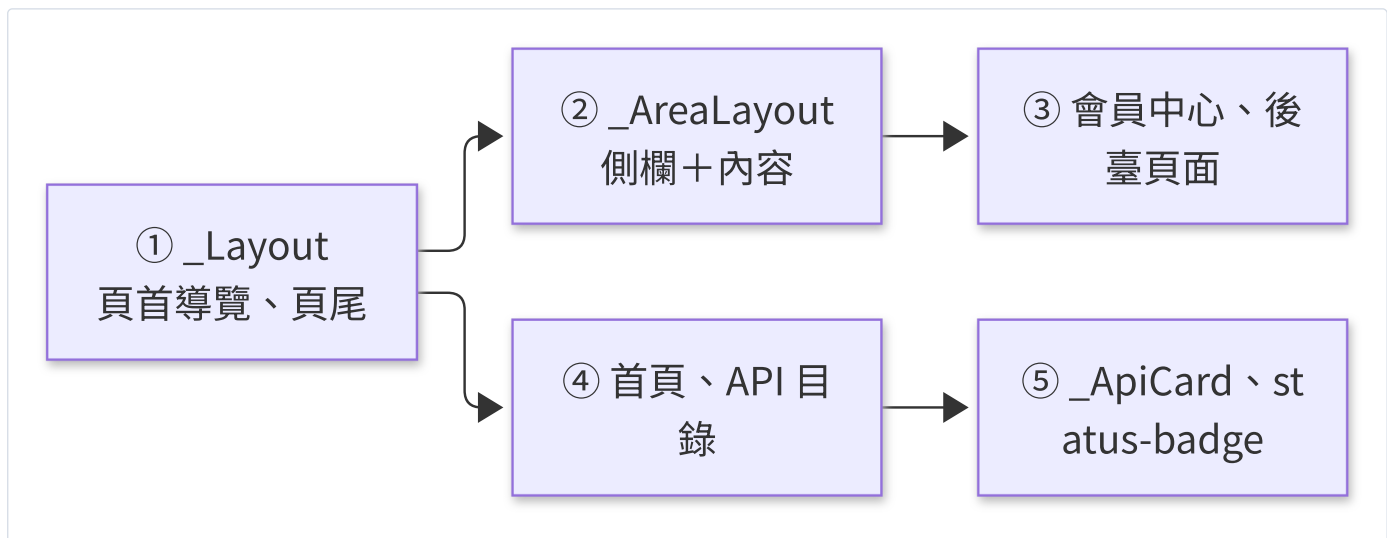


圖 02-3 版型的巢狀關係與共用元件

編號	元件	檔案	用途
①	全站 Layout	Views/Shared/_Layout.cshtml	墨綠導覽列、主要內容區 @RenderBody()、頁尾、共用 JS
②	巢狀 Layout	Views/Shared/_AreaLayout.cshtml	自己再指定 Layout = "_Layout"；依目前 Area 放入 _MemberNav 或 _AdminNav 側欄
③	Area 頁面	Areas/*/Views/**	由 Areas/_ViewStart.cshtml 統一指定使用 ②
④	前臺頁面	Views/Home、Views/Catalog	由 Views/_ViewStart.cshtml 使用 ①
⑤	元件	_ApiClient.cshtml (Partial View)、StatusBadgeTagHelper (Tag Helper)	同一個 API 卡片、狀態標籤在首頁、目錄、會員中心、後臺重複使用

Partial View 與 Tag Helper 怎麼選：一塊 HTML 片段、需要的資料就是一個 Model → Partial（<partial name="_ApiClient" model="api" />）；要像 HTML 標籤一樣帶屬性、輸出邏輯需要 C# → Tag Helper（<status-badge status="@api.Status" />）。需要自己查資料的元件用 View Component，本單元沒有用到。

古典 RWD 的設計規則

RWD（Responsive Web Design，響應式設計）是同一份 HTML 依視窗寬度調整版面。本專題的古典風格（大綱〈十一〉）落實成下面幾條具體規則：

項目	規則	為什麼
色彩	米白 #f7f2e7 底、墨綠 #1f4d3a 主色、深棕 #5a3e2b 次要、古銅金 #a88450 只用於框線	古銅金與米白底的對比不足，拿來當文字會讀不清楚
字型	標題明體（PMingLiU 等，--serif），內文黑體（Microsoft JhengHei，--sans）	明體營造古典感，黑體在小字與表單比較清楚；全用系統字型，離線也能顯示
390px 手機	導覽列收合成三；側欄變成可左右滑的一列；後臺表格改成卡片	手機寬度放不下 6 欄表格
768px 平板	導覽列仍收合；後臺表格放在 .table-responsive 裡水平捲動	表格比容器寬時只捲表格本身，不讓整頁出現水平捲軸
1440px 桌面	導覽列展開；側欄在左、內容在右（col-lg-3 / col-lg-9）	大螢幕一次看完
長內容	網址用 overflow-wrap: anywhere 在任意字元換行	大綱要求長 URL、JSON 需可控換行
狀態	圖示＋文字（✓ 已發布、⌚ 審核中）	色弱或黑白列印時仍分得出狀態
鍵盤	所有可操作元素有 3px 的焦點外框；首頁有「跳到主要內容」連結	大綱要求保留鍵盤焦點

限制與注意事項

- 本單元的資料是模擬的：InMemoryApiCatalogRepository 寫死 8 筆 API，重啟網站不會變。Section 03 才接上 SQL Server。
- 會員身分是寫死的：DemoCurrentUser 一律代表王小明（member-a）。Section 04 加入登入後刪除這個類別。
- 「新增 API 草稿」只做表單驗證，通過後顯示提示、不存檔；存檔在 Section 05。
- 系統字型因作業系統而異：沒有 PMingLiU 的 macOS 會退回其他襯線字型，版面仍正常，只是字形不同。

名詞對照

名詞	白話說明	在本課程中是
DTO	跨層傳遞、只放資料的物件	ApiSummaryDto
ViewModel	某個畫面專用的資料模型	CatalogIndexViewModel、ApiDraftFormViewModel
Repository	把「資料從哪來」包起來的類別	IApiCatalogRepository
過度提交（overposting）	表單多塞欄位，讓模型繫結改到不該改的屬性	用沒有 OwnerId / Status 的 ViewModel 防範
Area	把一組 Controller 與 View 分區的機制	Member、Admin
Tag Helper	用類似 HTML 標籤的寫法呼叫 C# 產生 HTML	<status-badge>

教師示範（25 分鐘）

示範情境：打開首頁、API 目錄、會員中心、後臺四種頁面，證明它們**共用同一套版型**；再把視窗從 1440px 縮到 768px、390px，證明**版面跟著改變但不會破版**。最後走讀「公開目錄只看得到已發布 API」這條規則所在的 Service。

示範步驟

1. 在 `範例/02_MVC分層與古典RWD版型/McpPlatform` 執行 `dotnet run --project src/McpPlatform.Web --launch-profile http`，瀏覽器打開 `http://localhost:5199`，按 F12 開啟「裝置工具列」，寬度設 1440。



圖 02-4 前臺首頁 (1440px)

畫面重點：

1. ①**墨綠導覽列與古銅金雙線下框**：`_Layout` 的 `.classic-navbar`；右側「示範身分：王小明」提醒這還不是真的登入。
2. ②**主標題用明體**：`--serif` 字型堆疊；框線是 `.classic-frame` 的外細框加內側淡金線。
3. ③「**最新發布**」三張卡片：同一個 `_ApiCard` Partial；只出現「✓ 已發布」的 API，草稿與審核中的不會出現在前臺。
4. 點導覽列「管理後臺」→ 側欄「API 管理」。

管理後臺

總覽

API 管理

上架審核 (Section 06)

會員管理 (Section 12)

API 管理

全部狀態 · 共 8 筆。

名稱	提供者	版本	端點	狀態	更新時間
匯率換算 金融	王小明	v1	GET https://fx.example.test/api/rates/convert	草稿	2026-09-26 16:00
天氣預報查詢 生活	王小明	v2	GET https://weather.example.test/api/v2/forecast/36-hours?locationCode=TPE&elementName=Wx,PoP,MinT,MaxT&format=json	審核中	2026-09-25 16:00
門市庫存查詢 零售	林小華	v1	GET https://store.example.test/api/v1/stores/{storeId}/inventory	✓ 已發布	2026-09-24 16:00
包裹追蹤 物流	林小華	v1	GET https://parcel.example.test/api/tracking/{trackingNo}	✗ 已退回	2026-09-23 16:00
商品查詢 零售	教師示範帳號	v1	GET http://localhost:5080/api/products	✓ 已發布	2026-09-20 16:00
課程查詢 教育	教師示範帳號	v1	GET http://localhost:5080/api/courses	✓ 已發布	2026-09-20 16:00
服務工單建立 客服	教師示範帳號	v1	POST http://localhost:5080/api/tickets	✓ 已發布	2026-09-20 16:00
會員點數查詢 (舊版) 零售	王小明	v1	GET https://points.example.test/api/v1/points	✗ 已下架	2026-09-10 16:00

圖 02-5 後臺 API 管理 (1440px)

畫面重點：

- ①左側「管理後臺」側欄：_AreaLayout 依 Area 放入 _AdminNav；目前頁「API 管理」加粗（aria-current="page"）。灰字的「上架審核（Section 06）」是之後才會做的功能。
- ②表格共 8 筆、五種狀態都有：後臺呼叫的是 ListForAdminAsync，不限狀態；和前臺只有 4 筆已發布形成對比。
- ③端點欄的長網址：天氣預報查詢的網址在欄內斷行，沒有把表格撐寬。
- ④更新時間：資料存的是 UTC 08:00，畫面經 ToTaipeiString() 顯示成臺灣時間 16:00。
5. 裝置工具列寬度改成 768，再改成 390，同一頁重新整理。



圖 02-6 同一頁在 768px 與 390px 的樣子

畫面重點：

- ①兩種寬度的導覽列都收合成三： `navbar-expand-lg` 表示 992px 以上才展開。
- ②側欄變成一列可左右滑動的連結：寬度 < 992px 時 `.area-nav-list` 改為 `display: flex` 加 `overflow-x: auto`，右邊被切掉的「會員管理」手指一滑就出現。
- ③左圖 768px 的表格最右欄「更新時間」被切掉：表格實際寬 874px、容器只有 694px，只有表格本身可以左右捲動，整頁沒有水平捲軸。
- ④右圖 390px 表格消失、改成一筆一張卡片：表格外層是 `d-none d-md-block`（768 以上才顯示），卡片清單是 `d-md-none`（768 以下才顯示），兩者用的是同一份 `Model.Items`。

程式碼走讀：規則寫在 Service

公開目錄「只看得到已發布」、會員中心「只看得到自己的」，這兩條規則都在 `McpPlatform.Core/Catalog/ApiCatalogService.cs`。前臺 `CatalogController`、首頁 `HomeController` 都呼叫 `SearchPublishedAsync`，所以規則只寫這一次。

```
// 範例/02_MVC分層與古典RWD版型/McpPlatform/src/McpPlatform.Core/Catalog/ApiCatalogService.cs
public sealed class ApiCatalogService(IApiCatalogRepository repository) : IApiCatalogService
{
    public Task<IReadOnlyList<ApiSummaryDto>> SearchPublishedAsync(string? keyword, string? category, CancellationToken ct)
    {
        var criteria = new ApiSearchCriteria(
            Keyword: Normalize(keyword), // ①
            Category: Normalize(category),
            Status: ApiVersionStatus.Published); // ② 公開目錄強制只看已發布
        return repository.SearchAsync(criteria, ct);
    }

    public Task<IReadOnlyList<ApiSummaryDto>> ListOwnedAsync(string ownerId, CancellationToken ct)
    {
        ArgumentException.ThrowIfNullOrWhiteSpace(ownerId); // ③ 沒有會員 Id 不查
        return repository.SearchAsync(new ApiSearchCriteria(OwnerId: ownerId), ct);
    }

    public Task<IReadOnlyList<ApiSummaryDto>> ListForAdminAsync(ApiVersionStatus? status, CancellationToken ct) =>
        repository.SearchAsync(new ApiSearchCriteria(Status: status), ct);

    public Task<IReadOnlyList<string>> ListCategoriesAsync(CancellationToken ct) =>
        repository.ListCategoriesAsync(ct);

    /// <summary>空白字串視為「不限制」，並去掉前後空白。</summary>
    private static string? Normalize(string? value) =>
        string.IsNullOrEmpty(value) ? null : value.Trim();
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	關鍵字、分類先經過 <code>Normalize</code> ：空白字串變 <code>null</code> 、去掉前後空白	搜尋框什麼都沒填送出的是 <code>""</code> ；不轉成 <code>null</code> 的話，Repository 會去找「名稱包含空字串」而產生多餘條件
②	查詢條件寫死 <code>Status: ApiVersionStatus.Published</code>	呼叫端沒有任何參數可以改變它；前臺網址加 <code>?status=Draft</code> 也不會生效，因為 <code>CatalogController</code> 根本不收這個參數
③	沒給會員 Id 就丟 <code>ArgumentException</code>	如果 <code>ownerId</code> 是空的卻繼續查， <code>ApiSearchCriteria(OwnerId: null)</code> 等於不限擁有者，會把所有人的 API 都列出來

執行結果（本單元實測）：`/Catalog` 共 4 筆、`/Catalog?q=查詢` 共 3 筆、`/Catalog?category=零售` 共 2 筆（已下架的「會員點數查詢（舊版）」不在內）；`/Admin/Apis` 共 8 筆；`/Admin/Apis?status=Submitted` 共 1 筆。

學員實作（60 分鐘）

範例專案：範例/02_MVC分層與古典RWD版型/McpPlatform/（從 範例/01_專題需求MCP概念與開發環境/McpPlatform/ 複製後往上加）。本單元新增 `McpPlatform.Core`、`McpPlatform.Infrastructure` 兩個專案；網站新增 `ViewModels/`、`TagHelpers/`、`Formatting/`、`Areas/Member`、`Areas/Admin`、`Controllers/CatalogController.cs`、`Views/Catalog/`、`Views/Shared/_AreaLayout.cshtml`、`_MemberNav`、`_AdminNav`、`_ApiCard`；修改 `Program.cs`、`_Layout.cshtml`、`_ViewImports.cshtml`、`Views/Home/Index.cshtml`、`wwwroot/css/site.css`，刪除範本的 `Privacy.cshtml` 與 `_Layout.cshtml.css`。

步驟一 建立 Core 與 Infrastructure 專案（10 分鐘）

目的：建立兩個類別庫，並用專案參考把圖 02-2 的方向固定下來。

```
# 在 McpPlatform/ 根目錄執行
dotnet new classlib -n McpPlatform.Core -o src/McpPlatform.Core
dotnet new classlib -n McpPlatform.Infrastructure -o src/McpPlatform.Infrastructure
dotnet sln add src/McpPlatform.Core src/McpPlatform.Infrastructure
dotnet add src/McpPlatform.Infrastructure reference src/McpPlatform.Core
dotnet add src/McpPlatform.Web reference src/McpPlatform.Core src/McpPlatform.Infrastructure
```

預期結果：最後三行輸出三句 參考 `..\McpPlatform.Core\McpPlatform.Core.csproj` 已新增至專案。（第三句是 Infrastructure）。刪掉兩個專案範本附的 `Class1.cs`。不要對 Core 執行任何 `dotnet add reference`。

步驟二 資料層與分層註冊（15 分鐘）

目的：在 Core 定義 DTO、查詢條件、`IApiCatalogRepository`、`IApiCatalogService` 與 `ApiCatalogService`（見教師示範的程式走讀）；在 Infrastructure 實作記憶體版 Repository；在 `Program.cs` 把介面和實作接起來。

Core 與 Infrastructure 的檔案直接從範例複製：

檔案	內容重點
<code>Core/Catalog/ApiVersionStatus.cs</code>	五種狀態： <code>Draft</code> 、 <code>Submitted</code> 、 <code>Published</code> 、 <code>Rejected</code> 、 <code>Retired</code> ，對應 Section 01 圖 01-3
<code>Core/Catalog/ApiSummaryDto.cs</code>	<code>ApiSummaryDto</code> （11 個欄位的 <code>record</code> ）與 <code>ApiSearchCriteria</code> （四個可為 <code>null</code> 的條件， <code>null</code> 表示不限制）
<code>Core/Catalog/IApiCatalogRepository.cs</code>	<code>SearchAsync(criteria, ct)</code> 、 <code>ListCategoriesAsync(ct)</code>
<code>Infrastructure/Catalog/InMemoryApiCatalogRepository.cs</code>	8 筆模擬資料：3 支教師示範 API、王小明 3 支、林小華 2 支；五種狀態都有；一筆刻意超長的網址用來測試換行

接著在 `Program.cs` 註冊兩層，並加上 Area 路由：

```
// 範例/02_MVC分層與古典RWD版型/McpPlatform/src/McpPlatform.Web/Program.cs（分層註冊與 Area 路由）
builder.Services.AddScoped<IApiCatalogService, ApiCatalogService>(); // ① 業務層
builder.Services.AddSingleton<IApiCatalogRepository, InMemoryApiCatalogRepository>(); // ② 資料層（Section 03 換成 EF Core）

//（省略：DemoApiOptions、HttpClient、SetupCheckService 註冊與中介軟體，與 Section 01 相同）

// Area 路由放在 default 前面：它比較具體（多了 area 片段），微軟文件建議先註冊較具體的路由。
app.MapControllerRoute(
    name: "areas",
    pattern: "{area:exists}/{controller=Home}/{action=Index}/{id?}" // ③
).WithStaticAssets();
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	要 <code>IApiCatalogService</code> 時給 <code>ApiCatalogService</code> ，每個 HTTP 請求一個新實例（Scoped）	Section 03 的 EF Core <code>DbContext</code> 也是 Scoped，業務層跟著用 Scoped 最不容易出錯
②	要 <code>IApiCatalogRepository</code> 時給記憶體版，整個網站共用一個實例（Singleton）	假資料是唯讀的靜態清單，不需要每次重建；Section 03 換成 EF Core 版時改成 <code>AddScoped</code>
③	<code>{area:exists}</code> 限制第一段必須是真的存在的 Area 名稱	<code>/Admin/Api</code> 會被解讀成 <code>area=Admin, controller=Api</code> ； <code>/Catalog</code> 不是 Area，交給下面的 default 路由

預期結果：`dotnet build McpPlatform.slnx` 顯示 0 個錯誤。此時還沒有 View，先不要執行。

步驟三 Controller、ViewModel 與 Area（15 分鐘）

目的：建立前臺目錄、會員中心、後臺的 Controller，每個都只呼叫 Service、再把 DTO 包成 ViewModel。

前臺目錄的 Controller 是最典型的寫法：

```
// 範例/02_MVC分層與古典RWD版型/McpPlatform/src/McpPlatform.Web/Controllers/CatalogController.cs
public sealed class CatalogController(IApiCatalogService catalog) : Controller
{
    public async Task<IActionResult> Index(string? q, string? category, CancellationToken ct) // ①
    {
        var items = await catalog.SearchPublishedAsync(q, category, ct); // ②
        var categories = await catalog.ListCategoriesAsync(ct);

        var model = new CatalogIndexViewModel // ③
        {
            Keyword = q,
            Category = category,
            Categories = categories.Select(c => new SelectListItem(c, c, c == category)).ToList(),
            Items = items
        };
        return View(model); // ④
    }
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	只收 <code>q</code> 與 <code>category</code> 兩個查詢字串參數	參數清單就是「呼叫端能控制什麼」；沒有 <code>status</code> 參數，就不可能從網址看到草稿
②	呼叫 Service 的公開目錄方法，不碰 Repository	Controller 只認得 Core 的介面；這一行換成 Repository 會繞過 ② 的規則
③	把 DTO 清單與畫面需要的下拉選單、搜尋值包成 ViewModel	<code>SelectListItem</code> 是 MVC 的類別，只能出現在 Web 層，不能放進 Core 的 DTO
④	依慣例找 <code>Views/Catalog/Index.cshtml</code>	前臺 View 使用 <code>Views/_ViewStart.cshtml</code> 指定的 <code>_Layout</code>

會員中心的表單 POST 示範 ViewModel 的驗證與防護：

```
// 範例/02_MVC分層與古典RWD版型/McpPlatform/src/McpPlatform.Web/Areas/Member/Controllers/ApiController.cs (POST 動作)
[HttpPost]
[ValidateAntiForgeryToken] // ①
public async Task<IActionResult> Create(ApiDraftFormViewModel form, CancellationToken ct) // ②
{
    if (!ModelState.IsValid) // ③
    {
        await LoadCategoriesAsync(ct);
        return View(form);
    }

    TempData["Notice"] = $"{form.Name} 表單驗證通過。草稿存檔將在 Section 05 實作。"; // ④
    return RedirectToAction("Index", "Home", new { area = "Member" }); // ⑤
}
```

(類別上方有 `[Area("Member")]`，完整檔案見範例。)

程式說明：

標號	做了什麼	為什麼這樣寫
①	驗證防偽權杖 (anti-forgery token)	View 的 <code><form asp-action></code> Tag Helper 會自動帶一個隱藏欄位；別的网站偽造的表單沒有它，送出直接 400 (CSRF 防護)
②	參數型別是 <code>ApiDraftFormViewModel</code>	它沒有 <code>OwnerId</code> 、 <code>Status</code> ，表單多塞這兩個欄位也無處可放 (防過度提交)
③	依 ViewModel 的 <code>[Required]</code> 、 <code>[Url]</code> 、 <code>[RegularExpression]</code> 檢查；不過就帶著使用者輸入回到原頁	驗證在伺服器端一定執行，不能只靠瀏覽器端 JavaScript
④	用 <code>TempData</code> 存一次性提示	<code>TempData</code> 在下一個請求讀出後就消失，重新整理不會重複顯示
⑤	成功後重新導向 (302) 到會員中心	Post-Redirect-Get：使用者按 F5 不會再送一次表單

預期結果：`dotnet build` 顯示 0 個錯誤。如果 View 裡出現 CS0246 找不到類型或命名空間名稱 'AdminApiListViewModel'，是 `Areas/_ViewImports.cshtml` 還沒建 (下一步處理)。

常見改錯：Area 裡的 Controller 忘了加 `[Area("Admin")]` —— 它會被當成一般 Controller，`/Admin/Apis` 變成 404，`/Apis` 反而找得到但找不到 View（見〈常見錯誤〉）。

步驟四 共用版型與元件（20 分鐘）

目的：讓前臺、會員中心、後臺共用頁首頁尾，會員中心與後臺再共用側欄版面；做出 `<status-badge>` 元件；套上古典 RWD 樣式。

先建兩個 Area 共用的檔案：`Areas/_ViewStart.cshtml` 內容是 `@{ Layout = "_AreaLayout"; }`；`Areas/_ViewImports.cshtml` 的 `@using` 與 `@addTagHelper` 和 `Views/_ViewImports.cshtml` 相同 —— **Area 的 View 不會繼承 Views/ 底下的 `_ViewImports`**，少了它就是上一步的 CS0246。

`_AreaLayout` 是一個「本身也有 Layout 的 Layout」：

```
@* 範例/02_MVC分層與古典RWD版型/McpPlatform/src/McpPlatform.Web/Views/Shared/_AreaLayout.cshtml *@
@* 會員中心與後臺共用的「側欄+內容」版面，外層仍是全站 _Layout。 *@
@{
    Layout = "_Layout"; // ①
    var area = ViewContext.RouteData.Values["area"] as string;
}
<div class="row g-4">
    <aside class="col-lg-3">
        <nav class="area-nav classic-frame" aria-label="@((area == "Admin" ? "後臺選單" : "會員中心選單"))">
            @if (area == "Admin")
            {
                <partial name="_AdminNav" /> @* ② *@
            }
            else
            {
                <partial name="_MemberNav" />
            }
        </nav>
    </aside>
    <section class="col-lg-9">
        @RenderBody() @* ③ *@
    </section>
</div>

@section Scripts {
    @await RenderSectionAsync("Scripts", required: false) @* ④ *@
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	這個 Layout 自己再套 <code>_Layout</code>	頁首導覽列、頁尾只寫在 <code>_Layout</code> 一個地方，三種頁面自然一致
②	依路由的 <code>area</code> 值放入不同的側欄 Partial	兩個 Area 的版面結構相同、只有選單不同，不必複製兩份 Layout
③	Area 頁面的內容放在右側 <code>col-lg-9</code>	992px 以上側欄在左；以下 <code>col-lg-*</code> 失效，側欄自動疊在內容上方
④	把頁面的 <code>Scripts</code> 區段再轉交給 <code>_Layout</code>	巢狀 Layout 若不轉交，頁面寫的 <code>@section Scripts</code> 會因「沒被渲染」而拋例外

狀態標籤元件把「圖示+文字」的規則集中在一個類別：

```
// 範例/02_MVC分層與古典RWD版型/McpPlatform/src/McpPlatform.Web/TagHelpers/StatusBadgeTagHelper.cs
[HtmlTargetElement("status-badge", TagStructure = TagStructure.WithoutEndTag)]
public sealed class StatusBadgeTagHelper : TagHelper
{
    public ApiVersionStatus Status { get; set; }

    public override void Process(TagHelperContext context, TagHelperOutput output)
    {
        var (icon, text, css) = Status switch // ①
        {
            ApiVersionStatus.Draft => ("📝", "草稿", "status-draft"),
            ApiVersionStatus.Submitted => ("🔍", "審核中", "status-submitted"),
            ApiVersionStatus.Published => ("✓", "已發布", "status-published"),
            ApiVersionStatus.Rejected => ("↔", "已退回", "status-rejected"),
            ApiVersionStatus.Retired => ("🚫", "已下架", "status-retired"),
            _ => ("?", Status.ToString(), "status-draft")
        };

        output.TagName = "span"; // ②
        output.TagMode = TagMode.StartTagAndEndTag;
        output.Attributes.SetAttribute("class", $"status {css}");
        output.Content.AppendHtml($"<span aria-hidden=\"true\">{icon}</span> "); // ③ 圖示不給螢幕報讀器念
        output.Content.Append(text); // ④ 文字會被 HTML 編碼
    }
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	一個 <code>switch</code> 決定五種狀態的圖示、中文、CSS 類別	新增狀態時只改這裡；View 裡不會散落五組 <code>if</code>
②	把 <code><status-badge></code> 換成 <code></code> ，並從自閉合改成有結束標籤	瀏覽器不認得 <code>status-badge</code> ； <code></code> 自閉合寫法在 HTML 裡無效
③	圖示包在 <code>aria-hidden="true"</code> 的 <code>span</code> 裡	螢幕報讀器只念「已發布」，不會念出「勾號已發布」
④	文字用 <code>Append</code> （會 HTML 編碼），不用 <code>AppendHtml</code>	只有自己寫死的圖示標記才用 <code>AppendHtml</code> ；任何可能來自資料的文字都要編碼，避免 XSS

要讓 View 認得 `<status-badge>`，兩個 `_ViewImports.cshtml` 都要有 `@addTagHelper *, McpPlatform.Web`。

最後把 `wwwroot/css/site.css` 換成範例的古典主題。所有顏色集中在最上方：

```
/* 範例/02_MVC分層與古典RWD版型/McpPlatform/src/McpPlatform.Web/wwwroot/css/site.css (節錄) */
:root {
    --paper: #f7f2e7; /* 米白底色 */
    --paper-deep: #efe7d6; /* 區塊底色（比底色深一階） */
    --ink: #2b2620; /* 內文深墨色 */
    --green: #1f4d3a; /* 墨綠：導覽列、主要按鈕、標題 */
    --green-dark: #163829;
    --brown: #5a3e2b; /* 深棕：次要文字、分隔 */
    --gold: #a88450; /* 古銅金：只用於框線與裝飾，不當文字色（對比不足） */
    --focus: #b3541e; /* 鍵盤焦點外框 */
    --serif: "Noto Serif TC", "Source Han Serif TC", "PMingLiU", "MingLiU", serif;
    --sans: "Microsoft JhengHei", "Noto Sans TC", "PingFang TC", sans-serif;
}
```

變數	用在哪裡	改錯的後果
<code>--paper</code> / <code>--ink</code>	<code>body</code> 背景與內文	兩者對比太低時，長時間閱讀容易疲勞
<code>--green</code>	導覽列、標題、主要按鈕	換成淺色時白字導覽列會看不清楚
<code>--gold</code>	<code>.classic-frame</code> 框線、導覽列雙線、表頭底線	拿來當文字色會對比不足（大綱要求保留足夠對比）
<code>--focus</code>	<code>:focus-visible</code> 的 3px 外框	刪掉焦點樣式後，只用鍵盤的人看不出自己在哪個元素上
<code>--serif</code> / <code>--sans</code>	標題／內文	全是系統字型，不下載網路字型，教室離線也正常

後臺清單頁（`Areas/Admin/Views/Apis/Index.cshtml`）同一份 `Model.Items` 輸出兩次：外層 `div.table-responsive.d-none.d-md-block` 裡的表格，與 `ul.admin-cards.d-md-none` 的卡片清單。

預期結果：執行網站後打開七個頁面都回 200：`/`、`/Catalog`、`/Member`、`/Member/Apis/Create`、`/Admin`、`/Admin/Apis`、`/SetupCheck`。畫面與圖 02-4～02-6 一致。

檢查點

做到這裡，你應該可以：

- ☐ `dotnet build McpPlatform.slnx` 顯示 0 個警告、0 個錯誤；`McpPlatform.Core.csproj` 裡沒有任何 `ProjectReference`
- ☐ `/Catalog` 顯示「共 4 筆」，全部是「✓ 已發布」；`/Admin/Apis` 顯示「共 8 筆」
- ☐ 首頁、目錄、會員中心、後臺的導覽列與頁尾完全相同；會員中心與後臺左側有各自的側欄
- ☐ 在「新增 API 草稿」什麼都不填直接送出，每個必填欄位下方出現紅字錯誤，填的內容沒有被清掉

常見錯誤

以下訊息都是撰寫本單元時實際遇到或刻意重現的原文。

症狀 (錯誤訊息原文)	原因	處理
建置時： <code>error MSB3027: 無法將 "...\McpPlatform.Core.dll" 複製到 "bin\Debug\net10.0\McpPlatform.Core.dll"。已超過重試次數 10。失敗。檔案鎖定者: "McpPlatform.Web (30448)"</code>	網站還在執行，DLL 被鎖住，無法覆蓋	在執行網站的終端機按 <code>Ctrl+C</code> 再建置；找不到終端機就在 PowerShell 執行 <code>Get-Process McpPlatform.Web Stop-Process</code>
建置時： <code>error CS0246: 找不到類型或命名空間名稱 'AdminApiListViewModel' (是否遺漏了 using 指示詞或組件參考?)</code> ，檔案位置在 <code>Areas\Admin\Views\...</code>	缺少 <code>Areas/_ViewImports.cshtml</code> ；Area 的 View 不繼承 <code>Views/_ViewImports.cshtml</code> 的 <code>@using</code>	複製 <code>Views/_ViewImports.cshtml</code> 一份到 <code>Areas/</code> ；同時它也帶了 <code>@addTagHelper</code> ，否則 <code><status-badge></code> 會原封不動輸出
<code>/Admin/Apis</code> 回 404；改打 <code>/Apis</code> 回 500： <code>InvalidOperationException: The view 'Index' was not found. The following locations were searched: /Views/Apis/Index.cshtml /Views/Shared/Index.cshtml</code>	<code>Areas/Admin/Controllers/ApisController.cs</code> 少了 <code>[Area("Admin")]</code> ，被當成一般 Controller，也就不會去 <code>Areas/Admin/Views</code> 找 View	在類別上加 <code>[Area("Admin")]</code> ；錯誤訊息列出的搜尋位置裡沒有 <code>/Areas/...</code> ，就是這個原因
在 <code>/Member/Apis/Create</code> 送出表單得到 400，畫面空白	POST 沒帶防偽權杖。通常是自己手寫 <code><form action=" ... "></code> 而不是用 <code>asp-action</code> Tag Helper	改用 <code><form asp-action="Create" method="post"></code> ；它會自動加上 <code>__RequestVerificationToken</code> 隱藏欄位

單元驗收（15 分鐘）

驗收標準：桌面與手機可導覽，表單不溢出，後臺表格具水平捲動或卡片替代。

驗收方式（開啟網站，按 F12 → 裝置工具列，依序設定寬度）：

1. **桌面與手機可導覽**：寬度 1440 時導覽列三個連結直接可點；寬度 390 時點三展開選單，能到達「API 目錄」「會員中心」「管理後臺」；會員中心與後臺的側欄在 390 寬時可以左右滑動，全部選單項目都點得到。
2. **不出現整頁水平捲軸**：在 390、768、1440 三種寬度下，於七個頁面各自在主控台（Console）執行 `document.documentElement.scrollWidth - window.innerWidth`，結果都要是 0。本單元實測 21 個組合全部為 0。
3. **表單不溢出**：寬度 390 打開「新增 API 草稿」，端點 URL 填 `fx.example.test/api/v2`、其他留空後送出。所有欄位上下直排、寬度不超出畫面，錯誤訊息在各欄位下方（圖 02-7 右）。
4. **後臺表格具水平捲動或卡片替代**： `/Admin/Apis` 在 768 寬時表格可左右捲動（實測表格寬 874px、容器 694px），在 390 寬時表格隱藏、改成卡片（圖 02-6）。



圖 02-7 390px 寬的會員中心與「新增 API 草稿」驗證錯誤

畫面重點：

1. ①左圖「我的 API」只有王小明的 3 支：`ListOwnedAsync("member-a")`；草稿、審核中、已下架都看得到，林小華的不會出現。長網址在卡片內換行。
2. ②右圖分類、用途說明、端點 URL 三個欄位紅框：伺服器端 `ModelState` 驗證失敗後帶回原頁；已填的「匯率換算 v2」仍保留。
3. ③端點 URL 的錯誤訊息：端點 URL 格式不正確，需以 `http://` 或 `https://` 開頭。，來自 `[Url]` 屬性。
4. 反向驗證：用任何 HTTP 工具對 `/Member/Apis/Create` 送出沒有 `__RequestVerificationToken` 的 POST（例如 `.http` 檔送 `Name=x&Category=y&Description=z&HttpMethod=GET&EndpointUrl=https://a.test`），必須得到 400，不可以出現「表單驗證通過」的提示。

回顧：規則寫在 Service、資料來源藏在 Repository 介面後面、畫面專用的東西放 ViewModel；版型用一個 `_Layout` 加一個巢狀 `_AreaLayout` 統一三種頁面。下一單元（Section 03）建立 `MCPServerManag` 資料庫與 EF Core Migration，並用 EF Core 版 Repository 取代 `InMemoryApiCatalogRepository`。

延伸閱讀

- [ASP.NET Core 中的 Area](#) —— Area 的資料夾慣例、路由與 `_ViewStart` / `_ViewImports` 的查找規則。
- [ASP.NET Core 中的 Layout](#) —— 巢狀 Layout、`RenderSection` 與 `_ViewStart`。
- [在 ASP.NET Core 撰寫 Tag Helper](#) —— `HtmlTargetElement`、`TagHelperOutput` 的完整用法。
- [防止 ASP.NET Core 中的跨網站請求偽造 \(XSRF/CSRF\) 攻擊](#) —— 防偽權杖的運作方式。

第 3 章

MSSQL 資料模型與 EF Core

Identity 關聯、API 與版本、申請與 Grant 的區別、外鍵、索引、唯一與 CHECK 約束、rowversion、UTC 時間、EF Core Migration、示範資料匯入



課程時數 2 小時（120 分鐘）

當堂成果

MCPServerManag、Migration 與示範資料

範例專案

範例/03_MSSQL 資料模型與 EF Core

學習目標

上完這個單元，你能夠：

- 說出 13 張業務資料表各屬於「API 規格」「審核案件」「使用權」「營運紀錄」哪一群，以及每張表回答什麼問題。
- 解釋為什麼**使用申請**（`ApiAccessRequests`）與**使用權**（`ApiGrants`）要分成兩張表，並用唯一索引保證一件申請最多產生一筆 Grant。
- 用 EF Core 的 Fluent API 設定長度、外鍵、唯一索引、`CHECK` 條件約束與 `rowversion`，並從 SQL Server 的錯誤訊息判斷是哪一條約束擋下了壞資料。
- 用 `dotnet ef migrations add` 與 `dotnet ef database update` 從空白狀態建立 `MCPServerManag`（20 張資料表），並產生可重複執行的 SQL 腳本。
- 匯入示範資料，並以一條 SQL 查出「商品查詢」的版本、上架審核、使用申請、Grant 與呼叫紀錄的關聯。

先備知識

- 完成 Section 02：範例/02_MVC分層與古典RWD版型/McpPlatform 可 `dotnet build`；知道 `IApiCatalogRepository` 介面與記憶體版實作的角色。
- Section 01 已設定 User Secrets 的 `ConnectionStrings:DefaultConnection`，而且 `/SetupCheck` 顯示 SQL Server 「✓ 正常」、`MCPServerManag` 「尚未建立」。本單元的範例是複製 Section 02 而來，`UserSecretsId` 相同，所以不必重設。
- 看得懂 SQL 的 `SELECT ... JOIN ... WHERE ... GROUP BY`；知道主鍵、外鍵的意思。
- 安裝 SSMS，或使用 SQL Server 內附的 `sqlcmd`（本機實測路徑 `C:\Program Files\Microsoft SQL Server\Client SDK\ODBC\180\Tools\Binn\SQLCMD.EXE`）。

觀念說明（20 分鐘）

為什麼資料模型要能追溯

如果把「API 狀態」「審核結果」「誰能用」都塞在同一張表的幾個欄位裡，新值一寫進去舊值就消失了。之後管理者問「這支 API 當初是誰核准的？理由是什麼？為什麼王小明昨天還能叫、今天不行？」——資料庫答不出來。

本專題的大綱要求「審核結果保留版本、操作者、時間與理由」「撤銷或到期後保留原始核准歷程」「API 下架不刪除歷史資料」。所以資料模型的原則是：**會改變的狀態和已經發生的事件分開存**；事件只新增、不覆蓋。

13 張業務資料表分成四群

群組	資料表	回答的問題
API 規格	<code>ApiDefinitions</code> 、 <code>ApiVersions</code> 、 <code>ApiCredentialReferences</code> 、 <code>ToolDefinitions</code>	這支 API 是誰的、有哪些版本、呼叫上游要帶什麼憑證、映射成哪個 MCP Tool
審核案件	<code>ApiPublicationRequests</code> 、 <code>ApiAccessRequests</code> 、 <code>ApprovalRecords</code>	誰在何時送了什麼申請、誰在何時以什麼理由核准或退回
使用權	<code>ApiGrants</code> 、 <code>ApiUsageCounters</code> 、 <code>ExternalIdentityLinks</code> 、 <code>ConnectorClients</code>	現在誰能用哪個版本、今天用了幾次、外部身分對應到哪位會員、哪些 Client 已登錄
營運紀錄	<code>ApiCallLogs</code> 、 <code>AuditLogs</code>	每次呼叫的結果與耗時、每次管理操作的軌跡

另外 `AspNetUsers`、`AspNetRoles` 等 7 張由 ASP.NET Core Identity 產生，合計 20 張。

申請與 Grant 的區別

下圖是「一位會員取得並使用某個 API 版本」在資料表之間留下的足跡。看的時候注意：**每一步都是新增一列**，沒有任何一步是覆蓋前一步的資料。



圖 03-1 從使用申請到呼叫紀錄的資料足跡

編號	資料表	寫入時機	之後會變嗎
①	<code>ApiVersions</code>	提供者建立版本	<code>Status</code> 會隨上架流程改變；URL、Schema 送審後凍結
②	<code>ApiAccessRequests</code>	會員送出使用申請	<code>Status</code> 只從 <code>Pending</code> 變成 <code>Approved</code> 或 <code>Rejected</code> 一次，之後不再改
③	<code>ApprovalRecords</code>	管理者做出決定	永不修改 ：審核人、決定、理由、時間就是歷史
④	<code>ApiGrants</code>	核准時建立	<code>Status</code> 會從 <code>Active</code> 變成 <code>Revoked</code> 或 <code>Expired</code> ；但 <code>SourceAccessRequestId</code> 永遠指回 ②
⑤	<code>ApiCallLogs</code>	每次工具呼叫	只新增

所以「王小明當初有沒有被核准」看 ②③，「王小明現在能不能叫」看 ④。撤銷時只改 ④ 的 `Status`，②③ 的核准歷程仍在——這就是大綱「申請審核狀態與授權有效狀態分開保存」的落實方式。

Identity 關聯與外鍵

會員資料表 `AspNetUsers` 由 ASP.NET Core Identity 提供。本課程用 `ApplicationUser` 繼承內建的 `IdentityUser`，多加 `DisplayName`、`IsActive`（停用）、`CreatedAtUtc` 三個欄位。

`ApplicationUser` 放在 **Infrastructure**，不放 Core：因為 `IdentityUser` 來自 Identity 套件，放進 Core 就破壞了 Section 02「Core 不參考任何東西」的規則。Core 的實體只用字串 `OwnerId`、`UserId`、`ReviewerId` 記住會員 Id，外鍵關係在 Infrastructure 的設定檔裡建立（`HasOne<ApplicationUser>().WithMany().HasForeignKey(x => x.OwnerId)`）。

所有外鍵一律設成 **Restrict**（不串聯刪除）。假如 `ApiDefinitions` → `ApiVersions` 設成 Cascade，刪一支 API 就會連帶刪掉它所有版本；再串下去，申請、Grant 也跟著消失，歷史就沒了。Restrict 讓「刪除仍被引用的資料」直接失敗，逼使用者改用「下架」或「停權」。

約束：讓資料庫擋住壞資料

程式會有 bug，所以最後一道防線放在資料庫：

約束	名稱	擋住什麼
唯一索引	<code>IX_ApiGrants_SourceAccessRequestId</code>	同一件使用申請被核准兩次、產生兩筆 Grant（Section 07 的重複核准）
唯一索引	<code>IX_ApiVersions_ApiDefinitionId_Version</code>	同一支 API 出現兩個 <code>v2</code>
唯一索引	<code>IX_ApiUsageCounters_ApiGrantId_PeriodStart</code>	同一 Grant 同一天有兩列計數，額度被算錯（Section 11）
唯一索引	<code>IX_ExternalIdentityLinks_Issuer_Subject</code>	同一個外部身分對應到兩位會員（Section 10）
唯一索引	<code>IX_ToolDefinitions_ToolName</code>	兩個版本搶同一個 MCP Tool 名稱（Section 09）
CHECK	<code>CK_ApprovalRecords_ExactlyOneRequest</code>	一筆審核紀錄同時屬於上架申請又屬於使用申請，或兩者都不屬於
一般索引	<code>IX_ApiCallLogs_CorrelationId</code> 、 <code>IX_ApiCallLogs_CalledAtUtc</code>	不是約束，是讓「依 <code>CorrelationId</code> 追查」「依日期統計」不必掃整張表（Section 13）

rowversion 與 UTC 時間

rowversion 是 SQL Server 的一種欄位型別：每次該列被更新，資料庫自動給它一個新的、遞增的 8 位元組值。`ApiVersions.RowVersion` 設成 EF Core 的併發權杖後，EF Core 更新時會在 `WHERE` 加上「`RowVersion` 還是我讀到的那個值」。如果兩位管理者同時打開同一件送審案，第一位核准後 `RowVersion` 變了，第二位按核准時 `WHERE` 對不上、更新 0 列，EF Core 丟出 `DbUpdateConcurrencyException` —— 同一案就不會被核准兩次。這在 Section 06 實作。

所有時間欄位一律存 **UTC**（欄位名以 `Utc` 結尾，寫入時用 `DateTimeOffset.UtcNow`）。理由：伺服器可能部署在不同時區，夏令時間的國家還會有一小時重複；統一存 UTC，顯示時才轉成臺灣時間（Section 02 的 `ToTaipeiString()`），儀表板的「今日」也依臺灣時區換算邊界（Section 13）。

Migration 怎麼運作

Migration 是 EF Core 把「C# 模型的變更」轉成「可以依序執行的資料庫變更腳本」的機制。下圖是從寫好實體到資料庫出現資料表的過程。注意：② 只產生檔案、④ 才真的改資料庫，兩者之間可以先 Code Review。

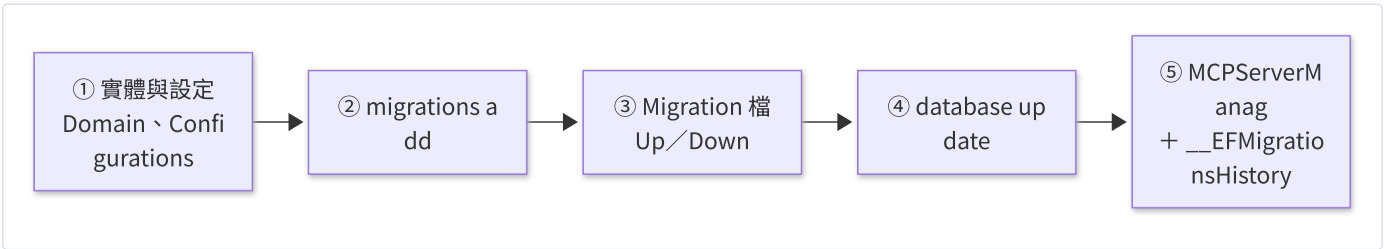


圖 03-2 從 C# 模型到資料庫資料表

編號	做了什麼	對應位置
①	寫實體類別與 <code>IEntityTypeConfiguration<T></code>	<code>Core/Domain/*.cs</code> 、 <code>Infrastructure/Data/Configurations/*.cs</code>
② → ③	比對目前模型與上次快照，把差異寫成 C# 程式： <code>Up</code> 建表、 <code>Down</code> 還原	<code>Infrastructure/Data/Migrations/...InitialCreate.cs</code>
③ → ④	讀 <code>__EFMigrationsHistory</code> 找出還沒套用的 Migration，依時間戳記順序執行 <code>Up</code>	啟動專案的連線字串
⑤	建好資料表，並在 <code>__EFMigrationsHistory</code> 寫一列 Migration 名稱；下次 <code>update</code> 就會跳過它	SQL Server

指令	做什麼	會不會動到資料庫
<code>dotnet ef migrations add InitialCreate</code>	比對目前模型與上一次的快照，產生 <code>___InitialCreate.cs</code> （ <code>Up</code> ／ <code>Down</code> 方法）並更新 <code>AppDbContextModelSnapshot.cs</code>	不會
<code>dotnet ef database update</code>	依序執行還沒套用的 Migration，並記錄在 <code>__EFMigrationsHistory</code> 資料表	會；資料庫不存在就先建立
<code>dotnet ef migrations script --idempotent</code>	輸出 SQL 腳本，每段都先檢查 <code>__EFMigrationsHistory</code> 是否已套用	不會；給 DBA 審閱或部署用（Section 15）

資料表一律由 Migration 建立，網站啟動時不自動建表。本單元的示範資料匯入程式會先檢查「資料庫存在而且沒有待套用的 Migration」，不符合就只記一行警告並跳過。

限制與注意事項

- Migration 檔要**提交到 Git**；它是資料庫結構的版本紀錄。不要手改已經套用過的 Migration，要改就新增一個。
- `dotnet ef` 指令需要兩個專案參數：`--project`（Migration 放在哪，這裡是 Infrastructure）與 `--startup-project`（從哪裡讀連線字串與 DI 設定，這裡是 Web）。
- 示範資料的會員**還沒有密碼**，不能登入；Section 04 加入 Identity 登入後由種子程式設定。
- `ApiCallLogs` 的 `ApiGrantId`、`ApiVersionId` 刻意**不設外鍵**：日誌量大、只新增，不讓外鍵檢查拖慢寫入，也避免日後清理業務資料時被日誌卡住。

名詞對照

名詞	白話說明	在本課程中是
DbContext	EF Core 中代表「一個資料庫連線工作階段」的類別	<code>AppDbContext</code>
Fluent API	用 C# 方法鏈設定資料表細節（長度、索引、約束）	<code>Data/Configurations/*.cs</code>
Migration	可依序執行的資料庫結構變更	<code>Data/Migrations/___InitialCreate.cs</code>
rowversion	SQL Server 每次更新自動變化的版本欄位	<code>ApiVersions.RowVersion</code>
CHECK 條件約束	資料庫層級的欄位值規則	<code>CK_ApprovalRecords_ExactlyOneRequest</code>
冪等	執行一次和執行多次效果相同	<code>DemoDataSeeder</code> 、 <code>--idempotent</code> 腳本

教師示範 (25 分鐘)

示範情境：從「MCPServerManag 不存在」出發，用一條指令建出 20 張資料表，啟動網站匯入示範資料，在後臺看到各表筆數與 API 關聯；最後故意寫入兩筆違規資料，讓學員看到資料庫本身如何擋下它們。

示範步驟

1. 在 範例/03_MSSQL資料模型與EFCore/McpPlatform 執行 `dotnet tool restore` (還原鎖定版本的 `dotnet-ef`)，再執行：

```
dotnet ef database update --project src/McpPlatform.Infrastructure --startup-project src/McpPlatform.Web
```

輸出最後幾行 (本單元實測)：

```
Applying migration '20260926104357_InitialCreate'.  
Done.
```

2. 啟動網站 `dotnet run --project src/McpPlatform.Web --launch-profile http`，終端機出現 已匯入示範資料。再啟動一次時不會再出現這行，因為匯入程式是幕等的。
3. 開 `http://localhost:5199/SetupCheck`，系統資料庫那一列從「尚未建立」變成「已連線，系統資料庫已存在。」。
4. 點「管理後臺」→ 側欄「資料庫概況」。

MCP Server API 平台

API 目錄

會員中心

管理後臺

示範身分：王小明 (Section 04 起改為登入)

管理後臺

總覽

API 管理

資料庫概況

上架審核 (Section 06)

會員管理 (Section 12)

資料庫概況

資料庫 MCPServerManag · 已套用 1 個 Migration：20260926104357_InitialCreate

各資料表筆數

資料表	筆數
AspNetUsers	4
ApiDefinitions	6
ApiVersions	7
ApiPublicationRequests	6
ApiAccessRequests	3
ApprovalRecords	7
ApiGrants	1
ApiUsageCounters	1
ApiCallLogs	5
AuditLogs	1
ToolDefinitions	0
ConnectorClients	0
ExternalIdentityLinks	0
ApiCredentialReferences	0

API 關聯查詢

商品查詢

查詢

商品查詢 (ApiDefinitions.Id = 1 · 提供者：教師示範帳號)

版本	狀態	上架申請	使用申請	有效 Grant	呼叫紀錄
v1	已發布	1	1	1	5

© 2026 MCP Server API 平台 | 環境檢查

圖 03-3 後臺資料庫概況：Migration 紀錄、各表筆數與 API 關聯

畫面重點：

1. ①標題下方的 Migration 清單：20260926104357_InitialCreate，由 `db.Database.GetAppliedMigrationsAsync()` 讀 `__EFMigrationsHistory` 而來。你的時間戳記會不同。
2. ②左側「各資料表筆數」：AspNetUsers 4、ApiDefinitions 6、ApiVersions 7 (天氣預報有 v1、v2 兩個版本)、ApprovalRecords 7、ApiGrants 1、ApiCallLogs 5；ToolDefinitions 等 4 張目前是 0，之後的單元才會寫入。
3. ③右側「API 關聯查詢」：選「商品查詢」，v1 有 1 件上架申請、1 件使用申請、1 筆有效 Grant、5 筆呼叫紀錄。改選「天氣預報查詢」會看到 v1「已發布」與 v2「審核中」兩列。
4. 在 SSMS 對 MCPServerManag 執行一段違反 CHECK 約束的新增 (同一筆審核紀錄同時指向上架申請 1 與使用申請 1)：

```
-- 在 MCPServerManag 執行；會失敗，不會寫入任何資料
INSERT INTO dbo.ApprovalRecords (PublicationRequestId, AccessRequestId, ReviewerId, Decision, Reason, DecidedAtUtc)
VALUES (1, 1, 'admin', 'Approved', 'x', SYSUTCDATETIME());
```

執行結果（本單元實測原文）：

訊息 547，層級 16，狀態 1，伺服器 ERICROG，行 1
 INSERT 陳述式與 CHECK 條件約束 "CK_ApprovalRecords_ExactlyOneRequest" 衝突。衝突發生在資料庫 "MCPServerManag"，資料表 "dbo.ApprovalRecords"。
 陳述式已經結束。

再對同一件使用申請新增第二筆 Grant（SourceAccessRequestId = 1），得到 訊息 2601 ... 無法以唯一索引 'IX_ApiGrants_SourceAccessRequestId' 在物件 'dbo.ApiGrants' 中插入重複的索引鍵資料列。重複的索引鍵值是 (1)。。兩次失敗後，ApprovalRecords 仍是 7 筆、ApiGrants 仍是 1 筆。

程式碼走讀：AppDbContext

AppDbContext 是整個資料層的入口，繼承 IdentityDbContext<ApplicationUser>，所以同一個 Migration 會連 Identity 的 7 張表一起建。

```
// 範例/03_MSSQL資料模型與EFCore/McpPlatform/src/McpPlatform.Infrastructure/Data/AppDbContext.cs (DbSet 宣告以外的部分)
public class AppDbContext(DbContextOptions<AppDbContext> options) : IdentityDbContext<ApplicationUser>(options)
{
    protected override void ConfigureConventions(ModelConfigurationBuilder configurationBuilder)
    {
        // 所有列舉一律存成字串（例如 'Published'），直接查資料表就看得懂，不必對照數字。
        configurationBuilder.Properties<Enum>().HaveConversion<string>().HasMaxLength(20); // ①
    }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder); // ② 先建 Identity 的資料表
        builder.ApplyConfigurationsFromAssembly(typeof(AppDbContext).Assembly); // ③ 再套用 Configurations/ 底下的設定

        // 所有外鍵都不串聯刪除：下架、停權都不應該連帶刪掉歷史紀錄。
        foreach (var fk in builder.Model.GetEntityTypes().SelectMany(e => e.GetForeignKeys()))
            fk.DeleteBehavior = DeleteBehavior.Restrict; // ④
    }
}
```

（省略：13 個 public DbSet<...> ... => Set<...>(); 宣告，完整內容見範例檔。）

程式說明：

標號	做了什麼	為什麼這樣寫
①	全部列舉型別存成 <code>nvarchar(20)</code> 的字串	查 <code>SELECT Status FROM ApiVersions</code> 看到 <code>Published</code> 而不是 <code>2</code> ；列舉中間插入新值時也不會讓舊資料意思改變
②	先呼叫 <code>base.OnModelCreating</code>	Identity 的 <code>AspNetUsers</code> 等設定在基底類別裡；漏掉這行，Migration 會缺 Identity 的主鍵設定而失敗
③	自動套用同組件內所有 <code>IEntityTypeConfiguration<T></code>	每張表的設定放在自己的類別， <code>AppDbContext</code> 不會長成幾百行
④	把所有外鍵（含 Identity 的）改成 <code>Restrict</code>	SQL Server 不允許多條串聯刪除路徑；更重要的是不讓刪除動作連帶抹掉歷史

學員實作（60 分鐘）

範例專案：範例/03_MSSQL資料模型與EFCore/McpPlatform/（從範例/02_MVC分層與古典RWD版型/McpPlatform/複製後往上加）。本單元新增 Core 的 Domain/（實體與列舉）、Diagnostics/（資料庫概況介面）；Infrastructure 的 Data/（AppDbContext、ApplicationUser、Configurations/、Migrations/、DemoDataSeeder、DatabaseOverviewReader）與 Catalog/EfApiCatalogRepository.cs；Web 的後臺「資料庫概況」頁；方案根目錄的 dotnet-tools.json 與 sql/。Section 02 的 InMemoryApiCatalogRepository 保留在專案裡，但不再註冊。

步驟一 加入 EF Core 套件與工具（10 分鐘）

目的：在正確的專案加入正確的套件，並把 dotnet-ef 工具的版本也鎖住。

```
# 在 McpPlatform/ 根目錄執行
dotnet new tool-manifest
dotnet tool install dotnet-ef --version 10.0.12
dotnet add src/McpPlatform.Infrastructure package Microsoft.EntityFrameworkCore.SqlServer --version 10.0.12
dotnet add src/McpPlatform.Infrastructure package Microsoft.AspNetCore.Identity.EntityFrameworkCore --version 10.0.12
dotnet add src/McpPlatform.Web package Microsoft.EntityFrameworkCore.Design --version 10.0.12
```

預期結果：已成功安裝工具 'dotnet-ef'（版本 '10.0.12'）。；Directory.Packages.props 多三行 PackageVersion，版本都是 10.0.12（與執行階段 .NET 10.0.12 一致）。

套件	加在哪	為什麼放這裡
Microsoft.EntityFrameworkCore.SqlServer	Infrastructure	資料存取實作只屬於 Infrastructure
Microsoft.AspNetCore.Identity.EntityFrameworkCore	Infrastructure	提供 IdentityDbContext、IdentityUser
Microsoft.EntityFrameworkCore.Design	Web	dotnet ef 需要啟動專案有這個套件才能在設計階段建立 DbContext
dotnet-ef（工具）	dotnet-tools.json	別台電腦執行 dotnet tool restore 就裝到同一版，不受全域安裝的版本影響

步驟二 定義實體與資料表設定（20 分鐘）

目的：在 Core 寫實體類別，在 Infrastructure 用 Fluent API 描述資料表細節。

實體直接從範例複製：Core/Domain/Enums.cs（5 個列舉）、ApiEntities.cs（ApiDefinition、ApiVersion、ApiCredentialReference、ToolDefinition）、ReviewEntities.cs（兩種申請、ApprovalRecord、ApiGrant）、OperationEntities.cs（其餘 5 個）。實體是只有屬性的普通類別，例如 ApiVersion 有 Version、HttpMethod、EndpointUrl、InputSchema、Status、RowVersion、三個 utc 時間。

設定檔以 ApiVersions 最具代表性：

```
// 範例/03_MSSQL資料模型與EFCore/McpPlatform/src/McpPlatform.Infrastructure/Data/Configurations/ApiConfigurations.cs (ApiVersion 部分)
public sealed class ApiVersionConfiguration : IEntityTypeConfiguration<ApiVersion>
{
    public void Configure(EntityTypeBuilder<ApiVersion> e)
    {
        e.Property(x => x.Version).HasMaxLength(20);
        e.Property(x => x.HttpMethod).HasMaxLength(10);
        e.Property(x => x.EndpointUrl).HasMaxLength(500);
        e.Property(x => x.RowVersion).IsRowVersion(); // ① rowversion 欄位
        e.HasOne(x => x.ApiDefinition).WithMany(d => d.Versions).HasForeignKey(x => x.ApiDefinitionId);
        e.HasIndex(x => new { x.ApiDefinitionId, x.Version }).IsUnique(); // ② 同一支 API 的版本號不重複
        e.HasIndex(x => x.Status); // ③ 公開目錄依狀態查詢
    }
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	對應成 SQL Server 的 rowversion 型別，並自動當作併發權杖	更新時 EF Core 在 WHERE 比對它，兩人同時修改時後到者會失敗（Section 06）
②	(ApiDefinitionId, Version) 唯一	版本號是給人看的識別；重複的 v2 會讓申請與授權指錯版本
③	Status 一般索引	公開目錄每次都查 Status = 'Published'，有索引就不必掃整張表

沒有寫 HasMaxLength 的字串（例如 InputSchema）會成為 nvarchar(max)，適合存 JSON Schema 這種長度不定的文字。

審核紀錄與 Grant 的設定示範 CHECK 約束與「一件申請一筆 Grant」：

```
// 範例/03_MSSQL資料模型與EFCore/McpPlatform/src/McpPlatform.Infrastructure/Data/Configurations/ReviewConfigurations.cs (兩個類別)
public sealed class ApprovalRecordConfiguration : IEntityTypeConfiguration<ApprovalRecord>
{
    public void Configure(EntityTypeBuilder<ApprovalRecord> e)
    {
        e.Property(x => x.Reason).HasMaxLength(500);
        e.HasOne<ApplicationUser>().WithMany().HasForeignKey(x => x.ReviewerId);
        e.ToTable(t => t.HasCheckConstraint( // ② 兩個外鍵恰好一個有值
            "CK_ApprovalRecords_ExactlyOneRequest",
            "([PublicationRequestId] IS NOT NULL AND [AccessRequestId] IS NULL) OR ([PublicationRequestId] IS NULL AND [AccessRequestId] IS NOT NULL)"));
    }
}

public sealed class ApiGrantConfiguration : IEntityTypeConfiguration<ApiGrant>
{
    public void Configure(EntityTypeBuilder<ApiGrant> e)
    {
        e.Property(x => x.Scope).HasMaxLength(50);
        e.Property(x => x.RevokedReason).HasMaxLength(200);
        e.HasOne<ApplicationUser>().WithMany().HasForeignKey(x => x.UserId);
        e.HasIndex(x => x.SourceAccessRequestId).IsUnique(); // ② 一件申請最多產生一筆 Grant
        e.HasIndex(x => new { x.UserId, x.ApiVersionId, x.Status }); // ③ 呼叫時查「這人對這版本有沒有有效 Grant」
    }
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	建立 CHECK 條件約束： <code>PublicationRequestId</code> 與 <code>AccessRequestId</code> 恰好一個不是 <code>NULL</code>	大綱要求「以約束確保只對應一類案件」；只靠程式判斷，哪天有人直接寫 SQL 就破功
②	<code>SourceAccessRequestId</code> 唯一	就算 Section 07 的程式因為併發被呼叫兩次，第二筆 Grant 也寫不進去（教師示範的 2601 錯誤）
③	<code>(UserId, ApiVersionId, Status)</code> 複合索引	每次 MCP 呼叫都要查這三個欄位（Section 11），這是最常用的查詢
—	<code>HasOne<ApplicationUser>().WithMany()</code>	沒有導覽屬性的外鍵：Core 的實體不認識 <code>ApplicationUser</code> ，關聯只存在 Infrastructure 的設定裡

預期結果：`dotnet build McpPlatform.slnx` 顯示 0 個錯誤。

步驟三 註冊 DbContext、建立 Migration 與資料庫（15 分鐘）

目的：讓網站知道怎麼連資料庫，產生第一個 Migration，並真的建出 `MCPServerManag`。

```
// 範例/03_MSSQL資料模型與EFCore/McpPlatform/src/McpPlatform.Web/Program.cs (資料庫相關的註冊)
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection")
?? throw new InvalidOperationException("缺少 ConnectionStrings:DefaultConnection (請用 dotnet user-secrets set 設定)。");
builder.Services.AddDbContext<AppDbContext>(options => options.UseSqlServer(connectionString)); // ①

// 分層註冊：Controller 只認得 Core 的介面，實作由這裡決定。
builder.Services.AddScoped<IApiCatalogService, ApiCatalogService>(); // 業務層
builder.Services.AddScoped<IApiCatalogRepository, EfApiCatalogRepository>(); // ② 資料層：Section 02 的記憶體版換成 EF Core 版
builder.Services.AddScoped<IDatabaseOverviewReader, DatabaseOverviewReader>();
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	以 Scoped 註冊 <code>AppDbContext</code> ，連線字串取自 <code>DefaultConnection</code> ；讀不到就在啟動時丟例外	<code>DbContext</code> 不是執行緒安全的，一個 HTTP 請求一個實例；連線字串缺漏要一開機就發現
②	只改這一行，就把資料來源從記憶體換成 SQL Server	Section 02 圖 02-1 的承諾：Controller 與 Service 一行都不用改；也從 <code>AddSingleton</code> 改成 <code>AddScoped</code> ，因為它依賴 Scoped 的 <code>AppDbContext</code>

`DefaultConnection` 的連線字串裡 `Database=MCPServerManag` 在 Section 01 就寫好了。接著產生 Migration 並套用：


```
# 在 McpPlatform/ 根目錄執行
dotnet ef migrations add InitialCreate --project src/McpPlatform.Infrastructure --startup-project src/McpPlatform.Web --output-dir Data/Migrations
dotnet ef database update --project src/McpPlatform.Infrastructure --startup-project src/McpPlatform.Web
dotnet ef migrations script --idempotent --project src/McpPlatform.Infrastructure --startup-project src/McpPlatform.Web -o sql/01_InitialCreate_idempotent.sql
```

預期結果：

```
Done. To undo this action, use 'ef migrations remove'
Applying migration '20260926104357_InitialCreate'.
Done.
```

Data/Migrations/ 多三個檔案：`___InitialCreate.cs`（20 個 `CreateTable`）、`___InitialCreate.Designer.cs`、`AppDbContextModelSnapshot.cs`；`sql/01_InitialCreate_idempotent.sql` 約 630 行。在 SSMS 重新整理，`MCPServerManag` 底下有 20 張資料表加上 `___EFMigrationsHistory`。

常見改錯：少了 `--startup-project`，得到 `Your startup project 'McpPlatform.Infrastructure' doesn't reference Microsoft.EntityFrameworkCore.Design.`（見〈常見錯誤〉）。

步驟四 換上 EF Core 資料層並匯入示範資料（15 分鐘）

目的：讓前臺目錄、會員中心、後臺改讀 SQL Server，並在開發環境自動匯入一組示範資料。

EF Core 版的 Repository 實作 Section 02 的同一個介面：

```
// 範例/03_MSSQL資料模型與EFCore/McpPlatform/src/McpPlatform.Infrastructure/Catalog/EfApiCatalogRepository.cs (SearchAsync)
public async Task<IReadOnlyList<ApiSummaryDto>> SearchAsync(ApiSearchCriteria criteria, CancellationToken ct)
{
    var query =
        from v in db.ApiVersions.AsNoTracking() // ① 唯讀查詢不追蹤
        join u in db.Users on v.ApiDefinition.OwnerId equals u.Id // ② 取擁有者顯示名稱
        select new { Version = v, Api = v.ApiDefinition, OwnerName = u.DisplayName };

    if (criteria.Status is { } status) // ③ 條件逐一疊加，最後才送出一條 SQL
        query = query.Where(x => x.Version.Status == status);
    if (criteria.OwnerId is { } ownerId)
        query = query.Where(x => x.Api.OwnerId == ownerId);
    if (criteria.Category is { } category)
        query = query.Where(x => x.Api.Category == category);
    if (criteria.Keyword is { } keyword)
        query = query.Where(x => x.Api.Name.Contains(keyword) || x.Api.Description.Contains(keyword));

    return await query
        .OrderByDescending(x => x.Version.UpdatedAtUtc)
        .Select(x => new ApiSummaryDto( // ④ 只選需要的欄位，直接投影成 DTO
            x.Version.Id, x.Api.Name, x.Api.Category, x.Api.Description,
            x.Api.OwnerId, x.OwnerName, x.Version.Version, x.Version.HttpMethod,
            x.Version.EndpointUrl, x.Version.Status, x.Version.UpdatedAtUtc))
        .ToListAsync(ct); // ⑤ 非同步執行
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	<code>AsNoTracking()</code>	只讀不改的查詢不需要變更追蹤，省記憶體也比較快
②	用 LINQ <code>join</code> 接 <code>db.Users</code> 取 <code>DisplayName</code>	實體沒有指向 <code>ApplicationUser</code> 的導覽屬性，所以用明確的 <code>join</code>
③	有值的條件才加 <code>Where</code> ；此時還沒查資料庫	<code>IQueryable</code> 延遲執行：所有條件組好後才翻成一條 SQL，而不是先撈全部再在記憶體過濾（Section 02 記憶體版的做法）
④	<code>Select</code> 直接投影成 DTO	SQL 只選需要的 11 個欄位； <code>ApiCredentialReferences</code> 完全沒被碰到，公開目錄不可能洩漏憑證
⑤	<code>ToListAsync(ct)</code>	查詢期間不佔住執行緒；使用者關閉頁面時 <code>ct</code> 取消，SQL 也跟著取消

示範資料由 `Data/DemoDataSeeder.cs` 匯入，在一個交易內寫入 4 位會員（`teacher`、`member-a` 王小明、`member-b` 林小華、`admin`）、6 支 API 的 7 個版本、6 件上架申請、3 件使用申請、7 筆審核紀錄、1 筆 Grant、5 筆呼叫紀錄（4 成功 1 逾時）；資料庫已有任何 API 就直接跳過。它由 `Program.cs` 在開發環境呼叫，並先確認資料表已由 Migration 建好：

```
// 範例/03_MSSQL資料模型與EFCore/McpPlatform/src/McpPlatform.Web/Program.cs (示範資料匯入)
if (app.Environment.IsDevelopment() && app.Configuration.GetValue<bool>("Seed:DemoData"))
{
    await using var scope = app.Services.CreateAsyncScope();
    var db = scope.ServiceProvider.GetRequiredService<AppDbContext>();
    var logger = scope.ServiceProvider.GetRequiredService<ILogger<Program>>();
    if (!await db.Database.CanConnectAsync() || (await db.Database.GetPendingMigrationsAsync()).Any()) // ③
        logger.LogWarning("資料庫不存在或尚未套用最新 Migration，略過示範資料匯入。請先執行 dotnet ef database update。");
    else if (await DemoDataSeeder.SeedAsync(db)) // ④
        logger.LogInformation("已匯入示範資料。");
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
③	連不上資料庫，或還有沒套用的 Migration，就只記警告	在缺表的資料庫上匯入會丟出例外讓網站起不來；而且我們 不讓網站自動建表 ，建表是有意識的 database update
④	SeedAsync 回傳 true 表示這次真的匯入	第二次啟動回傳 false 、不印訊息，學員能分辨匯入有沒有重複
—	外層 IsDevelopment() 且 Seed:DemoData 為 true	開關寫在 appsettings.Development.json ；正式環境永遠不會匯入假資料

另外在 **appsettings.Development.json** 的 **Logging:LogLevel** 加 **"Microsoft.EntityFrameworkCore.Database.Command": "Warning"**，否則每條 SQL 都會印到終端機。

預期結果：啟動網站，終端機出現 **已匯入示範資料。**；**/Catalog** 共 4 筆（天氣預報只有 v1 出現，v2 還在審核中）；**/Member** 顯示「你共有 3 支 API」；**/Admin/Apis** 共 7 筆；**/Admin/Database** 與圖 03-3 一致。

檢查點

做到這裡，你應該可以：

- ☐ SSMS 裡 **MCPServerManag** 有 20 張資料表，**__EFMigrationsHistory** 有 1 列 **___InitialCreate**
- ☐ 重啟網站第二次，終端機**沒有**再出現「已匯入示範資料。」，後臺各表筆數不變
- ☐ **/Admin/Database** 選「天氣預報查詢」，看到 v1「 已發布」與 v2「 審核中」兩列
- ☐ **sql/01_InitialCreate_idempotent.sql** 存在，而且 Migration 資料夾已加入 Git

常見錯誤

以下訊息都是撰寫本單元時實際執行得到的原文。

症狀（錯誤訊息原文）	原因	處理
<code>Your startup project 'McpPlatform.Infrastructure' doesn't reference Microsoft.EntityFrameworkCore.Design. This package is required for the Entity Framework Core Tools to work.</code>	只給了 <code>--project</code> ， <code>dotnet ef</code> 把 Infrastructure 當成啟動專案	每個 <code>dotnet ef</code> 指令都加 <code>--startup-project src/McpPlatform.Web</code>
<code>Unrecognized option '--connection'（執行 dotnet ef database drop --connection ...）</code>	<code>database update</code> 支援 <code>--connection</code> ，但 <code>database drop</code> 不支援	刪除臨時資料庫改用 SQL： <code>ALTER DATABASE [名稱] SET SINGLE_USER WITH ROLLBACK IMMEDIATE; DROP DATABASE [名稱];</code>
訊息 547 ... INSERT 陳述式與 CHECK 條件約束 "CK_ApprovalRecords_ExactlyOneRequest" 衝突。	審核紀錄同時填了（或都沒填） <code>PublicationRequestId</code> 與 <code>AccessRequestId</code>	這是約束在正常運作；檢查寫入程式，兩個欄位只能填一個
訊息 2601 ... 無法以唯一索引 'IX_ApiGrants_SourceAccessRequestId' 在物件 'dbo.ApiGrants' 中插入重複的索引鍵資料列。重複的索引鍵值是 (1)。	同一件使用申請要建第二筆 Grant	約束在正常運作；Section 07 會在 Service 先檢查並把這個錯誤轉成友善訊息
用 <code>sqlcmd</code> 查詢，中文變成 <code>â~d</code>	Git Bash 以 UTF-8 顯示， <code>sqlcmd</code> 輸出是 Big5（cp950）	在 PowerShell 或 SSMS 執行；或 Git Bash 加 <code>\ iconv -f cp950 -t utf-8</code>

單元驗收（15 分鐘）

驗收標準：可由 Migration 建立空白資料庫；查詢某 API 的版本、案件及呼叫關聯。

驗收方式：

1. 由 Migration 建立空白資料庫：不動 `MCPServerManag`，另建一個驗收用資料庫（`--connection` 只影響這一次指令）：

```
# 在 McpPlatform/ 根目錄執行；密碼請填入自己的 sa 密碼
dotnet ef database update --project src/McpPlatform.Infrastructure --startup-project src/McpPlatform.Web --connection "Server=localhost;Database=MCPServerManag_Verify;User Id=sa;Password=<sa 密碼>;TrustServerCertificate=True"
```

預期輸出 `Applying migration '20260926104357_InitialCreate'.` 與 `Done.`。在 `MCPServerManag_Verify` 查 `SELECT COUNT(*) FROM sys.tables WHERE name <> '__EFMigrationsHistory'` 得到 `20`，`SELECT COUNT(*) FROM dbo.ApiDefinitions` 得到 `0`（Migration 只建結構、不放資料）。驗完用 SQL 刪除它：`ALTER DATABASE [MCPServerManag_Verify] SET SINGLE_USER WITH ROLLBACK IMMEDIATE; DROP DATABASE [MCPServerManag_Verify];`。

2. 查詢某 API 的版本、案件及呼叫關聯：在 `MCPServerManag` 執行 `sql/02_api_relations.sql`（三段 `SELECT`，只讀、可重複執行）。本單元實測結果：

```
ApiId|Name|VersionId|Version|HttpMethod|Status
1|商品查詢|1|v1|GET|Published

Version|PublicationRequestId|RequestStatus|Decision|Reason|Reviewer
v1|1|Approved|Approved|規格完整，端點為課堂允許清單內的主機。|系統管理員

Applicant|AccessStatus|GrantId|GrantStatus|Quota|Calls|SuccessCalls
王小明|Approved|1|Active|100|5|4
```

第三段的關聯路徑正是圖 03-1：`ApiVersions` → `ApiAccessRequests` → `ApiGrants`（以 `SourceAccessRequestId` 連回申請）→ `ApiCallLogs`。

3. 反向驗證：執行教師示範第 5 步的違規 `INSERT`，必須得到訊息 547，且 `ApprovalRecords` 筆數不變。資料庫若接受了這筆資料，代表 CHECK 約束沒有建立——檢查 Migration 的 `...InitialCreate.cs` 裡有沒有 `CK_ApprovalRecords_ExactlyOneRequest`。

回顧：狀態與事件分開存，審核與授權都留下不可覆蓋的紀錄；唯一索引與 CHECK 約束是最後一道防線；資料表只由 Migration 建立。下一單元（Section 04）啟用 Identity 的註冊、登入與角色，讓示範會員真的能登入，並保護 `Member` 與 `Admin` 兩個 Area。

延伸閱讀

- [EF Core Migration 概觀](#)——`migrations add`、`database update` 與 `__EFMigrationsHistory` 的運作。
- [EF Core 處理併發衝突](#)——`rowversion` 併發權杖與 `DbUpdateConcurrencyException`，Section 06 會用到。
- [EF Core 索引](#)——唯一索引、複合索引的設定方式。
- [ASP.NET Core Identity 模型自訂](#)——繼承 `IdentityUser` 加欄位的做法。

第 4 章

會員註冊登入與角色授權

Identity 密碼雜湊、Cookie、角色、Policy、Area 授權慣例、Anti-forgery、登入錯誤與帳號鎖定、Admin 種子、本人資料範圍



課程時數 2 小時（120 分鐘） 當堂成果 可登入、登出且權限受控的網站 範例專案 範例/04_會員註冊登入與角色授權

學習目標

上完這個單元，你能夠：

- 用 ASP.NET Core Identity 完成註冊、登入、登出與「我的資料」，並說出密碼在資料庫裡存的是什麼（`AQAAAAIAAYag...` 開頭的雜湊值，不是明碼）。
- 區分**認證**（你是誰）與**授權**（你能做什麼），並指出 `UseAuthentication`、`UseAuthorization`、授權政策、Service 查詢條件各擋下哪一種越權。
- 用一個 MVC 慣例讓 `Admin` Area 的**所有** Controller 自動要求 `Admin` 角色，讓 `Member` Area 自動要求登入；之後新增的 Controller 忘了加 `[Authorize]` 也不會變成公開頁面。
- 讓 `/api/` 開頭的管理 API 在未登入時回 `401`、權限不足時回 `403`，而不是轉址到登入頁。
- 實作連錯 5 次鎖定 15 分鐘、所有 POST 驗證防偽權杖、註冊表單不能指定角色，並用 404 保護「別人的 API」。

先備知識

- 完成 Section 03：MCPServerManag 已由 Migration 建立並匯入示範資料，AspNetUsers 有 teacher、member-a（王小明，a@demo.test）、member-b（林小華，b@demo.test）、admin（admin@demo.test）4 位會員，但都還沒有密碼。
- Section 02 的 ViewModel 防過度提交、[ValidateAntiForgeryToken]；本單元把它擴大到全站。
- 知道 HTTP 狀態碼 302（轉址）、401（未認證）、403（已認證但沒權限）、404（找不到）。

觀念說明（20 分鐘）

為什麼不自己寫會員系統

自己寫登入最常出現三個漏洞：密碼用 MD5 或明碼存（資料庫外洩就全部淪陷）、沒有登入失敗上限（可以無限次猜密碼）、登入狀態存在可被 JavaScript 讀取的地方（XSS 一次就被偷走）。ASP.NET Core Identity 是官方的會員框架，已經處理好：

項目	Identity 的做法	本課程的設定
密碼儲存	PBKDF2 加鹽雜湊，存在 <code>AspNetUsers.PasswordHash</code>	密碼至少 8 字元，含大寫、小寫、數字、符號
暴力猜測	失敗次數存 <code>AccessFailedCount</code> ，到上限寫入 <code>LockoutEnd</code>	連錯 5 次鎖 15 分鐘
登入狀態	加密簽章的 Cookie，只有伺服器能解開	<code>HttpOnly</code> ，8 小時滑動到期
角色	<code>AspNetRoles</code> 、 <code>AspNetUserRoles</code>	只有一個 <code>Admin</code> 角色

認證與授權：一個請求要過幾道關

認證（Authentication）回答「你是誰」；授權（Authorization）回答「你能做什麼」。下圖是一個請求打到 `/Member/Api/Details/7` 時經過的關卡。注意：④ 與 ⑤ 是兩種不同的檢查，缺一不可。



圖 04-1 一個會員中心請求經過的認證與授權關卡

編號	做了什麼	不通過時的結果	對應程式
① → ②	解密 Cookie，還原成 <code>User</code> （含 Id、Email、角色、 <code>display_name</code> ）	沒有 Cookie 或解不開 → 當成訪客	<code>app.UseAuthentication()</code>
③	依政策判斷： <code>Member</code> Area 要登入、 <code>Admin</code> Area 要 <code>Admin</code> 角色	訪客 → 302 到 <code>/Account/Login (/api/ 回 401)</code> ；已登入但沒角色 → 302 到 <code>/Account/AccessDenied (/api/ 回 403)</code>	<code>AreaAuthorizationConvention</code> 、 <code>AddAuthorizationBuilder</code>
④	從 <code>User</code> 取 <code>NameIdentifier</code> ，不從網址或表單拿會員 Id	—	<code>User.GetUserId()</code>
⑤	查詢條件同時帶「版本 Id」與「擁有者 = 登入者」	別人的或不存在 → 回 <code>null</code> → Controller 回 404	<code>ApiCatalogService.GetOwnedAsync</code>

③ 只知道「這個人有沒有登入、是不是管理者」，不知道「這筆資料是誰的」；所以王小明通過 ③ 之後，仍要靠 ⑤ 擋住他看林小華的 API。大綱說的「對會員資源額外檢查 OwnerId」指的就是 ⑤。

角色、政策與 Area 慣例

角色（Role）是貼在會員身上的標籤，本課程只有 `Admin`。政策（Policy）是一組授權條件，有名字：

政策	條件	用在哪
<code>AdminOnly</code>	已登入，且角色包含 <code>Admin</code>	<code>Admin</code> Area 全部、 <code>/api/admin/apis</code>
<code>Member</code>	已登入（任何角色）	<code>Member</code> Area 全部

最常見的漏洞是忘了在新的 Controller 加 `[Authorize]`。本課程改用 MVC 的慣例（Convention）：網站啟動時掃描所有 Controller，只要 `[Area("Admin")]` 就自動加上 `AdminOnly` 政策。之後 Section 06、12、13 新增的後臺 Controller 一律受保護，不靠記憶。

大綱要求「以 Admin Area、Controller/Action、Web API 及 Service 層共同落實權限」，本單元的分工是：Area 用慣例、Web API 用 `[Authorize(Policy = ...)]`、會員資源用 Service 的查詢條件；導覽列隱藏「管理後臺」連結只是介面整潔，不算保護。

Cookie 與防偽權杖

登入成功後，伺服器發一個名為 `McpPlatform.Auth` 的 Cookie。它設了 `HttpOnly`，JavaScript 讀不到；ASP.NET Core 預設也會加上 `SameSite=Lax`，別的網站嵌入的表單 POST 過來時，瀏覽器不會帶這個 Cookie。

但只靠 `SameSite` 不夠（舊瀏覽器、同站子網域等例外），所以再加一層**防偽權杖（Anti-forgery token）**：每個 `<form asp-...>` 會自動帶一個隱藏欄位

`__RequestVerificationToken`，伺服器比對它和另一個 Cookie 是否成對。本單元用 `AutoValidateAntiforgeryTokenAttribute` 讓**全站所有 POST** 都檢查，不必每個 Action 加 `[ValidateAntiForgeryToken]`。登出也是 POST —— 如果登出是 GET，別的網站放一張 `` 就能把你登出。

登入錯誤與帳號鎖定

下圖是 `AccountController.Login` 收到 Email 與密碼後的判斷順序。注意：「帳號不存在」與「密碼錯誤」回同一句話，「已停權」只有在密碼正確之後才說。

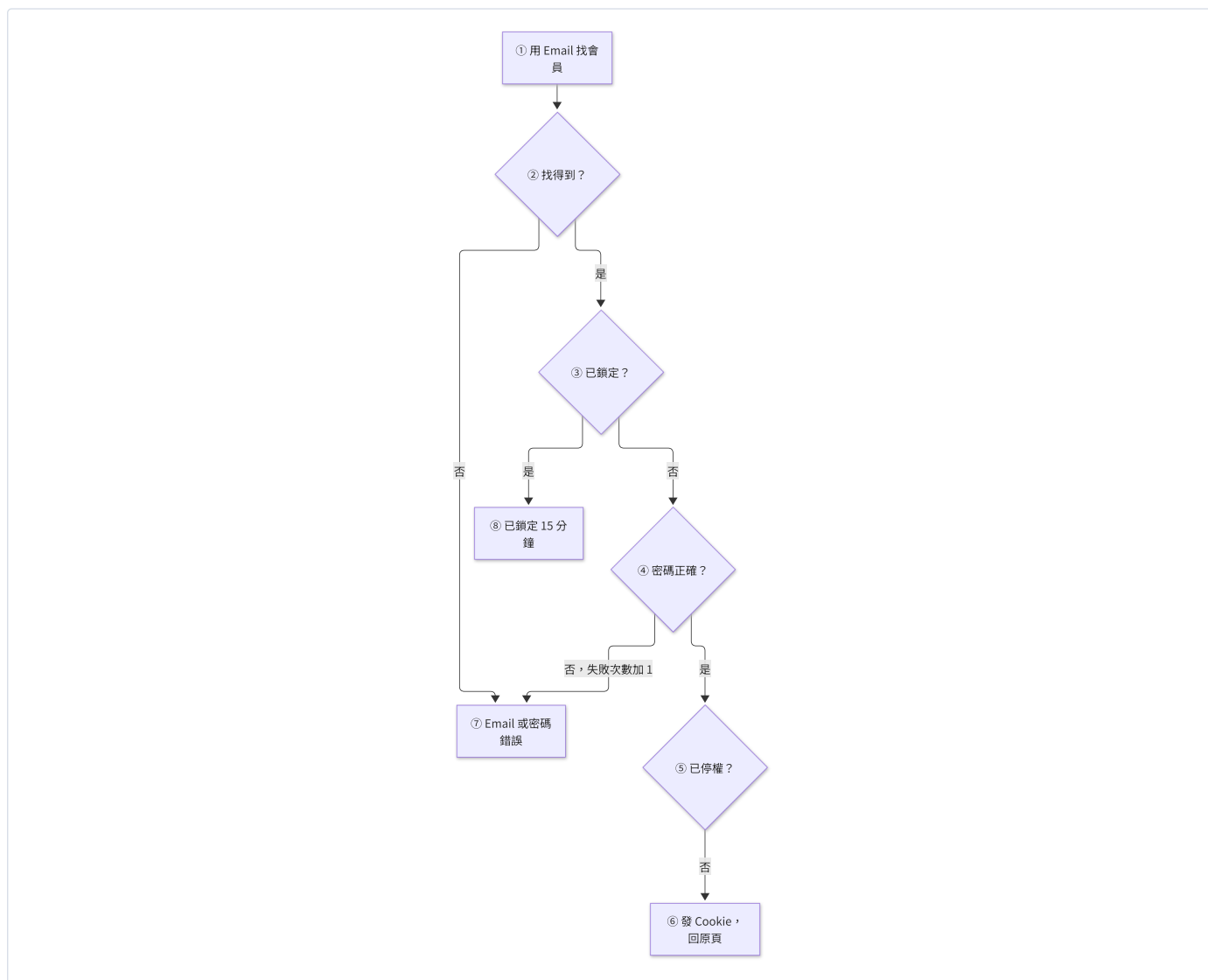


圖 04-2 登入的判斷順序

編號	判斷	為什麼這樣排
② → ⑦	Email 不存在	和密碼錯誤同一句話，攻擊者無法用登入頁探測哪些 Email 已註冊
③ → ⑧	<code>LockoutEnd</code> 在未來	鎖定期間連 正確密碼也不接受 ，猜中了也沒用
④ → ⑦	密碼錯	<code>CheckPasswordSignInAsync(..., lockoutOnFailure: true)</code> 讓 <code>AccessFailedCount</code> 加 1；第 5 次失敗當下就寫入 <code>LockoutEnd</code> ，那一次直接回 ⑧
⑤	<code>IsActive = false</code>	放在密碼驗證之後：不知道密碼的人看不出這個帳號被停權（Section 12 實作停權）
⑥	發 Cookie，轉回 <code>returnUrl</code>	只接受站內網址（ <code>Url.IsLocalUrl</code> ），防止 <code>returnUrl=https://釣魚網站</code> 的開放轉址

限制與注意事項

- 註冊不代表能呼叫 API（Section 01 的驗收句）：註冊只建立 `AspNetUsers` 一列，沒有角色、沒有 Grant。
- 本單元不做 Email 驗證與忘記密碼；正式上線需要寄信服務，超出 30 小時範圍。
- 示範帳號的密碼來自 User Secrets 的 `Seed:DemoPassword`、`Seed:AdminPassword`；沒設定的話種子程式只記警告，那些帳號無法登入。
- 已登入的 Cookie 在 8 小時內有效。管理者停權或改角色後，要讓舊 Cookie 立即失效需要 Security Stamp 驗證（Section 12）。

名詞對照

名詞	白話說明	在本課程中是
認證（Authentication）	確認「你是誰」	Identity + Cookie <code>McpPlatform.Auth</code>
授權（Authorization）	決定「你能做什麼」	政策 <code>AdminOnly</code> 、 <code>Member</code>
宣告（Claim）	身分裡的一項資料，例如 Id、Email、角色	<code>NameIdentifier</code> 、 <code>display_name</code>
政策（Policy）	有名字的一組授權條件	<code>AuthPolicies.AdminOnly</code>
防偽權杖	證明表單是本站產生的隱藏欄位	<code>__RequestVerificationToken</code>
帳號鎖定	失敗太多次暫停登入	5 次、15 分鐘

教師示範 (25 分鐘)

示範情境：用三種身分（訪客、王小明、系統管理員）打同一組網址，證明後臺與管理 API 只有管理者進得去、王小明看不到林小華的 API；再示範登入失敗與鎖定。

示範步驟

1. 設定示範密碼（只需一次），再啟動網站：

```
# 在 McpPlatform/ 根目錄執行；兩個密碼請自訂，需含大小寫、數字、符號，至少 8 字元
dotnet user-secrets set "Seed:DemoPassword" "<自訂示範密碼>" --project src/McpPlatform.Web
dotnet user-secrets set "Seed:AdminPassword" "<自訂管理者密碼>" --project src/McpPlatform.Web
dotnet run --project src/McpPlatform.Web --launch-profile http
```

2. 開 <http://localhost:5199/Account/Login>，Email 填 `a@demo.test`、密碼故意打錯。

The screenshot shows the MCP Server API platform login page. The header is dark green with the text 'MCP Server API 平台' and navigation links 'API 目錄' and '會員中心'. On the right, there are links '登入' and '註冊'. The main content area has a light beige background with a central login form titled '登入'. The form contains an error message: 'Email 或密碼錯誤。'. Below this, there are input fields for 'Email' (containing 'a@demo.test') and '密碼'. There is a checkbox for '記住我 (8 小時)' and a dark green '登入' button. At the bottom of the form, there is a link '還沒有帳號? 註冊會員'. The footer of the page is dark green with the text '© 2026 MCP Server API 平台 | [環境檢查](#)'.

圖 04-3 登入失敗的畫面

畫面重點：

1. ①紅字「Email 或密碼錯誤。」：沒有說是哪一個錯；換成不存在的 Email 也是這一句。
2. ②Email 欄位保留、密碼欄位清空：`asp-for` 會帶回使用者輸入，但 `type=password` 欄位不回填。
3. ③右上角是「登入／註冊」：還沒登入，導覽列也沒有「管理後臺」。
4. 用正確密碼登入後打開會員中心。



圖 04-4 以王小明登入後的會員中心

畫面重點：

1. ①右上角「王小明」與「登出」按鈕：名字來自 `display_name` 宣告；「登出」是一個 POST 表單。
2. ②導覽列沒有「管理後臺」： `User.IsInRole("Admin")` 為 false 所以不顯示 —— 但這不是保護，下一步直接輸入網址。
3. ③三張卡片都是王小明的： `ListOwnedAsync(User.GetUserId())`；每張卡片底下的「查看…詳細」連到 `/Member/Apis/Details/{版本 Id}`。
4. 網址列直接輸入 `http://localhost:5199/Admin`。



圖 04-5 一般會員直連後臺被導到「沒有權限」

畫面重點：

- 1. ①網址變成 `/Account/AccessDenied?ReturnUrl=%2FAdmin`：Cookie 驗證通過（知道你是王小明），但 `AdminOnly` 政策不通過，所以是「沒有權限」而不是「請登入」。
- 2. ②頁面說明目前登入身分：讓使用者知道要換帳號，而不是以為網站壞了。
- 3. 在另一個終端機用 PowerShell 以三種身分打網址，得到本單元實測的權限矩陣：

身分	/Member	/Admin	/api/admin/apis	/Member/Apis/Details/7 (林小華的)
訪客	302 登入	302 登入	401	-
王小明	200	302 /Account/AccessDenied	403	404
系統管理員	-	200	200 (JSON 7 筆)	-

程式碼走讀：Area 授權慣例

這個類別在網站啟動時對每個 Controller 執行一次，放在 `Security/AreaAuthorizationConvention.cs`，由 `Program.cs` 的 `AddControllersWithViews` 註冊。

```
// 範例/04_會員註冊登入與角色授權/McpPlatform/src/McpPlatform.Web/Security/AreaAuthorizationConvention.cs (AreaAuthorizationConvention 類別)
public sealed class AreaAuthorizationConvention(string area, string policy) : IControllerModelConvention
{
    public void Apply(ControllerModel controller)
    {
        var controllerArea = controller.RouteValues.TryGetValue("area", out var value) ? value : null; // 讀 [Area("...")]
        if (string.Equals(controllerArea, area, StringComparison.OrdinalIgnoreCase))
            controller.Filters.Add(new AuthorizeFilter(policy)); // 整個 Controller 加授權
    }
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	<code>[Area("Admin")]</code> 會變成 Controller 的路由值 <code>area=Admin</code> ，這裡把它讀出來	依 Area 判斷，而不是依類別名稱或命名空間；搬移檔案不影響
②	加一個 <code>AuthorizeFilter(policy)</code> ，效果等同在類別上寫 <code>[Authorize(Policy = "AdminOnly")]</code>	慣例在啟動時套用，不可能「忘記」；若某個 Action 真的要公開，必須明確寫 <code>[AllowAnonymous]</code> ，一眼就看得到的

執行結果：上面權限矩陣中王小明打 `/Admin` 、 `/Admin/Api` 都被導到 `AccessDenied` ； `Admin` Area 的三個 Controller（ `HomeController` 、 `ApiController` 、 `DatabaseController` ）本身都沒有寫 `[Authorize]` 。

學員實作（60 分鐘）

範例專案：範例/04_會員註冊登入與角色授權/McpPlatform/（從 範例/03_MSSQL資料模型與EFCore/McpPlatform/ 複製後往上加）。本單元新增 Infrastructure 的 Identity/（IdentitySeeder、ChineseIdentityErrorDescriber、AppClaimsPrincipalFactory）；Web 的 Security/AreaAuthorizationConvention.cs、Controllers/AccountController.cs、Controllers/Api/AdminApisController.cs、ViewModels/AccountViewModels.cs、Views/Account/_LoginPartial.cshtml、會員中心的 ProfileController 與 API 詳細頁；Core 的 ApiSearchCriteria 多一個 VersionId、IApiCatalogService 多一個 GetOwnedAsync；刪除 Section 02 的 Services/DemoCurrentUser.cs。資料表不變，本單元沒有新的 Migration：Identity 的 7 張表在 Section 03 已經建好。

步驟一 啟用 Identity 與 Cookie（15 分鐘）

目的：讓網站知道怎麼驗證密碼、怎麼記住登入狀態、誰算管理者。

```
// 範例/04_會員註冊登入與角色授權/McpPlatform/src/McpPlatform.Web/Program.cs (Identity、Cookie、政策)
builder.Services.AddIdentity<ApplicationUser, IdentityRole>(options =>
{
    options.Password.RequiredLength = 8; // ④ 至少 8 字元，含大小寫、數字、符號（後四項為預設）
    options.Lockout.MaxFailedAccessAttempts = 5; // ⑤ 連續錯 5 次鎖 15 分鐘
    options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(15);
    options.User.RequireUniqueEmail = true;
})
.AddEntityFrameworkStores<AppDbContext>()
.AddDefaultTokenProviders()
.AddErrorDescriber<ChineseIdentityErrorDescriber>()
.AddClaimsPrincipalFactory<AppClaimsPrincipalFactory>();

builder.Services.ConfigureApplicationCookie(options =>
{
    options.Cookie.Name = "McpPlatform.Auth";
    options.Cookie.HttpOnly = true; // ⑥ JavaScript 讀不到登入 Cookie
    options.ExpireTimeSpan = TimeSpan.FromHours(8);
    options.SlidingExpiration = true;
    options.LoginPath = "/Account/Login";
    options.AccessDeniedPath = "/Account/AccessDenied";
    // ⑦ /api/ 開頭的請求不要轉址到登入頁，直接回 401/403，呼叫端才判斷得出結果。
    options.Events.OnRedirectToLogin = ctx => ApiAwareRedirect(ctx.HttpContext, ctx.RedirectUri, StatusCodes.Status401Unauthorized);
    options.Events.OnRedirectToAccessDenied = ctx => ApiAwareRedirect(ctx.HttpContext, ctx.RedirectUri, StatusCodes.Status403Forbidden);
});

builder.Services.AddAuthorizationBuilder()
.AddPolicy(AuthPolicies.AdminOnly, p => p.RequireAuthenticatedUser().RequireRole(IdentitySeeder.AdminRole)) // ⑧
.AddPolicy(AuthPolicies.Member, p => p.RequireAuthenticatedUser());
```

（同一檔案後面還有 `app.UseAuthentication()`；必須放在 `app.UseAuthorization()` 之前，以及 `ApiAwareRedirect` 函式：路徑以 `/api` 開頭就設狀態碼，否則照常轉址。）

程式說明：

標號	做了什麼	為什麼這樣寫
④	密碼最短 8 字元；大寫、小寫、數字、符號四項要求是 Identity 預設就開啟的	註冊 <code>abc12345</code> 會得到「密碼至少要有一個符號」「密碼至少要有一個大寫英文字母」兩則錯誤（本單元實測）
⑤	連續 5 次鎖 15 分鐘	限制線上暴力猜測；時間不設太長，避免有人故意打錯把別人鎖住太久
—	<code>AddEntityFrameworkStores<AppDbContext>()</code>	會員、角色存在 Section 03 建好的 <code>AspNetUsers</code> 、 <code>AspNetRoles</code>
—	<code>AddErrorDescriber</code> 、 <code>AddClaimsPrincipalFactory</code>	錯誤訊息改中文；登入時在 Cookie 多放一個 <code>display_name</code> 宣告
⑥	Cookie <code>HttpOnly</code>	就算頁面被注入惡意腳本，也讀不到登入 Cookie
⑦	<code>/api/...</code> 未登入回 401、權限不足回 403	預設行為是 302 轉址到登入頁；程式呼叫 API 時會拿到一整頁 HTML 而不是明確的錯誤碼
⑧	<code>AdminOnly</code> = 已登入 + <code>Admin</code> 角色	角色名稱用常數 <code>IdentitySeeder.AdminRole</code> ，種子程式與政策不會拼錯

預期結果：`dotnet build` 顯示 0 個錯誤。

步驟二 Area 授權慣例與全站防偽（10 分鐘）

目的：讓整個 `Admin`、`Member` Area 自動受保護，全站 POST 自動驗證防偽權杖。

```
// 範例/04_會員註冊登入與角色授權/McpPlatform/src/McpPlatform.Web/Program.cs (MVC 選項)
builder.Services.AddControllersWithViews(options =>
{
    options.Filters.Add(new AutoValidateAntiforgeryTokenAttribute()); // ① 全站 POST 一律驗證防偽權杖
    options.Conventions.Add(new AreaAuthorizationConvention("Admin", AuthPolicies.AdminOnly)); // ② 後臺整區要 Admin
    options.Conventions.Add(new AreaAuthorizationConvention("Member", AuthPolicies.Member)); // ③ 會員中心整區要登入
});
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	對 POST、PUT、PATCH、DELETE 自動驗證防偽權杖；GET 不驗	GET 不應該改變資料，所以不需要；一個全域篩選器取代散落各處的 <code>[ValidateAntiForgeryToken]</code>
②	<code>Admin</code> Area 全部套 <code>AdminOnly</code>	教師示範走讀的慣例類別
③	<code>Member</code> Area 全部套 <code>Member</code> （只要登入）	會員中心所有頁面都操作「自己的資料」，一定要知道你是誰

管理用 Web API 不在 Area 裡，所以在 `Controllers/Api/AdminApisController.cs` 明確寫 `[Authorize(Policy = AuthPolicies.AdminOnly)]`，路由 `api/admin/apis`。

預期結果：未登入打 `/Member` 得到 302 轉到 `/Account/Login?ReturnUrl=%2FMember`；打 `/api/admin/apis` 得到 401。

步驟三 註冊與登入（20 分鐘）

目的：做出不能指定角色的註冊表單，以及圖 04-2 的登入流程。

註冊表單 `RegisterViewModel` 只有 `Email`、`DisplayName`、`Password`、`ConfirmPassword` 四個欄位（`ViewModels/AccountViewModels.cs`）。

`AccountController.Register` 用 `userManager.CreateAsync(user, form.Password)` 建立會員——密碼在這一行被雜湊，之後用 `SignInAsync` 直接登入，沒有任何加角色的程式。

登入是本步驟的重點：

```
// 範例/04_會員註冊登入與角色授權/McpPlatform/src/McpPlatform.Web/Controllers/AccountController.cs (登入 POST)
[HttpPost, AllowAnonymous]
public async Task<IActionResult> Login(LoginViewModel form, string? returnUrl)
{
    if (!ModelState.IsValid) return View(form);

    var user = await userManager.FindByEmailAsync(form.Email);
    if (user is null)
    {
        ModelState.AddModelError(string.Empty, LoginFailed); // ① 帳號不存在與密碼錯誤同一句話
        return View(form);
    }

    var check = await signInManager.CheckPasswordSignInAsync(user, form.Password, lockoutOnFailure: true); // ②
    if (check.IsLockedOut)
    {
        ModelState.AddModelError(string.Empty, "登入失敗次數過多，帳號已鎖定 15 分鐘。");
        return View(form);
    }
    if (!check.Succeeded)
    {
        ModelState.AddModelError(string.Empty, LoginFailed);
        return View(form);
    }
    if (!user.IsActive) // ③ 密碼對了才告知停權
    {
        ModelState.AddModelError(string.Empty, "此帳號已停權，請聯絡管理者。");
        return View(form);
    }

    await signInManager.SignInAsync(user, form.RememberMe);
    return LocalRedirect(Url.IsLocalUrl(returnUrl) ? returnUrl : "/"); // ④ 只接受站內網址
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
③	<code>LoginFailed</code> 常數 = 「Email 或密碼錯誤。」	兩種失敗同一句話，無法用來探測帳號是否存在
④	只驗證密碼、不發 Cookie； <code>lockoutOnFailure: true</code> 讓失敗次數累計	先驗密碼再檢查停權（⑤），停權的人不會拿到 Cookie
⑤	停權訊息只在密碼正確之後出現	對應圖 04-2 的 ⑤； <code>IsActive</code> 是 Section 03 在 <code>ApplicationUser</code> 加的欄位
⑥	<code>returnUrl</code> 必須是站內網址，否則回首頁	<code>LocalRedirect</code> 遇到外部網址會丟例外，加上 <code>Url.IsLocalUrl</code> 先過濾，避免開放轉址

預期結果（本單元實測，用一個剛註冊的測試帳號）：錯誤密碼連送 5 次，前 4 次顯示「Email 或密碼錯誤。」，**第 5 次**就顯示「登入失敗次數過多，帳號已鎖定 15 分鐘。」；之後用正確密碼仍是鎖定訊息。資料庫裡該會員的 `LockoutEnd` 是 15 分鐘後的 UTC 時間，`PasswordHash` 以 `AQAAAAIAAYag` 開頭。

常見改錯：用 `PasswordSignInAsync(email, ...)` 一步登入 —— 它會直接發 Cookie，沒有機會先檢查 `IsActive`，停權的人也登得進來。

步驟四 管理者種子與本人資料範圍（15 分鐘）

目的：用設定檔的密碼建立可登入的示範帳號與 `Admin` 角色；讓會員只看得到自己的 API。

```
// 範例/04_會員註冊登入與角色授權/McpPlatform/src/McpPlatform.Infrastructure/Identity/IdentitySeeder.cs (SeedAsync)
public static async Task SeedAsync(
    UserManager<ApplicationUser> users, RoleManager<IdentityRole> roles,
    string? demoPassword, string? adminPassword, ILogger logger)
{
    if (!await roles.RoleExistsAsync(AdminRole)) // ① 角色只建一次
        await roles.CreateAsync(new IdentityRole(AdminRole));

    var admin = await users.FindByIdAsync("admin");
    if (admin is not null)
    {
        if (!await users.IsInRoleAsync(admin, AdminRole))
            await users.AddToRoleAsync(admin, AdminRole); // ② 角色由種子程式指派，不由註冊表單
        await EnsurePasswordAsync(users, admin, adminPassword, "Seed:AdminPassword", logger);
    }

    foreach (var id in new[] { "teacher", "member-a", "member-b" })
    {
        if (await users.FindByIdAsync(id) is { } user)
            await EnsurePasswordAsync(users, user, demoPassword, "Seed:DemoPassword", logger);
    }
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	<code>Admin</code> 角色不存在才建立	每次啟動都會執行，必須幕等
②	只有 <code>admin</code> 這個帳號被加入 <code>Admin</code> 角色	大綱要求「註冊表單不得讓使用者選擇 Admin」；角色的唯一來源是種子程式與（Section 12 的）管理介面
—	<code>EnsurePasswordAsync</code> 只替沒有密碼的帳號呼叫 <code>AddPasswordAsync</code>	已設定的密碼不會被覆蓋；密碼值來自 <code>Seed:DemoPassword</code> 、 <code>Seed:AdminPassword</code> 設定，不在程式碼裡

會員讀自己 API 的詳細頁，擁有者條件寫在 Service：

```
// 範例/04_會員註冊登入與角色授權/McpPlatform/src/McpPlatform.Web/Areas/Member/Controllers/ApiController.cs (Details)
[HttpGet]
public async Task<IActionResult> Details(int id, CancellationToken ct)
{
    var api = await catalog.GetOwnedAsync(User.GetUserId(), id, ct); // ① 擁有者 = 登入者，查詢條件寫在 Service
    if (api is null)
        return NotFound(); // ② 不回 403：不透露「這個 Id 存在但不是你的」
    return View(api);
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	<code>GetOwnedAsync</code> 以 <code>ApiSearchCriteria(OwnerId: ownerId, VersionId: versionId)</code> 查詢，兩個條件同時成立才有結果	「先查出來再比對擁有者」容易漏寫比對；把擁有者放進查詢條件，查不到就是查不到
②	別人的與不存在的都回 404	回 403 等於告訴攻擊者「Id 7 存在」，可以拿來列舉

預期結果：以王小明登入，`/Member/Apis/Details/4`（自己的天氣預報 v1）回 `200`；`/Member/Apis/Details/7`（林小華的包裹追蹤）回 `404`；`/Member/Apis/Details/999` 也回 `404`。

檢查點

做到這裡，你應該可以：

- ☐ 用 `a@demo.test`、`admin@demo.test` 與你在 User Secrets 設定的密碼都能登入，右上角顯示「王小明」「系統管理員」
- ☐ 以王小明登入時，導覽列沒有「管理後臺」，直接輸入 `/Admin` 被導到「沒有權限」
- ☐ 註冊一個新帳號後，「我的資料」的角色顯示「一般會員」
- ☐ `SELECT PasswordHash FROMAspNetUsers` 看不到任何明碼密碼

常見錯誤

症狀（錯誤訊息原文）	原因	處理
用 <code>sqlcmd</code> 刪除 <code>AspNetUsers</code> 資料：訊息 1934 ... DELETE 失敗，因為下列 SET 選項的設定錯誤： 'QUOTED_IDENTIFIER'。	<code>AspNetUsers</code> 有 Identity 建立的篩選索引（ <code>UserNameIndex</code> ）， <code>sqlcmd</code> 預設 <code>QUOTED_IDENTIFIER OFF</code>	<code>sqlcmd</code> 加 <code>-I</code> 參數；SSMS 預設已是 ON，不會遇到
手寫 <code><form action="/Account/Logout" method="post"></code> 按登出得到 400	全站 POST 都要防偽權杖，手寫的 <code><form></code> 不會自動帶	改用 <code><form asp-controller="Account" asp-action="Logout" method="post"></code> ，Tag Helper 會加上隱藏欄位
註冊時出現「密碼至少要有一個符號（例如 # 或 !）。」「密碼至少要有一個大寫英文字母。」	密碼 <code>abc12345</code> 不符合 Identity 預設規則	這是預期的保護；改用 <code>Test#2026pw</code> 這類含四種字元的密碼
示範帳號登不進去，終端機有 未設定 <code>Seed:DemoPassword</code> ， <code>a@demo.test</code> 暫時無法登入。	沒有執行教師示範第 1 步的 <code>dotnet user-secrets set</code>	設定後重新啟動網站；種子程式在啟動時執行

單元驗收（15 分鐘）

驗收標準：一般會員直連後臺或管理 API 均失敗；A 會員無法讀取 B 的私人資料。

驗收方式：

1. **一般會員直連後臺失敗：**以 `a@demo.test` 登入，網址列輸入 `/Admin`、`/Admin/Apis`、`/Admin/Database`，三者都被導到 `/Account/AccessDenied`（圖 04-5）。
2. **一般會員直連管理 API 失敗：**同一個瀏覽器分頁打 `/api/admin/apis`，得到 `403`，沒有任何 JSON；登出後再打一次得到 `401`。換 `admin@demo.test` 登入，得到 `200` 與 7 筆 JSON。
3. **A 會員無法讀取 B 的私人資料：**以王小明登入，打 `/Member/Apis/Details/7`（林小華的包裹追蹤），得到 `404`；打 `/Member/Apis/Details/4`（自己的），得到 `200`。
4. **反向驗證：**用任何 HTTP 工具送出註冊表單，額外加兩個欄位 `Role=Admin`、`IsActive=false`。本單元實測：註冊成功（`302 /Member`），但該帳號打 `/api/admin/apis` 仍是 `403`；資料庫裡 `IsActive = 1`、`AspNetUserRoles` 0 筆——兩個偷塞的欄位都被忽略。

回顧：Identity 負責密碼、鎖定與 Cookie；授權分三層——Area 慣例擋後臺、政策擋 API、Service 查詢條件擋別人的資料。下一單元（Section 05）讓會員建立真正的 API 草稿與版本，並驗證 JSON Schema。

延伸閱讀

- [ASP.NET Core Identity 簡介](#)——`UserManager`、`SignInManager` 與密碼雜湊。
- [ASP.NET Core 以原則為基礎的授權](#)——Policy、Requirement 與 `AddAuthorizationBuilder`。
- [ASP.NET Core 角色授權](#)——`RequireRole` 與 `[Authorize(Roles = ...)]`。
- [防止跨網站請求偽造（XSRF/CSRF）攻擊](#)——`AutoValidateAntiforgeryTokenAttribute` 的適用範圍。

PART 2

第二篇

API 上架與授權審核

S05～S07

把既有 API 整理成可審核的呼叫契約並管理版本，建立送審、核准發布、退回與下架的狀態流程，再讓會員申請使用，由管理者核准後才建立具期限、額度與 Scope 的 ApiGrant。

第 5 章 API目錄規格與版本管理

API 名稱、分類、用途、URL、HTTP Method、認證方式、參數 JSON Schema、請求範例、版本號；草稿 CRUD、新增版本、Schema 驗證、送審前檢查、憑證加密、公開目錄詳細頁

第 6 章 API上架送審與管理者審核

狀態轉換、審核備註、不可變版本快照、送審者與審核者分離、交易與併發衝突；送審、待審清單、核准發布、退回修訂、API 下架

第 7 章 API使用申請與授權審核

用途、申請版本、有效期間、額度、Scope；Approval 與 ApiGrant 分離；使用申請、個人進度、管理者核准與駁回、Grant 建立與撤銷

第 5 章

API目錄規格與版本管理

API 名稱、分類、用途、URL、HTTP Method、認證方式、參數 JSON Schema、請求範例、版本號；草稿 CRUD、新增版本、Schema 驗證、送審前檢查、憑證加密、公開目錄詳細頁

Section 05 | API 目錄、規格與版本管理

把 API 寫成一份完整契約，送審前逐項檢查

120 分鐘 | 觀念 20 · 示範 25 · 實作 60 · 驗收 15

API 規格書

URL

Method

JSON Schema

請求範例

https://api.example.com

GET / POST

```
{  
  "type": "object",  
  "properties": {  
    "keyword": {  
      "type": "string"  
    }  
  }  
}
```

```
{  
  "keyword": "茶"  
}
```

v1

v2

檢查清單

JSON Schema

送審前檢查

版本

Data Protection

參數的規格書

內容齊了才能送

送審後內容凍結

金鑰加密存放

缺資訊不能送審 | 只能改自己的草稿

課程時數2 小時（120 分鐘）

當堂成果會員 API 草稿及版本編輯功能

範例專案範例 / 05_API 目錄規格與版本管理

MCP Server 治理平台實戰手冊（30HRS）

— 67 —

學習目標

上完這個單元，你能夠：

- 把一支既有的 HTTP JSON API 整理成可審核的**呼叫契約**：名稱、分類、用途、URL、Method、認證方式、輸入與輸出 JSON Schema、請求範例、版本號。
- 區分**存檔檢查**（格式錯就不能存）與**送審前檢查**（內容不齊就不能送審），並用 JsonSchema.Net 驗證「Schema 本身合法」與「範例符合 Schema」。
- 實作草稿的新增、修改、刪除與「建立新版本」，並保證**已送審的版本不能改、會員只能改自己的草稿**。
- 用 ASP.NET Core Data Protection 加密上游 API Key，證明金鑰明文不會出現在任何頁面，公開目錄只查得到已發布版本。
- 寫一個手動調整預設值的 Migration，讓既有資料在加欄位後仍然有效。

先備知識

- 完成 Section 04：可以用 `a@demo.test`（王小明）、`b@demo.test`（林小華）登入；`Member` Area 需要登入；`User.GetUserId()` 取得登入者 Id。
- Section 03 的 `ApiDefinitions`（一支 API）與 `ApiVersions`（它的版本）、`ApiCredentialReferences`（憑證，與公開規格分表）。
- 看得懂 JSON；知道 JSON Schema 的 `type`、`properties`、`required`、`maxLength` 關鍵字（本單元會用到的只有這幾個）。

觀念說明（20 分鐘）

為什麼要把 API 整理成「呼叫契約」

平台之後會代替 AI 呼叫這支 API（Section 08、09）。如果登錄時只寫一個網址，平台不知道要帶什麼參數、參數長什麼樣、要不要帶金鑰；MCP Client 也不知道該傳什麼。所以登錄的是一份契約：

欄位	誰會用到	缺了會怎樣
名稱、分類、用途說明	使用者在目錄搜尋；Section 09 映射成 MCP Tool 的描述	AI 不知道何時該用這個工具
URL、HTTP Method	Section 08 的 Gateway 組出請求	呼叫不到或呼叫錯方法
認證方式	Gateway 決定要不要帶 API Key 標頭	上游回 401
輸入參數 Schema	Section 09 的 MCP <code>inputSchema</code> ；Section 11 呼叫前驗證參數	AI 亂傳參數，錯誤一路打到上游
輸出 Schema	使用者理解回傳內容；審核者評估風險	審核者無從判斷會回傳什麼資料
請求範例	審核者與使用者一看就懂；平台用它驗證 Schema 寫得對不對	契約寫錯了沒人發現
版本號	申請、授權、呼叫都綁定版本	改了 URL 卻沒人重新審核

JSON Schema：參數的規格書

JSON Schema 是用 JSON 描述「一份 JSON 資料該長什麼樣子」的標準。例如商品查詢的輸入：

```
{"type": "object", "properties": {"keyword": {"type": "string", "maxLength": 50}}}
```

意思是：輸入是一個物件，可以有 `keyword` 欄位，它是字串、最多 50 字。`{"keyword": "茶"}` 符合；`{"keyword": 123}` 不符合（型別錯）。

本課程用 `JsonSchema.Net 9.4.0` 做兩種驗證：

驗證	問題	用什麼驗
Schema 本身合法嗎	<code>{<code>"type": "objekt"</code>}</code> 拼錯了	用 Draft 2020-12 的後設 Schema（描述 Schema 的 Schema）檢查
資料符合 Schema 嗎	範例 <code>{<code>"address": "台北"</code>}</code> 但 Schema 規定至少 5 字	用使用者寫的 Schema 檢查範例

MCP 規定工具的 `inputSchema` 最上層必須是物件（`"type": "object"`），因為 AI 呼叫工具時傳的是「參數名稱 → 值」的一組對應。這是送審前檢查的其中一項。

存檔檢查與送審前檢查

下圖是一個版本從「新增」到「可以送審」要通過的兩道檢查。注意：草稿可以不完整，但格式錯的資料連存都不能存。

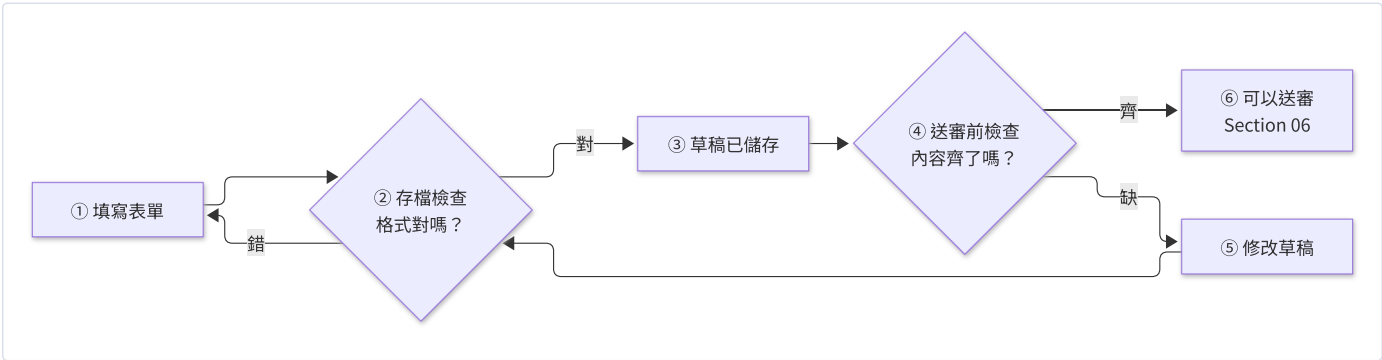


圖 05-1 存檔檢查與送審前檢查

編號	檢查什麼	不通過時
②	名稱、分類必填；Method 是 <code>GET</code> / <code>POST</code> ；URL 是 <code>http(s)</code> 開頭的完整網址、不含帳號與 <code>#</code> ；兩個 Schema 是合法 JSON Schema；範例是合法 JSON	表單帶著錯誤訊息回到原頁， 不存檔
④	用途說明至少 10 字；端點用 HTTPS（ <code>localhost</code> 除外）；輸入 Schema 最上層是物件；輸出 Schema 不是空的 <code>{}</code> ；請求範例不是 <code>{}</code> 且符合輸入 Schema；認證方式為 API Key 時已設定憑證	草稿照樣存在，但詳細頁列出未通過項目， 不能送審

為什麼分兩道？寫規格常常要分好幾次完成；如果每次存檔都要求全部齊備，使用者寫到一半根本存不了。反過來，格式錯的資料（壞掉的 JSON）存進去只會讓後面每個讀它的地方出錯，所以一開始就擋。

版本：送審後就凍結

狀態	能不能修改	要改怎麼辦
Draft（草稿）	可以；也可以刪除	—
Rejected（被退回）	可以，改完重新送審	—
Submitted、Published、Retired	不行	按「建立新版本」，系統複製上一版內容成為 v2 草稿

大綱：「變更 URL、Method、Schema 或權限需求時建立新版本並重新送審。」已發布的 v1 可能已有會員持有 Grant、正在被 AI 呼叫；如果能原地修改，等於繞過審核把呼叫導到別的地方。同一支 API 同時只能有一個草稿或審核中的版本，避免兩個草稿互相覆蓋。

憑證：只寫不讀

有些上游 API 需要 API Key。它和規格分表存（ApiCredentialReferences），而且：

- 1. 加密存檔：用 ASP.NET Core Data Protection 加密。資料庫裡是 CfDJ8... 開頭的密文，就算資料庫被整份拿走，沒有金鑰檔也解不開。
- 2. 只寫不讀：設定後任何頁面都不再顯示金鑰內容，只顯示「已設定：X-API-Key」。
- 3. 公開目錄不碰這張表：目錄查詢根本沒有 JOIN ApiCredentialReferences，回傳的 DTO（ApiPublicDetailDto）也沒有任何憑證欄位。只有 Section 08 的 Gateway 在呼叫上游當下才解密。

限制與注意事項

- 本單元只到「送審前檢查通過」，送審按鈕在 Section 06 才出現。
- 驗證規則只檢查 Schema 的結構與範例，不會真的呼叫上游 API 確認網址存在（那是 Section 08 測試台的事）。
- Data Protection 的金鑰檔在開發時存在 %LOCALAPPDATA%\ASP.NET\DataProtection-Keys。金鑰檔遺失，已加密的憑證就全部解不開；部署時要把金鑰存到固定位置並備份（Section 15）。
- JsonSchema.Net 的錯誤訊息是英文，例如 Value should be at least 5 characters；本課程保留原文並加上欄位位置。

名詞對照

名詞	白話說明	在本課程中是
呼叫契約	一支 API「怎麼叫、傳什麼、回什麼」的完整規格	一個 ApiVersions 列
JSON Schema	描述 JSON 資料格式的 JSON	InputSchema、OutputSchema 欄位
後設 Schema	描述「Schema 該長什麼樣」的 Schema	JsonSchema.Net 的 MetaSchemas.Draft202012
送審前檢查	送審前必須全部通過的條件清單	ApiDraftService.CheckReadinessAsync
Data Protection	ASP.NET Core 內建的加解密服務	DataProtectionSecretProtector

教師示範（25 分鐘）

示範情境：王小明登錄一支新的「郵遞區號查詢」API。先故意填錯格式，看存檔被擋；再存一份不完整的草稿，看送審前檢查列出缺項；補齊後全部通過。最後切到林小華，證明他動不了王小明的草稿，並確認金鑰明文沒有出現在任何頁面。

示範步驟

1. 在 **範例/05_API目錄規格與版本管理/McpPlatform** 先套用本單元的 Migration，再啟動網站：

```
dotnet ef database update --project src/McpPlatform.Infrastructure --startup-project src/McpPlatform.Web
dotnet run --project src/McpPlatform.Web --launch-profile http
```

輸出含 **Applying migration '20260926110825_AddVersionSpecFields'**。已在 Section 03 匯入示範資料的資料庫，另在 SSMS 執行 **sql/03_demo_examples.sql**（補上三支教師示範 API 的請求範例，可重複執行）。

2. 以王小明登入，會員中心 → 「新增 API」。名稱「郵遞區號查詢」、分類「生活」、端點 URL 填 **ftp://zip.example.test/api**、輸入 Schema 填 **{"type":"objekt"}**、請求範例填 **{**，按「建立草稿」。

MCP Server API 平台 API 目錄 會員中心 王小明 登出

會員中心
總覽
我的 API 與版本
新增 API
我的資料
使用申請 (Section 07)
呼叫紀錄 (Section 13)

新增 API

建立後是 v1 草稿。草稿可以反覆修改；存檔只檢查格式，內容是否齊備由「送審前檢查」判斷。

基本資料

API 名稱 分類
郵遞區號查詢 生活

用途說明

呼叫規格

HTTP 方法 認證方式 端點 URL
GET 不需認證 ftp://zip.example.test/api
端點 URL 必須是 http:// 或 https:// 開頭的完整網址，不可含帳密或 #，最多 500 字。

參數與範例

輸入參數 Schema (JSON Schema) 輸出 Schema (JSON Schema)
{ "type": "objekt" } {}
不是合法的 JSON Schema : /type type: Value is "string" but should be "array"; /type enum: Value should match one of the values specified by the enum

請求範例 (JSON)
{
請求範例必須是合法的 JSON。

建立草稿 取消

© 2026 MCP Server API 平台 | 環境檢查

圖 05-2 存檔檢查擋下三個格式錯誤

畫面重點：

1. ①端點 URL 下方的紅字：**ftp://** 不是 **http(s)**，Service 的 **ValidateForSave** 擋下。
2. ②輸入 Schema 下方的紅字：**不是合法的 JSON Schema : /type type: Value is "string" but should be "array"; /type enum: Value should match one of the values specified by the enum**。後設 Schema 規定 **type** 必須是 **object**、**string** 等固定名稱之一（或這些名稱組成的陣列），**objekt** 兩者都不是。
3. ③請求範例下方的紅字：**{** 不是完整的 JSON。三個錯誤一次全部列出，不是改一個才看到下一個。
4. 改成格式正確但內容不完整的版本（URL 用 **http://zip.example.test/api/zip**、認證方式選「API Key 標頭」、輸出 Schema 留 **{}**、範例 **{"address": "台北"}**），存檔成功，轉到詳細頁。示範資料裡王小明的「匯率換算 v1」也是同樣不完整的草稿：



圖 05-3 「匯率換算 v1」的送審前檢查

畫面重點：

- ①左欄六項檢查：綠色「✓ 通過」與紅色「✗ 未通過」並列，每項下方灰字說明原因，例如「目前是空的 {}」。
- ②紅字「有未通過的項目，不能送審。」： `ApiDraftService.IsReady` 為 false。
- ③「修改草稿」「刪除草稿」按鈕：只有 `Draft` 與 `Rejected` 版本才出現；打開已送審的「天氣預報查詢 v2」詳細頁，這兩個按鈕都不會出現。
- ④右欄「規格」：顯示輸入 Schema 與範例，都是 `{}`。
- 補齊用途說明、改成 `https://`、填輸出 Schema、範例改成 `{"address": "臺北市中正區重慶南路一段122號"}`，再到「設定 API Key」輸入金鑰 `sk-demo-9f3a7c2e41`。本單元實測，郵遞區號查詢的送審前檢查從「6 項中 1 項通過」變成「6 項全部通過」。
- 登出，開 `/Catalog/Details/1`（商品查詢，已發布）。

商品查詢 v1

零售 | 提供者：教師示範帳號 | 發布於 2026-09-20 18:00

說明	依關鍵字查詢商品名稱、分類、價格與庫存。
端點	GET http://localhost:5080/api/products
認證	不需認證

要呼叫這支 API，請登入後提出使用申請（Section 07）。

輸入參數 Schema

```
{"type":"object","properties":{"keyword":{"type":"string","maxLength":50}}}
```

請求範例

```
{"keyword":"茶"}
```

圖 05-4 未登入也能看的公開詳細頁

畫面重點：

1. ①標題下的「發布於 2026-09-20 18:00」：資料庫存 UTC 10:00，顯示臺灣時間。
2. ②「認證」欄：只說「不需認證」或「需要 API Key（由平台代為帶入，使用者看不到）」，沒有任何金鑰或標頭值。
3. ③右欄的輸入參數 Schema 與請求範例：`{"keyword": "茶"}`，使用者照著就知道要傳什麼。

程式碼走讀：送審前檢查

這段在 `McpPlatform.Core/Catalog/ApiDraftService.cs`，由會員中心詳細頁 `ApisController.Details` 呼叫。

```
// 範例/05_API目錄規格與版本管理/McpPlatform/src/McpPlatform.Core/Catalog/ApiDraftService.cs (CheckReadinessAsync 與兩個檢查)
public async Task<IReadOnlyList<ReadinessItem>?> CheckReadinessAsync(string ownerId, int versionId, CancellationToken ct)
{
    var d = await GetDetailAsync(ownerId, versionId, ct);
    if (d is null) return null;

    var items = new List<ReadinessItem>
    {
        new("用途說明至少 10 字", d.Description.Trim().Length ≥ 10, $"目前 {d.Description.Trim().Length} 字"),
        CheckEndpoint(d.EndpointUrl),
        CheckInputSchema(d.InputSchema),
        CheckOutputSchema(d.OutputSchema),
        CheckExample(d.InputSchema, d.RequestExample),
        new("API Key 憑證已設定", d.AuthType == ApiAuthType.None || d.HasCredential,
            d.AuthType == ApiAuthType.None ? "此版本不需要憑證" : d.HasCredential ? $"標頭 {d.CredentialHeaderName}" : "尚未設定"),
    };
    return items;
}

private ReadinessItem CheckInputSchema(string schema)
{
    var isObject = TryGetType(schema) == "object";
    return new("輸入 Schema 的最上層是 \"type\": \"object\"", isObject, isObject ? "符合" : "MCP 工具參數必須是一個 JSON 物件");
}

private ReadinessItem CheckExample(string schema, string example)
{
    if (example.Trim() is "" or "{}")
        return new("請求範例符合輸入 Schema", false, "請提供一筆實際的請求範例，不要只寫 {}");
    var problems = schemas.ValidateInstance(schema, example);
    return new("請求範例符合輸入 Schema", problems.Count == 0, problems.Count == 0 ? "通過" : string.Join(" ", problems.Take(2)));
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
—	<code>GetDetailAsync(ownerId, ...)</code> 查不到就回 <code>null</code>	別人的版本連檢查都跑不了，Controller 回 404
—	六項檢查放進一個清單，每項有標題、是否通過、說明	畫面一次列出全部，使用者不必改一項送一次
⑦	讀 Schema 最上層的 <code>type</code> ，必須是 <code>object</code>	MCP 工具參數是「名稱→值」的物件（Section 09）
⑧	用使用者自己寫的 Schema 驗證自己的範例	範例過不了 Schema，代表 Schema 或範例其中之一寫錯；在審核前就發現

執行結果（本單元實測，郵遞區號查詢補齊前）：

```
[未通過] 用途說明至少 10 字 — 目前 5 字
[未通過] 端點使用 HTTPS（本機 localhost 除外） — http://zip.example.test
[通過] 輸入 Schema 的最上層是 "type": "object" — 符合
[未通過] 已描述輸出 Schema — 目前是空的 {}
[未通過] 請求範例符合輸入 Schema — /address minLength:Value should be at least 5 characters
[未通過] API Key 憑證已設定 — 尚未設定
```

學員實作（60 分鐘）

範例專案：範例/05_API目錄規格與版本管理/McpPlatform/（從 Section 04 複製後往上加）。本單元新增 Core 的 `Catalog/DraftContracts.cs`（輸入、結果、介面）與 `Catalog/ApiDraftService.cs`；Infrastructure 的 `Catalog/EfApiDraftRepository.cs`、`Catalog/JsonSchemaValidator.cs` 與 Migration ...
`_AddVersionSpecFields`；Web 的 `Security/DataProtectionSecretProtector.cs`、會員中心 `Apis/` 的 `Index`、`Create`、`Edit`、`Details`、`Credential`、`_SpecForm` 頁面、公開目錄 `Catalog/Details`；`sql/03_demo_examples.sql`。

步驟一 擴充版本欄位與 Migration（10 分鐘）

目的：`ApiVersions` 多兩個欄位——認證方式 `AuthType`、請求範例 `RequestExample`，並讓既有資料在加欄位後仍然有效。

在 `Core/Domain/Enums.cs` 加 `public enum ApiAuthType { None, ApiKeyHeader }`，在 `ApiVersion` 加兩個屬性（`AuthType` 預設 `ApiAuthType.None`、`RequestExample` 預設 `"{}"`）。再加入套件並產生 Migration：

```
# 在 McpPlatform/ 根目錄執行
dotnet add src/McpPlatform.Infrastructure package JsonSchema.Net --version 9.4.0
dotnet ef migrations add AddVersionSpecFields --project src/McpPlatform.Infrastructure --startup-project src/McpPlatform.Web --output-dir Data/Migrations
```

EF Core 對新增的「不可為 null 的字串欄位」一律產生 `defaultValue: ""`。而 `AuthType` 存的是列舉名稱，空字串讀回來時無法轉成列舉，所以要手動改成 `"None"`：

```
// 範例/05_API目錄規格與版本管理/McpPlatform/src/McpPlatform.Infrastructure/Data/Migrations/20260926110825_AddVersionSpecFields.cs (Up 方法)
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.AddColumn<string>(
        name: "AuthType",
        table: "ApiVersions",
        type: "nvarchar(20)",
        maxLength: 20,
        nullable: false,
        defaultValue: "None"); // 手動改：既有資料視為「不需認證」，空字串會讓列舉轉換失敗

    migrationBuilder.AddColumn<string>(
        name: "RequestExample",
        table: "ApiVersions",
        type: "nvarchar(max)",
        nullable: false,
        defaultValue: "{}"); // 手動改：既有資料的範例先給空物件，由提供者之後補上
}
```

程式說明：

欄位	預設值	為什麼
<code>AuthType</code>	<code>'None'</code>	既有 7 個版本加欄位時自動填入；若保持 <code>''</code> ，讀取時 EF Core 丟出列舉轉換例外，整個目錄頁壞掉
<code>RequestExample</code>	<code>'{}'</code>	至少是合法 JSON；送審前檢查會把 <code>{}</code> 判為「請提供實際範例」

Migration 產生後可以、而且應該審閱並修改，只要還沒套用到任何資料庫。

預期結果：`dotnet ef database update ...` 輸出 `Applying migration '20260926110825_AddVersionSpecFields'.`；`SELECT Id, AuthType, RequestExample FROM ApiVersions` 既有 7 列都是 `None` 與 `{}`（執行 `sql/03_demo_examples.sql` 後，前 3 列變成實際範例）。

步驟二 JSON Schema 驗證器（10 分鐘）

目的：包裝 `JsonSchema.Net`，提供「Schema 合法嗎」「資料符合嗎」兩個方法，只回傳看得懂的錯誤。

```
// 範例/05_API目錄規格與版本管理/McpPlatform/src/McpPlatform.Infrastructure/Catalog/JsonSchemaValidator.cs (兩個公開方法與錯誤收集)
public IReadOnlyList<string> ValidateSchema(string schemaJson)
{
    if (!TryParse(schemaJson, out var element, out var parseError)) return [parseError];
    var result = MetaSchemas.Draft202012.Evaluate(element, Options); // ④ 用後設 Schema 檢查 Schema 本身
    return result.IsValid ? [] : Collect(result);
}

public IReadOnlyList<string> ValidateInstance(string schemaJson, string instanceJson)
{
    if (!TryParse(instanceJson, out var instance, out var parseError)) return [parseError];
    JsonSchema schema;
    try { schema = JsonSchema.FromText(schemaJson); }
    catch (Exception ex) when (ex is JsonException or ArgumentException) { return ["Schema 無法解析：" + ex.Message]; }

    var result = schema.Evaluate(instance, Options); // ④ 用 Schema 檢查資料
    return result.IsValid ? [] : Collect(result);
}

private static List<string> Collect(EvaluationResults result) =>
    (result.Details ?? [])
        .Where(d => d.Errors is not null)
        .SelectMany(d => d.Errors!.Where(e => !WrapperKeywords.Contains(e.Key))
            .Select(e => $"{d.InstanceLocation.ToString() is "" ? "(根)" : d.InstanceLocation.ToString()} {e.Key} : {e.Value}"))
        .Distinct()
        .ToList();
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	<code>MetaSchemas.Draft202012.Evaluate</code> 把使用者的 Schema 當成資料，用官方後設 Schema 檢查	<code>{"type": "objekt"}</code> 是合法 JSON，但不是合法 Schema；只 <code>JsonDocument.Parse</code> 抓不到這種錯
②	<code>JsonSchema.FromText</code> 載入使用者的 Schema，再 <code>Evaluate</code> 範例	<code>Options</code> 設 <code>OutputFormat.List</code> ，每個錯誤各自一筆，帶有資料位置 <code>/address</code>
—	<code>Collect</code> 濾掉 <code>allOf</code> 、 <code>anyOf</code> 、 <code>properties</code> 這類包裝錯誤	實測一個 <code>type</code> 拼錯、 <code>maxLength</code> 為 <code>-1</code> 的 Schema，後設 Schema 驗證產生 11 筆錯誤，其中 7 筆是「某個子 Schema 沒通過」的包裝訊息；只留最底層的才看得懂

預期結果：`ValidateInstance` 以郵遞區號的 Schema 驗證 `{"address": "台北"}`，回傳一筆 `/address minLength: Value should be at least 5 characters`。

步驟三 草稿 Service：只能改自己的、送審後不能改（20 分鐘）

目的：所有業務規則集中在 `ApiDraftService`，Controller 只負責 HTTP。

修改草稿是規則最多的一個方法：

```
// 範例/05_API目錄規格與版本管理/McpPlatform/src/McpPlatform.Core/Catalog/ApiDraftService.cs (UpdateAsync)
public async Task<DraftResult> UpdateAsync(string ownerId, int versionId, ApiDraftInput input, CancellationToken ct)
{
    var version = await repository.FindOwnedVersionAsync(ownerId, versionId, ct);
    if (version is null) return DraftResult.Missing(); // ④ 不是自己的 = 不存在
    if (version.Status is not (ApiVersionStatus.Draft or ApiVersionStatus.Rejected)) // ④ 送審後內容凍結
        return DraftResult.Invalid(new Dictionary<string, string>
        { ["" ] = $"{version.Version} 目前是「{version.Status}」，不能修改；請建立新版本。" });

    var errors = ValidateForSave(input);
    if (errors.Count > 0) return DraftResult.Invalid(errors);

    Apply(input, version.ApiDefinition, version, clock.UtcNow());
    await repository.SaveChangesAsync(ct);
    return DraftResult.Ok(version.Id);
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
②	<code>FindOwnedVersionAsync</code> 的 SQL 條件是 <code>v.Id == versionId && v.ApiDefinition.OwnerId == ownerId</code>	擁有者寫在查詢條件裡，林小華查王小明的草稿結果就是「沒有這筆」
③	只有 <code>Draft</code> 、 <code>Rejected</code> 能改	送審後凍結（圖 05-1 之後的 Section 06）；錯誤訊息直接告訴使用者該「建立新版本」
—	<code>ValidateForSave</code> 通過才寫入	存檔檢查，見下一段
—	時間用注入的 <code>TimeProvider</code> 而不是 <code>DateTimeOffset.UtcNow</code>	Section 14 的測試可以把時間固定住

存檔檢查的完整規則：

```
// 範例/05_API目錄規格與版本管理/McpPlatform/src/McpPlatform.Core/Catalog/ApiDraftService.cs (ValidateForSave)
private Dictionary<string, string> ValidateForSave(ApiDraftInput input)
{
    var errors = new Dictionary<string, string>();
    if (string.IsNullOrEmpty(input.Name) || input.Name.Length > 50) errors[nameof(input.Name)] = "API 名稱為必填，最多 50 字。";
    if (string.IsNullOrEmpty(input.Category) || input.Category.Length > 20) errors[nameof(input.Category)] = "分類為必填，最多 20 字。";
    if ((input.Description ?? "").Length > 500) errors[nameof(input.Description)] = "用途說明最多 500 字。";
    if (input.HttpMethod is not ("GET" or "POST")) errors[nameof(input.HttpMethod)] = "HTTP 方法只能是 GET 或 POST。";

    if (!Uri.TryCreate(input.EndpointUrl, UriKind.Absolute, out var uri)
        || uri.Scheme is not ("http" or "https")
        || !string.IsNullOrEmpty(uri.UserInfo) || !string.IsNullOrEmpty(uri.Fragment) || input.EndpointUrl.Length > 500)
        errors[nameof(input.EndpointUrl)] = "端點 URL 必須是 http:// 或 https:// 開頭的完整網址，不可含帳密或 #，最多 500 字。";

    foreach (var (field, text) in new[] { (nameof(input.InputSchema), input.InputSchema), (nameof(input.OutputSchema), input.OutputSchema) })
    {
        var problems = schemas.ValidateSchema(text); // 必須是合法 JSON Schema
        if (problems.Count > 0) errors[field] = "不是合法的 JSON Schema：" + string.Join("；", problems.Take(3));
    }
    if (!IsJson(input.RequestExample)) errors[nameof(input.RequestExample)] = "請求範例必須是合法的 JSON。";
    return errors;
}
```

程式說明：

規則	擋下什麼	為什麼
錯誤的 Key 用 <code>nameof(input.EndpointUrl)</code> 等欄位名	—	Controller 用同樣的 Key 呼叫 <code>ModelState.AddModelError</code> ，錯誤會出現在對應欄位下方（圖 05-2）
<code>Method</code> 只允許 <code>GET</code> 、 <code>POST</code>	<code>DELETE</code> 、 <code>PUT</code>	大綱第一版只支援受控的 GET／POST JSON API
URL 必須是絕對網址、 <code>http(s)</code> 、無 <code>UserInfo</code> 、無 <code>Fragment</code>	<code>ftp://...</code> 、 <code>https://user:pass@host/...#x</code>	帳密寫在網址裡等於明碼存憑證；Section 08 的 SSRF 防護也從這裡開始
⑥ 兩個 Schema 都要通過後設 Schema	<code>{"type": "objekt"}</code>	壞 Schema 存進去，Section 09 產生 MCP 工具時才爆
<code>(input.Description ?? "")</code>	—	防禦性寫法，見步驟四的常見改錯

預期結果：以林小華登入，`GET` 或 `POST /Member/Apis/Edit/11`（王小明的郵遞區號查詢）都得到 `404`；以王小明打開已送審的 `/Member/Apis/Edit/5`（天氣預報 v2），被 `302` 轉回詳細頁並提示「已送審或已發布，內容不能修改；請建立新版本」。

步驟四 Controller、表單與憑證加密（20 分鐘）

目的：把 Service 接到頁面上，並讓金鑰加密存檔。

修改草稿的 POST 先確認擁有權，再看表單內容：

```
// 範例/05_API目錄規格與版本管理/McpPlatform/src/McpPlatform.Web/Areas/Member/Controllers/ApiController.cs (Edit POST)
[HttpPost]
public async Task<IActionResult> Edit(int id, ApiSpecFormViewModel form, CancellationToken ct)
{
    if (await drafts.GetDetailAsync(User.GetUserId(), id, ct) is null)
        return NotFound(); // ㊟ 先確認是自己的，再看表單內容
    if (ModelState.IsValid)
    {
        var result = await drafts.UpdateAsync(User.GetUserId(), id, form.ToInput(), ct);
        if (result.Succeeded)
        {
            TempData["Notice"] = "草稿已儲存。";
            return RedirectToAction(nameof(Details), new { id });
        }
        AddErrors(result);
    }
    await LoadCategoriesAsync(ct);
    return View(form);
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
㊟	第一件事是確認這個版本屬於登入者，不是就 404	本單元實測的第一版把這行寫在 <code>ModelState.IsValid</code> 裡面：林小華送一份欄位不完整的表單時，程式直接帶著他的表單回到頁面（200），根本沒檢查擁有權。沒有資料外洩，但「越權請求得到 200」代表檢查順序錯了
—	<code>form.ToInput()</code> 轉成 Service 的輸入	表單 ViewModel 沒有 <code>OwnerId</code> 、 <code>Status</code> 、 <code>Version</code> ；擁有者由 <code>User.GetUserId()</code> 決定
—	<code>AddErrors(result)</code> 把 Service 的錯誤逐欄加入 <code>ModelState</code>	欄位名稱對得上，錯誤就顯示在對應欄位下方

`ApiSpecFormViewModel.ToInput()` 把每個欄位補成非 null：`new(Name ?? "", Category ?? "", Description ?? "", ...)`。另外在 `Program.cs` 的 `AddControllersWithViews` 加上 `options.SuppressImplicitRequiredAttributeForNonNullableReferenceTypes = true;`，理由見〈常見錯誤〉。

憑證加密只有幾行，重點在 Purpose 字串：

```
// 範例/05_API目錄規格與版本管理/McpPlatform/src/McpPlatform.Web/Security/DataProtectionSecretProtector.cs
public sealed class DataProtectionSecretProtector(IDataProtectionProvider provider) : ISecretProtector
{
    private readonly IDataProtector _protector = provider.CreateProtector("McpPlatform.UpstreamCredentials.v1");

    public string Protect(string plaintext) => _protector.Protect(plaintext);
    public string Unprotect(string protectedValue) => _protector.Unprotect(protectedValue);
}
```

程式說明：

項目	說明
<code>IDataProtectionProvider</code>	ASP.NET Core 內建（Identity 已經用它加密 Cookie），不必另外安裝
<code>CreateProtector("McpPlatform.UpstreamCredentials.v1")</code>	Purpose 字串把用途隔開：同一把主金鑰，用別的 Purpose 建立的 Protector 解不開這裡加密的值
介面 <code>ISecretProtector</code> 定義在 Core	<code>ApiDraftService</code> 只知道「加密」這個能力，不知道是 Data Protection，Core 不必參考 ASP.NET Core

在 `Program.cs` 註冊：`AddSingleton(TimeProvider.System)`、`AddScoped<ApiDraftService>()`、`AddScoped<IApiDraftRepository, EfApiDraftRepository>()`、`AddSingleton<ISchemaValidator, JsonSchemaValidator>()`、`AddSingleton<ISecretProtector, DataProtectionSecretProtector>()`。

預期結果：設定金鑰後，`SELECT HeaderName, LEFT(ProtectedValue, 40), LEN(ProtectedValue) FROM ApiCredentialReferences` 得到 `X-API-Key | CfDJ8HVw6_3RMZNSrmSbSJeeUgJwffspsYW4Uwi | 155`（你的密文不同，但一定是 `CfDJ8` 開頭）。

常見改錯：把 Controller 的欄位錯誤只靠 DataAnnotations 處理，沒把 Service 的 `DraftResult.Errors` 加回 `ModelState` —— 使用者按下存檔，頁面什麼都沒說就回來了。

檢查點

做到這裡，你應該可以：

- 新增一支 API，格式錯時三個錯誤同時出現在各自欄位下方（圖 05-2）
- 不完整的草稿可以存檔，但送審前檢查有紅色未通過項目（圖 05-3）

□ 已送審的版本沒有「修改」按鈕，直接打 `/Member/Apis/Edit/5` 會被轉回

□ 「我的 API 與版本」頁按「建立新版本」：本單元實測林小華的包裹追蹤（v1 已退回）得到 `v2 (Draft)`，端點等內容複製自 v1；王小明的天氣預報（已有 v2 審核中）被拒絕，提示「已有草稿或審核中的版本，請先完成它再建立新版本。」

□ 資料庫裡的憑證是 `CfDJ8` 開頭的密文

常見錯誤

以下都是撰寫本單元時實際遇到的原文。

症狀（錯誤訊息原文）	原因	處理
送出新增表單得到 500，日誌 <code>System.NullReferenceException: Object reference not set to an instance of an object. at McpPlatform.Core.Catalog.ApiDraftService.ValidateForSave</code>	模型繫結預設把空白欄位轉成 <code>null</code> （ <code>ConvertEmptyStringToNull</code> ），即使屬性宣告成非 <code>null</code> 的 <code>string</code>	在 <code>ToInput()</code> 用 <code>?? ""</code> 補齊；Service 也用 <code>?? ""</code> 防禦
用途說明留空，出現英文 <code>The 用途說明 field is required.</code>	專案開啟 nullable 參考型別時，MVC 自動把非 <code>null</code> 的 <code>string</code> 屬性當成必填	<code>Program.cs</code> 設 <code>SuppressImplicitRequiredAttributeForNonNullableRefere</code> <code>= true</code> ，必填一律明確寫 <code>[Required(ErrorMessage = ...)]</code>
不是合法的 JSON Schema: /type type:Value is "string" but should be "array"; /type enum:Value should match one of the values specified by the enum	<code>type</code> 拼錯（ <code>objekt</code> ）。後設 Schema 允許 <code>type</code> 是固定名稱之一或名稱陣列，兩者都不符合所以報兩個錯	改成 <code>object</code> 、 <code>string</code> 、 <code>integer</code> 、 <code>number</code> 、 <code>boolean</code> 、 <code>ai</code> <code>null</code> 之一
用 <code>sqlcmd</code> 在 Git Bash 查 <code>RequestExample</code> ，中文變亂碼	同 Section 03：輸出編碼是 cp950	在 PowerShell 先執行 <code>[Console]::OutputEncoding = [System.Text.Encoding]::UTF8</code> 再查詢；資料本身是正確的

單元驗收（15 分鐘）

驗收標準：缺少必要資訊不能送審；會員只能修改自己的草稿；憑證不出現在公開目錄。

驗收方式：

1. **缺少必要資訊不能送審：**打開王小明的「匯率換算 v1」詳細頁，送審前檢查至少有 3 項未通過，頁面顯示「有未通過的項目，不能送審。」（圖 05-3）。把缺項補齊後重新整理，6 項全部通過。
2. **會員只能修改自己的草稿：**以林小華登入，對王小明的草稿 `GET /Member/Apis/Edit/{id}`、`POST /Member/Apis/Edit/{id}`、`GET /Member/Apis/Credential/{id}`，三者都必須是 404（本單元實測三者皆 404）。
3. **憑證不出現在公開目錄：**設定金鑰 `sk-demo-9f3a7c2e41` 後，逐頁檢查回應內容是否包含這串字。本單元實測 `/Catalog`、`/Catalog/Details/1~3`、`/api/admin/apis`，連王小明自己的 `/Member/Apis/Details/{id}`、`/Member/Apis/Credential/{id}` 都不含金鑰明文；未發布版本的 `/Catalog/Details/{id}` 回 404。
4. **反向驗證：**對已送審的天氣預報 v2 送出修改（`POST /Member/Apis/Edit/5`），資料庫中 v2 的內容不得改變——Service 回傳「v2 目前是「Submitted」，不能修改；請建立新版本。」。

回顧：契約欄位讓平台與 AI 知道怎麼呼叫；存檔擋格式、送審前檢查擋內容；送審後凍結、改規格就開新版本；憑證加密、只寫不讀、公開查詢不碰它。下一單元（Section 06）加上送審、管理者核准與退回、下架，並用 rowversion 防止同一案被核准兩次。

延伸閱讀

- [JSON Schema 官方入門](#) —— `type`、`properties`、`required` 等關鍵字的完整說明。
- [JsonSchema.Net 文件](#) —— `JsonSchema.FromText`、`Evaluate` 與輸出格式。
- [ASP.NET Core Data Protection 概觀](#) —— Purpose 字串、金鑰管理與輪替。
- [ASP.NET Core 模型驗證](#) —— 不可為 null 的參考型別與隱含 `[Required]`。

第 6 章

API 上架送審與管理者審核

狀態轉換、審核備註、不可變版本快照、送審者與審核者分離、交易與併發衝突；送審、待審清單、核准發布、退回修訂、API 下架



課程時數 2 小時 (120 分鐘)

當堂成果

核准發布、退回、下架流程

範例專案

範例/06_API 上架送審與管理者審核

學習目標

上完這個單元，你能夠：

- 畫出 API 版本的五個狀態與六種合法轉換，並說明每一種轉換由誰觸發、在哪個方法裡檢查。
- 在送審時存下**不可變快照**，並在核准時確認版本內容與快照一致。
- 實作「不得審核本人送出的案件」「退回必須填理由」「已處理的案件不能再審」三條業務規則。
- 用 rowversion 讓兩位管理者同時按下核准時只有一位成功，另一位得到明確的衝突訊息；再用篩選唯一索引在資料庫層保證一件案件只有一筆審核紀錄。
- 讓提供者與管理者都看得到同一條審核歷程：送審、退回（含理由）、再送審、核准、下架。

先備知識

- 完成 Section 05： `ApiDraftService.CheckReadinessAsync` 的送審前檢查、 `Draft` / `Rejected` 才能修改的規則。
- Section 03 的 `ApiVersions.RowVersion` (rowversion 欄位)、 `ApprovalRecords` 的 CHECK 約束。
- Section 04 的 `Admin` Area 慣例：本單元新增的 `ReviewsController` 不寫 `[Authorize]` 也受保護。
- 本單元有一個新 Migration；開始前先在範例目錄執行 `dotnet ef database update ...`。

觀念說明（20 分鐘）

為什麼要審核

會員登錄的 API 會被發布成 MCP 工具，讓其他會員的 AI 呼叫。未經審核就上架，可能發生：端點指向內部網路（Section 08 的 SSRF 風險）、Schema 太寬鬆讓 AI 傳入任意內容、用途說明與實際行為不符。審核讓一位**不是提供者本人**的管理者，看著一份**不會再變動**的規格做決定，並留下紀錄。

版本的狀態與轉換

下圖是一個 API 版本一生可能經過的狀態。看的時候注意：**每一條箭頭都只能經由 `PublicationService` 或 `ApiDraftService` 的某一個方法發生**，沒有任何頁面可以直接把狀態改成別的值。

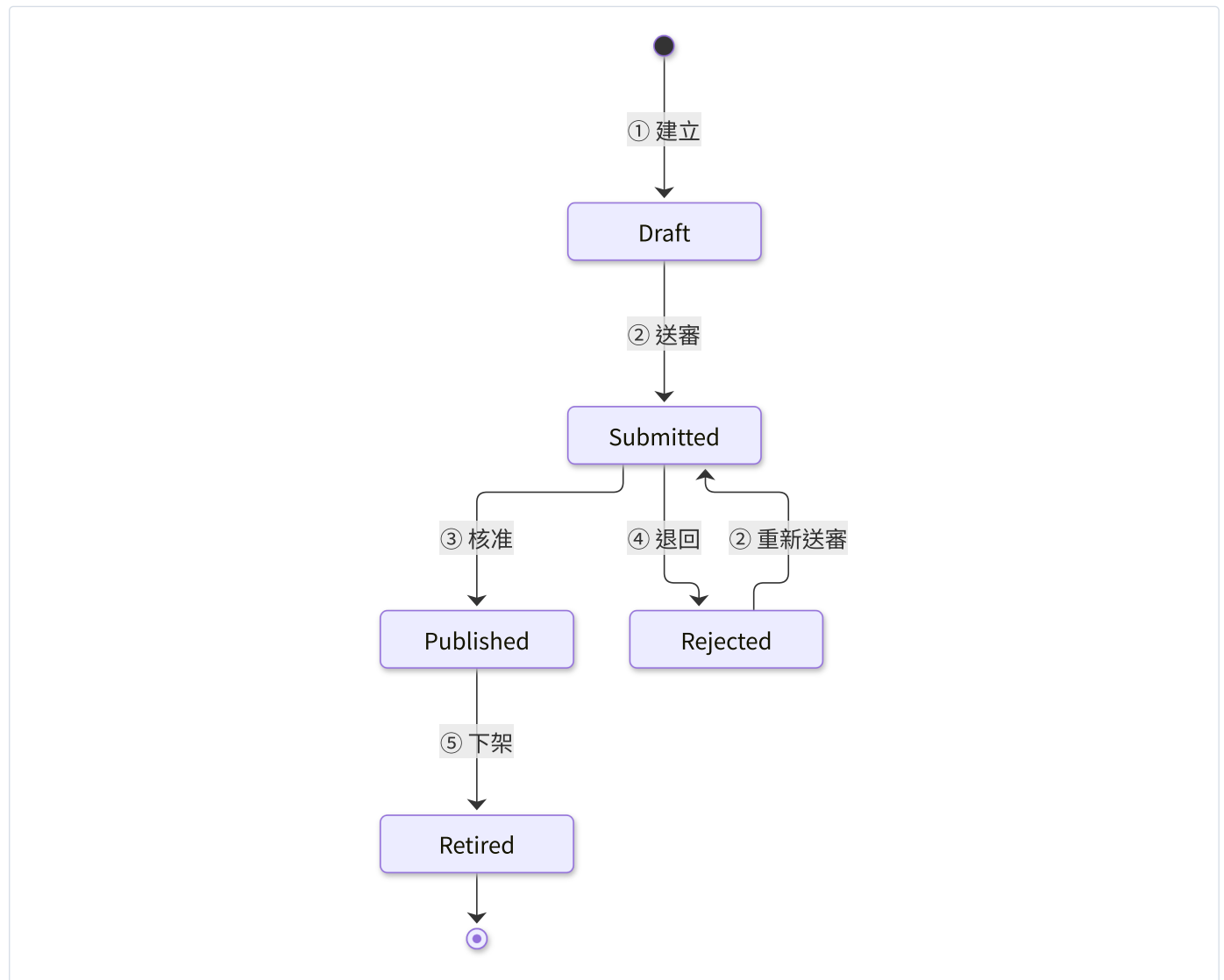


圖 06-1 API 版本的狀態轉換

編號	轉換	誰觸發	方法	前提條件
①	建立 → Draft	提供者	ApiDraftService.CreateAsync 、 CreateNextVersionAsync	同一支 API 沒有其他草稿或審核中的版本
②	Draft / Rejected → Submitted	提供者本人	PublicationService.SubmitAsync	送審前檢查全部通過；同一版本沒有其他待審案件
③	Submitted → Published	管理者	PublicationService.ApproveAsync	案件 Pending ；審核者不是送審者也不是擁有者；內容與快照一致；rowversion 未變
④	Submitted → Rejected	管理者	PublicationService.RejectAsync	同 ③，另外 必須填理由
⑤	Published → Retired	管理者	PublicationService.RetireAsync	必須填原因；rowversion 未變

沒有畫出來的箭頭都不合法。例如 Published 不能回到 Draft（要改就建立新版本），Retired 不能重新上架（要重新上架就送審新版本）。

不可變快照

送審那一刻，SubmitAsync 把版本的名稱、分類、說明、版本號、Method、URL、認證方式、兩個 Schema、請求範例序列化成一段 JSON，存進 ApiPublicationRequests.SnapshotJson。這一系列之後**永不修改**。

Section 05 已經讓 Submitted 版本不能從網站修改，為什麼還要快照？

1. **審核者看的內容要能事後證明**：半年後有人問「管理者當初核准的是什麼」，答案是快照，而不是「現在資料表裡的值」。
2. **防止繞過網站修改**：有人直接用 SQL 改了 ApiVersions.EndpointUrl，核准時 ApproveAsync 比對版本內容與快照不一致，就拒絕核准。
3. **每次送審各自一份**：被退回後修改再送審，會產生第二件案件、第二份快照；兩次的差異一目了然。

送審者與審核者分離

大綱規定管理者「不得核准本人送出的案件」。管理者也是會員，也能在會員中心登錄 API；如果他能審核自己的 API，審核就只剩形式。DecideAsync 同時檢查審核者不是**送審者**、也不是 API 的**擁有者**。

併發：兩位管理者同時按核准

下圖是兩位管理者（A、B）幾乎同時打開同一件案件、同時按下「核准」。注意：**兩人讀到的 rowversion 相同**，但資料庫只會讓第一個更新成功。

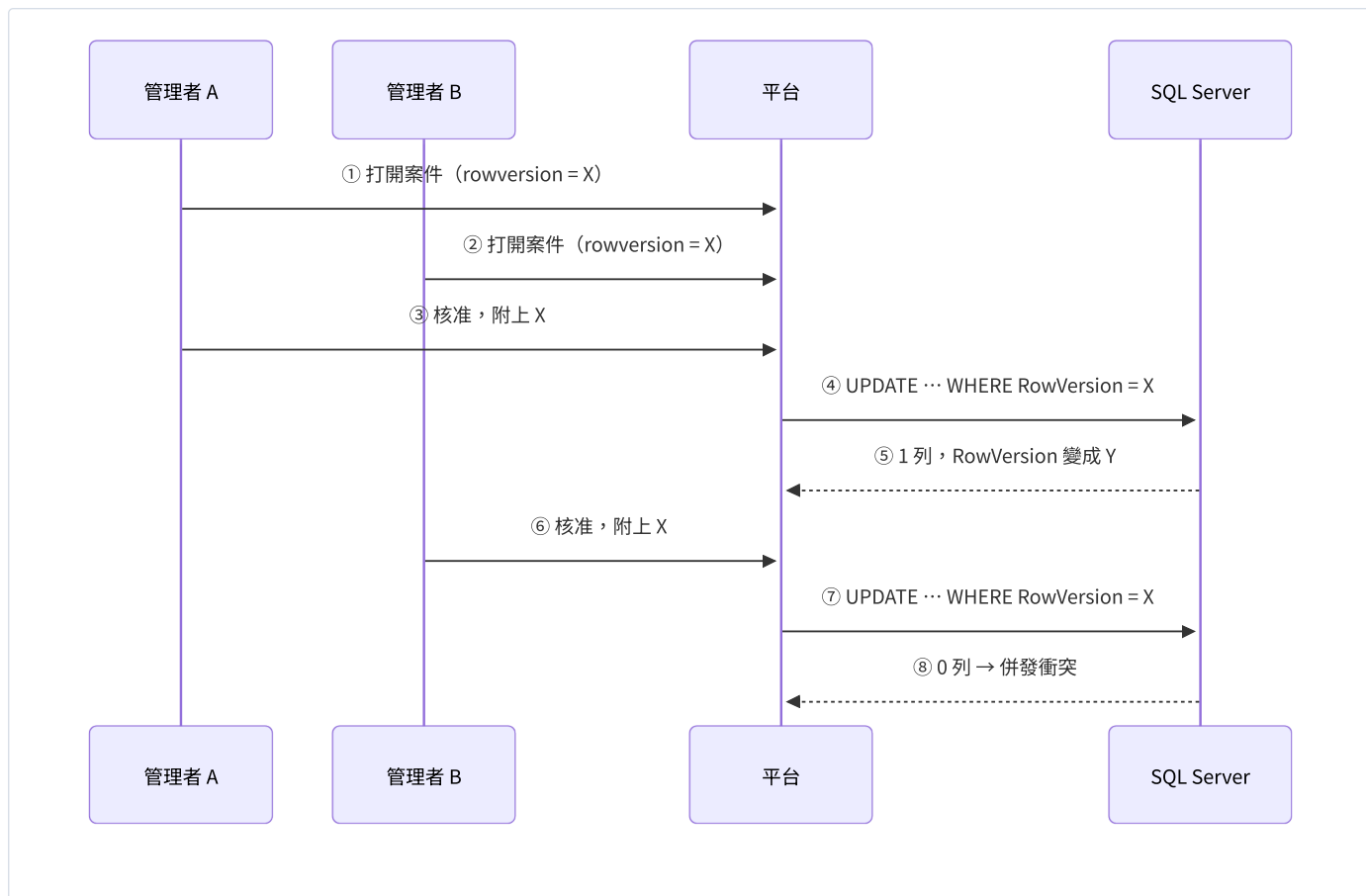


圖 06-2 兩位管理者同時核准同一件案件

訊息	內容	說明
①②	審核頁把版本目前的 rowversion 以 Base64 放在表單隱藏欄位 <code>rowVersion</code>	這是「我做決定時看到的版本」
③⑥	兩人送出的 <code>rowVersion</code> 都是 X	表單送出時帶回伺服器
④⑦	<code>UseClientRowVersion</code> 把 X 設為 EF Core 的「原始值」，EF Core 產生 <code>UPDATE ApiVersions ... WHERE Id = @id AND RowVersion = @X</code>	比對的是 使用者看到的版本 ，不是伺服器剛讀到的版本
⑤	A 的更新成功，SQL Server 自動把 rowversion 改成 Y；同一交易內寫入審核紀錄與稽核紀錄	A 看到「已核准並發布」
⑧	B 的 <code>WHERE RowVersion = X</code> 找不到列，EF Core 丟出 <code>DbUpdateConcurrencyException</code>	B 看到「這件送審案剛剛已被其他人處理，請重新整理後再確認。」， 整個交易回復 ，不會多一筆審核紀錄

rowversion 是第一道防線。第二道在資料庫：`ApprovalRecords.PublicationRequestId` 有一個篩選唯一索引 `UX_ApprovalRecords_PublicationRequestId`（只對不是 NULL 的值生效），一件上架申請最多只能有一筆審核紀錄。即使程式哪天少了 rowversion 檢查，第二筆審核紀錄也寫不進去。

交易

核准一件案件要改 4 個地方：`ApiPublicationRequests.Status`、`ApiVersions.Status` 與 `PublishedAtUtc`、新增 `ApprovalRecords`、新增 `AuditLogs`。EF Core 的一次 `SaveChangesAsync` 會把所有變更包在同一個資料庫交易裡——任何一步失敗（例如 rowversion 衝突），全部回復，不會出現「案件寫了核准但版本沒發布」的半套狀態。所以 `DecideAsync` 先在記憶體裡把所有變更準備好，最後只呼叫一次存檔。

限制與注意事項

- 本單元的「下架」只改版本狀態並寫稽核紀錄。下架後**新的呼叫被拒絕**要到 Section 11 的呼叫前檢查才生效；已核准的 Grant 如何處理在 Section 12 說明。
- 核准備註選填（寫成「(無備註)」），退回理由與下架原因必填。
- 審核歷程的時間都是伺服器的 UTC，畫面轉成臺灣時間顯示。

名詞對照

名詞	白話說明	在本課程中是
狀態轉換	資料從一個狀態合法地變成另一個狀態	圖 06-1 的 5 條箭頭
快照	某一刻的完整內容，之後不再修改	<code>ApiPublicationRequests.SnapshotJson</code>
樂觀併發	不先鎖資料，存檔時才檢查有沒有被別人改過	<code>rowversion</code> + <code>DbUpdateConcurrencyException</code>
篩選唯一索引	只對符合條件的列檢查唯一性	<code>UX_ApprovalRecords_PublicationRequestId</code>
交易	多個變更要嘛全部成功、要嘛全部不算	一次 <code>SaveChangesAsync</code>

教師示範（25 分鐘）

示範情境：王小明送審「郵遞區號查詢 v1」，管理者退回、王小明修改再送審、管理者核准；接著兩個瀏覽器同時核准同一件案件，觀察誰成功；最後看提供者與管理者兩邊的審核歷程。

示範步驟

1. 套用本單元 Migration：`dotnet ef database update --project src/McpPlatform.Infrastructure --startup-project src/McpPlatform.Web`，輸出含 `Applying migration '20260926112151_AddPublicationReview'.`。
2. 以王小明登入，打開「郵遞區號查詢 v1」（Section 05 已補齊，送審前檢查全部通過），按「送審」，出現「已送審 郵遞區號查詢 v1，等待管理者審核。」。此時未登入打開 `/Catalog/Details/11` 得到 `404`。
3. 以管理者登入，側欄「上架審核」。

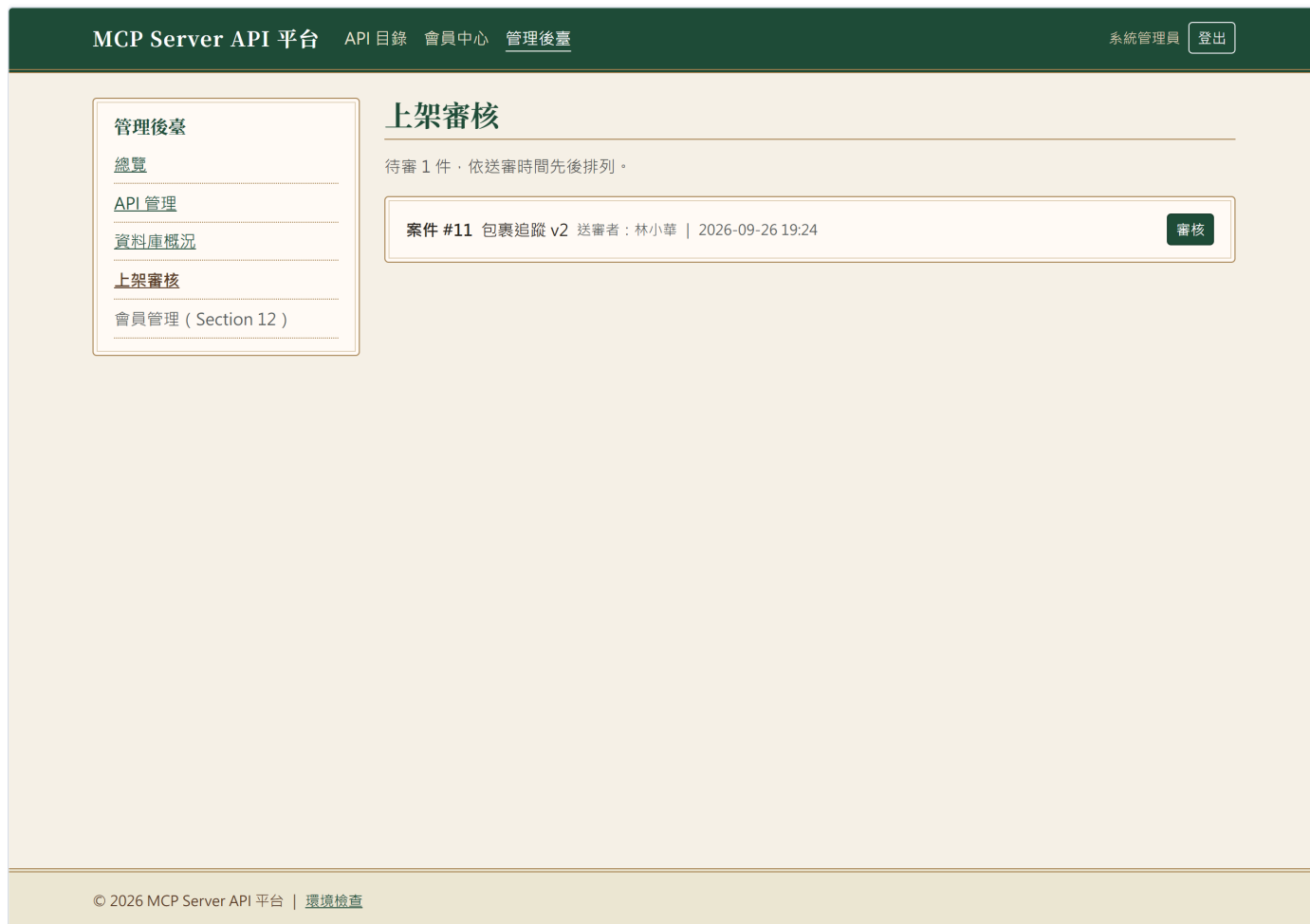


圖 06-3 後臺待審清單

畫面重點：

1. ①「待審 1 件，依送審時間先後排列」：`ListPendingAsync` 只列 `Status = Pending` 的案件，`ORDER BY RequestedAtUtc`。
2. ②「每一列的案件編號、API 與版本、送審者、送審時間」：案件編號是 `ApiPublicationRequests.Id`，同一版本每送審一次就有一個新編號。
3. ③「右側『審核』按鈕」：進入圖 06-4 的審核頁。
4. 按「審核」進入案件。



圖 06-4 審核頁：送審快照、審核決定、審核歷程

畫面重點：

1. ①左欄「送審快照」：直接顯示 `SnapshotJson`，是送審那一刻的內容，不是目前資料表的值。
2. ②右上「核准備註（選填）」與「核准發布」：表單裡有看不見的 `rowVersion` 隱藏欄位。
3. ③「退回理由（必填）」與「退回修訂」：理由空白送出會得到「退回必須填寫理由，提供者才知道要改什麼。」。
4. ④右下「審核歷程」：目前只有一筆「送審（林小華）」。
5. 對郵遞區號查詢的案件填退回理由「輸出 Schema 請補上 district（行政區）欄位。」按退回；王小明修改輸出 Schema 後再送審，產生第二件案件。
6. 用兩個瀏覽器（或一般視窗加無痕視窗）以管理者登入，都打開第二件案件，**同時**按「核准發布」。本單元實測結果：

管理者 A：302 → /Admin/Reviews 提示「已核准並發布 郵遞區號查詢 v1。」
管理者 B：302 → /Admin/Reviews/Details/9 提示「這件送審案剛剛已被其他人處理，請重新整理後再確認。」
資料庫：案件 #9 的審核紀錄 1 筆；版本狀態 Published

7. 王小明打開郵遞區號查詢 v1 的詳細頁。

會員中心

[總覽](#)[我的 API 與版本](#)[新增 API](#)[我的資料](#)[使用申請 \(Section 07 \)](#)[呼叫紀錄 \(Section 13 \)](#)

郵遞區號查詢 v1

✓ 已發布

送審前檢查

✓ 通過 用途說明至少 10 字

目前 21 字

✓ 通過 端點使用 HTTPS (本機

localhost 除外)

https://zip.example.test

✓ 通過 輸入 Schema 的最上層是

"type": "object"

符合

✓ 通過 已描述輸出 Schema

type = object

✓ 通過 請求範例符合輸入 Schema

通過

✓ 通過 API Key 憑證已設定

標頭 X-API-Key

審核歷程

2026-09-26 19:23 送審 (王小明)

案件 #8

2026-09-26 19:23 退回 (系統管理員)

案件 #8 : 輸出 Schema 請補上 district (行政區) 欄位。

2026-09-26 19:23 送審 (王小明)

案件 #9

2026-09-26 19:23 核准發布 (系統管理員)

案件 #9 : 規格完整

規格

分類	生活
說明	輸入地址，回傳 3+3 郵遞區號與行政區。
端點	GET https://zip.example.test/api/zip
認證	API Key 標頭 (已設定 X-API-Key)
輸入	<pre>{"type":"object","required":["address"],"properties":{"address":{"type":"string","minLength":5}}}</pre>
範例	<pre>{"address":"臺北市中正區重慶南路一段122號"}</pre>

圖 06-5 提供者頁面上的審核歷程

畫面重點：

- ① 右上「✓ 已發布」：狀態已是 **Published**；左欄送審前檢查下方不再有「修改」「送審」按鈕，因為已發布版本不能修改。
- ② 「審核歷程」四筆：送審（案件 #8）→ 退回（系統管理員，理由原文）→ 送審（案件 #9）→ 核准發布（理由「規格完整」）。退回的理由提供者看得到，才知道要改什麼。

程式碼走讀：審核的檢查順序

`DecideAsync` 是核准與退回共用的方法，在 `McpPlatform.Core/Publishing/PublicationService.cs`，由 `ReviewsController.Approve`、`Reject` 呼叫。前半段全是檢查，任何一項不通過都直接回傳，資料庫一個字都沒動：

```
// 範例/06_API上架送審與管理者審核/McpPlatform/src/McpPlatform.Core/Publishing/PublicationService.cs (DecideAsync 前半：檢查)
private async Task<PublicationResult> DecideAsync(string reviewerId, int requestId, ReviewDecision decision,
    string? reason, byte[] rowVersion, CancellationToken ct)
{
    var request = await repository.FindRequestAsync(requestId, ct);
    if (request is null) return PublicationResult.Missing();
    var version = request.ApiVersion;

    if (request.Status != ReviewStatus.Pending || version.Status != ApiVersionStatus.Submitted) // ① 已處理過的案件
        return PublicationResult.Refused($"這件送審案已是「{request.Status}」，不能重複審核。");
    if (reviewerId == request.RequestedBy || reviewerId == version.ApiDefinition.OwnerId) // ② 送審者與審核者分離
        return PublicationResult.Refused("不得審核本人送出或擁有的 API。");
    if (decision == ReviewDecision.Rejected && string.IsNullOrEmpty(reason))
        return PublicationResult.Refused("退回必須填寫理由，提供者才知道要改什麼。");
    if (!JsonNode.DeepEquals(JsonNode.Parse(SnapshotOf(version)), JsonNode.Parse(request.SnapshotJson))) // ③ 內容必須與送審快照相同
        return PublicationResult.Refused("版本內容與送審快照不一致，可能被繞過系統修改；請退回後重新送審。");
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	案件必須還是 <code>Pending</code> 、版本必須還是 <code>Submitted</code>	「事後再按一次核准」在這裡被擋下：實測提示「這件送審案已是「Approved」，不能重複審核。」
②	審核者 Id 不能等於送審者或擁有者	管理者自己登錄並送審的 API，自己按核准得到「不得審核本人送出或擁有的 API。」（本單元實測）
—	退回必須有理由	理由會出現在提供者的歷程裡（圖 06-5）
③	把目前版本重新序列化，與快照做結構比對（ <code>JsonNode.DeepEquals</code> ）	不用字串比對：Migration 以 SQL <code>FOR JSON</code> 回填的舊快照，與 C# 產生的 JSON 跳脫方式不同（例如 <code>/</code> 可能寫成 <code>\</code> ），但內容相同

後半段才開始改資料，最後一次存檔：

```
// 範例/06_API上架送審與管理者審核/McpPlatform/src/McpPlatform.Core/Publishing/PublicationService.cs (DecideAsync 後半：寫入)
repository.UseClientRowVersion(version, rowVersion); // ㉔ 以管理者畫面上的版本為比對基準
var now = clock.UtcNow();
var approved = decision == ReviewDecision.Approved;
request.Status = approved ? ReviewStatus.Approved : ReviewStatus.Rejected;
version.Status = approved ? ApiVersionStatus.Published : ApiVersionStatus.Rejected;
version.PublishedAtUtc = approved ? now : null;
version.UpdatedAtUtc = now;
repository.Add(new ApprovalRecord
{
    PublicationRequestId = request.Id, ReviewerId = reviewerId, Decision = decision,
    Reason = string.IsNullOrEmpty(reason) ? "（無備註）" : reason.Trim(), DecidedAtUtc = now
});
repository.Add(Audit(reviewerId, approved ? "ApprovePublication" : "RejectPublication", version.Id,
    $"{(approved ? "核准發布" : "退回")} {version.ApiDefinition.Name} {version.Version}", now));

try { await repository.SaveChangesAsync(ct); } // ㉔ 一次存檔 = 一個交易
catch (ConcurrencyConflictException) { return PublicationResult.Concurrent("這件送審案剛剛已被其他人處理，請重新整理後再確認。"); }
return PublicationResult.Ok(approved
    ? $"已核准並發布 {version.ApiDefinition.Name} {version.Version}。"
    : $"已退回 {version.ApiDefinition.Name} {version.Version}，提供者修改後可重新送審。");
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
④	把表單帶回來的 rowversion 設成比對基準	見圖 06-2；少了這行，比對的會是「伺服器剛剛讀到的值」，兩個並行請求都讀到 X 也都會成功
—	案件、版本、審核紀錄、稽核紀錄四項變更都只改記憶體	還沒碰資料庫
⑤	一次 <code>SaveChangesAsync</code> ；併發衝突轉成使用者看得懂的訊息	衝突時四項變更一起回復；Service 只認得 Core 的 <code>ConcurrencyConflictException</code> ，不認得 EF Core

學員實作（60 分鐘）

範例專案：範例/06_API上架送審與管理者審核/McpPlatform/（從 Section 05 複製後往上加）。本單元新增 Core 的 Publishing/（PublicationContracts.cs、PublicationService.cs）、Infrastructure 的 Publishing/EfPublicationRepository.cs 與 Migration ...AddPublicationReview、Web 的 Areas/Admin/Controllers/ReviewsController.cs、Security/RowVersionField.cs、Views/Shared/_ReviewHistory.cshtml、後臺審核與下架頁；修改會員中心詳細頁（送審按鈕與歷程）、DemoDataSeeder（送審快照）。

步驟一 快照欄位、唯一索引與資料回填（15 分鐘）

目的：ApiPublicationRequests 多一個 SnapshotJson；加兩個篩選唯一索引；既有案件補上快照。

在實體 ApiPublicationRequest 加 public string SnapshotJson { get; set; } = "{}";，並在設定檔加入：

```
// 範例/06_API上架送審與管理者審核/McpPlatform/src/McpPlatform.Infrastructure/Data/Configurations/ReviewConfigurations.cs（新增的兩個索引）
e.HasIndex(x => x.ApiVersionId).IsUnique()
  .HasFilter("[Status] = 'Pending'") // Section 06：同一版本同時只能有一件待審案
  .HasDatabaseName("UX_ApiPublicationRequests_OnePendingPerVersion");
```

```
// 範例/06_API上架送審與管理者審核/McpPlatform/src/McpPlatform.Infrastructure/Data/Configurations/ReviewConfigurations.cs（ApprovalRecord 部分）
e.HasIndex(x => x.PublicationRequestId).IsUnique()
  .HasFilter("[PublicationRequestId] IS NOT NULL") // Section 06：一件上架申請只能有一筆審核紀錄
  .HasDatabaseName("UX_ApprovalRecords_PublicationRequestId");
```

索引	篩選條件	擋下什麼
UX_ApiPublicationRequests_OnePendingPerVersion	Status = 'Pending'	同一版本連按兩次送審產生兩件待審案；已結案的案件不受限制，所以可以退回後再送審
UX_ApprovalRecords_PublicationRequestId	PublicationRequestId IS NOT NULL	同一件案件出現第二筆審核紀錄。篩選是必要的：使用申請的審核紀錄這欄是 NULL，沒有篩選的話多筆 NULL 會互相衝突

同樣的方式也為 AccessRequestId 建了 UX_ApprovalRecords_AccessRequestId，Section 07 會用到。產生 Migration：dotnet ef migrations add AddPublicationReview ...。產生的 SnapshotJson 預設值是 ""，改成 "{}"，並在 AddColumn 之後手動加入回填：

```
// 範例/06_API上架送審與管理者審核/McpPlatform/src/McpPlatform.Infrastructure/Data/Migrations/20260926112151_AddPublicationReview.cs（Up 方法中手動加入的回填）
// 手動加入：既有的送審案件補上快照，內容取自目前的版本（這些版本送審後就凍結，所以與送審當下相同）。
migrationBuilder.Sql("""
    UPDATE r SET SnapshotJson =
        (SELECT d.Name, d.Category, d.Description, v.Version, v.HttpMethod, v.EndpointUrl,
            v.AuthType, v.InputSchema, v.OutputSchema, v.RequestExample
         FOR JSON PATH, WITHOUT_ARRAY_WRAPPER)
    FROM dbo.ApiPublicationRequests AS r
    JOIN dbo.ApiVersions AS v ON v.Id = r.ApiVersionId
    JOIN dbo.ApiDefinitions AS d ON d.Id = v.ApiDefinitionId;
""");
```

程式說明：

項目	說明
migrationBuilder.Sql(...)	Migration 不只能改結構，也能搬資料；這段在全新資料庫上影響 0 列，在已有資料的資料庫上補齊快照
FOR JSON PATH, WITHOUT_ARRAY_WRAPPER	SQL Server 把一列轉成一個 JSON 物件（不包在陣列裡），欄位名稱與 C# 的 SnapshotOf 相同
為什麼可以用「目前的版本」	已送審的版本從送審那一刻起就不能從網站修改（Section 05），目前內容等於送審當時

預期結果：dotnet ef database update 輸出 Applying migration '20260926112151_AddPublicationReview'.；Section 03 示範資料中待審的「天氣預報查詢 v2」案件得到快照，並且可以被核准（本單元實測：「已核准並發布 天氣預報查詢 v2。」——證明 SQL 回填的快照通過了結構比對）。

步驟二 送審（15 分鐘）

目的：提供者在送審前檢查全部通過後，把版本送進待審清單。

```
// 範例/06_API上架送審與管理者審核/McpPlatform/src/McpPlatform.Core/Publishing/PublicationService.cs (SubmitAsync)
public async Task<PublicationResult> SubmitAsync(string ownerId, int versionId, CancellationToken ct)
{
    var version = await drafts.FindOwnedVersionAsync(ownerId, versionId, ct);
    if (version is null) return PublicationResult.Missing();
    if (version.Status is not (ApiVersionStatus.Draft or ApiVersionStatus.Rejected))
        return PublicationResult.Refused($"{version.Version} 目前是「{version.Status}」，不能送審。");

    var readiness = await draftService.CheckReadinessAsync(ownerId, versionId, ct);
    if (!ApiDraftService.IsReady(readiness!)) // ① 與 Section 05 同一份檢查
        return PublicationResult.Refused("送審前檢查尚未全部通過，不能送審。");

    var now = clock.UtcNow();
    version.Status = ApiVersionStatus.Submitted; // ② 從這一刻起內容凍結
    version.UpdatedAtUtc = now;
    repository.Add(new ApiPublicationRequest
    {
        ApiVersionId = version.Id, RequestedById = ownerId, RequestedAtUtc = now,
        Status = ReviewStatus.Pending, SnapshotJson = SnapshotOf(version) // ③ 存下送審當下的完整規格
    });
    repository.Add(Audit(ownerId, "SubmitPublication", version.Id, $"送審 {version.ApiDefinition.Name} {version.Version}", now));

    try { await repository.SaveChangesAsync(ct); }
    catch (ConcurrencyConflictException) { return PublicationResult.Concurrent("此版本剛剛已被送審，請重新整理。"); }
    return PublicationResult.Ok($"已送審 {version.ApiDefinition.Name} {version.Version}，等待管理者審核。");
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
—	<code>FindOwnedVersionAsync(ownerId, ...)</code>	只能送審自己的版本（Section 05 的擁有者條件）
①	重跑一次送審前檢查	頁面上的「送審」按鈕只在全部通過時出現，但有人可以直接 POST；伺服器端一定要再檢查一次
②	版本改成 <code>Submitted</code>	Section 05 的 <code>UpdateAsync</code> 從此拒絕修改
③	新增一件 <code>Pending</code> 案件並存快照	若同一版本已有待審案， <code>UX_ApiPublicationRequests_OnePendingPerVersion</code> 讓存檔失敗，轉成「此版本剛剛已被送審」

會員中心 `ApisController.Submit` 只有四行：呼叫 `SubmitAsync(User.GetUserId(), id, ct)`、`NotFound` 時回 404、把訊息放 `TempData`、轉回詳細頁。

預期結果：送審後詳細頁提示「已送審 郵遞區號查詢 v1，等待管理者審核。」，狀態標籤變成「🔒 審核中」，後臺待審清單多一件。

步驟三 併發衝突的資料層實作（15 分鐘）

目的：讓 rowversion 比對「使用者看到的版本」，並把 EF Core 的例外轉成 Core 的例外。

```
// 範例/06_API上架送審與管理者審核/McpPlatform/src/McpPlatform.Infrastructure/Publishing/EfPublicationRepository.cs (兩個方法)
public void UseClientRowVersion(ApiVersion version, byte[] rowVersionSeenByUser) =>
    db.Entry(version).Property(v => v.RowVersion).OriginalValue = rowVersionSeenByUser; // ① UPDATE ... WHERE RowVersion = 這個值

public async Task SaveChangesAsync(CancellationToken ct)
{
    try
    {
        await db.SaveChangesAsync(ct); // ② 多筆變更包在同一個交易
    }
    catch (DbUpdateConcurrencyException ex) // ③ rowversion 對不上：更新了 0 列
    {
        throw new ConcurrencyConflictException("資料已被其他人修改。", ex);
    }
    catch (DbUpdateException ex) when (ex.InnerException is SqlException { Number: 2601 or 2627 }) // ④ 違反唯一索引
    {
        throw new ConcurrencyConflictException("違反唯一性限制，同一件事已經做過了。", ex);
    }
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	設定 <code>RowVersion</code> 屬性的原始值	EF Core 產生 UPDATE 時， <code>WHERE</code> 條件用的是原始值；預設原始值是剛從資料庫讀到的，要換成使用者表單帶回來的
②	一次存檔內的所有 INSERT、UPDATE 在同一個交易	圖 06-2 的 ⑤：成功就全部生效
③	<code>DbUpdateConcurrencyException</code> ：預期更新 1 列，實際 0 列	圖 06-2 的 ⑧
④	SQL Server 錯誤 2601（唯一索引）、2627（唯一約束）	例如同一本兩件待審案、同一案件兩筆審核紀錄

rowversion 在表單以 Base64 往返。`Security/RowVersionField.TryDecode` 用 `Convert.TryFromBase64String` 解碼並確認剛好 8 位元組，失敗時 Controller 回 400。

常見改錯：用 `Convert.FromBase64String(rowVersion)` 直接轉——隱藏欄位被竄改時丟 `FormatException`，使用者看到 500。本單元第一版就是這樣（見〈常見錯誤〉）。

預期結果：兩個瀏覽器同時核准，一個成功、一個看到衝突訊息；把隱藏欄位改成 `not-base64!!` 送出得到 400。

步驟四 審核頁、下架與歷程（15 分鐘）

目的：做出圖 06-3～06-5 的三個畫面，以及下架確認頁。

`ReviewsController` 的核准動作：

```
// 範例/06_API上架送審與管理者審核/McpPlatform/src/McpPlatform.Web/Areas/Admin/Controllers/ReviewsController.cs (核准與結果處理)
[HttpPost]
public async Task<IActionResult> Approve(int id, string? note, string? rowVersion, CancellationToken ct) =>
    RowVersionField.TryDecode(rowVersion, out var rv)
        ? Done(id, await publication.ApproveAsync(User.GetUserId(), id, note, rv, ct)) // ① 審核者 = 登入者
        : BadRequest();

// 範例/06_API上架送審與管理者審核/McpPlatform/src/McpPlatform.Web/Areas/Admin/Controllers/ReviewsController.cs (Done)
private IActionResult Done(int id, PublicationResult result)
{
    if (result.NotFound) return NotFound();
    TempData["Notice"] = result.Message; // ② 成功、拒絕、衝突都告訴管理者發生什麼事
    return result.Succeeded
        ? RedirectToAction(nameof(Index))
        : RedirectToAction(nameof(Details), new { id });
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	審核者 Id 取自登入身分	表單裡沒有「審核者」欄位，無法冒充別的管理者
②	成功回待審清單；被拒絕或衝突回同一件案件頁，顯示原因	衝突時管理者回到案件頁、重新整理就會看到最新狀態

下架在 `Admin/Apis/Retire/{id}`：GET 顯示確認頁（狀態、歷程、原因欄位與 rowversion），POST 呼叫 `RetireAsync`。審核歷程 `_ReviewHistory` 部分檢視由 `GetHistoryAsync` 合併三個來源——送審（`ApiPublicationRequests`）、核准與退回（`ApprovalRecords`）、下架（`AuditLogs` 中 `Action = 'RetireVersion'`）——依時間排序，會員中心與後臺共用。

預期結果（本單元實測）：天氣預報查詢 v1 下架時原因空白 → 「下架必須填寫原因。」；填原因後 → 「已下架 天氣預報查詢 v1。」，`/Catalog/Details/4` 變成 404。

檢查點

做到這裡，你應該可以：

- ☐ 送審前檢查沒通過的版本看不到「送審」按鈕；直接 POST `/Member/Apis/Submit/{id}` 也被拒絕
- ☐ 審核中的版本在公開目錄與 `/Catalog/Details/{id}` 都查不到；核准後才出現
- ☐ 退回時理由空白會被擋下；提供者在自己的詳細頁看得到退回理由
- ☐ 同一件案件兩次核准只有一次生效；資料庫 `ApprovalRecords` 該案件只有 1 筆

常見錯誤

以下都是撰寫本單元時實際遇到的原文。

症狀（錯誤訊息原文）	原因	處理
按核准得到 500，日誌 <code>System.FormatException: The input is not a valid Base-64 string as it contains a non-base 64 character...</code>	隱藏欄位的 rowversion 被改壞（或用程式送出時拿到的是 HTML 編碼後的 <code>&#x2B;</code> 而不是 <code>+</code> ）	用 <code>RowVersionField.TryDecode</code> ，失敗回 400；用程式測試時先 <code>HtmlDecode</code> 隱藏欄位的值
審核真的快照滿滿都是 "	<code>JavaScriptEncoder.Create(UnicodeRanges.All)</code> 仍會跳脫引號	快照與顯示都改用 <code>JavaScriptEncoder.UnsafeRelaxedJsonEscaping</code> ；Razor 輸出時仍會做 HTML 編碼，沒有 XSS 風險
訊息 2601 ... 無法以唯一索引 'UX_ApprovalRecords_PublicationRequestId' 在物件 'dbo.ApprovalRecords' 中插入重複的索引鍵資料列。重複的索引鍵值是 (9)。	直接用 SQL 替已審核的案件再寫一筆審核紀錄	這是資料庫層的保護在運作；審核一律經由網站

單元驗收（15 分鐘）

驗收標準：未核准版本不可使用；同案重複核准不產生重複發布；退回修訂有完整歷程。

驗收方式：

1. **未核准版本不可使用：**送審後、核准前，未登入打開 `/Catalog/Details/{版本 Id}` 得到 `404`，公開目錄搜尋不到；核准後同一網址得到 `200`。本單元實測郵遞區號查詢 v1：審核中 `404`、核准後 `200`。
2. **同案重複核准不產生重複發布：**用兩個工作階段並行核准同一件案件，一個成功、一個得到「這件送審案剛剛已被其他人處理」；事後再按一次得到「這件送審案已是「Approved」，不能重複審核。」。在 SSMS 執行 `SELECT COUNT(*) FROM ApprovalRecords WHERE PublicationRequestId = 9`，結果是 `1`。
3. **退回修訂有完整歷程：**郵遞區號查詢 v1 的歷程依序是 送審（#8）→ 退回（理由原文）→ 送審（#9）→ 核准發布（圖 06-5）。對應的資料：
`ApiPublicationRequests` 兩列（#8 `Rejected`、#9 `Approved`）、`ApprovalRecords` 兩列。
4. **反向驗證：**管理者自己在會員中心登錄並送審一支 API，再到後臺核准，必須得到「不得審核本人送出或擁有的 API。」，版本維持「審核中」。

回顧：狀態轉換只能經由 Service；送審存快照、核准比對快照；rowversion 擋住同時核准，篩選唯一索引擋住重複紀錄；一次存檔就是一個交易。下一單元（Section 07）處理另一條流程——會員申請使用已發布的 API，核准後建立 ApiGrant。

延伸閱讀

- [EF Core 處理併發衝突](#)——rowversion、原始值與 `DbUpdateConcurrencyException`。
- [EF Core 交易](#)——`SaveChanges` 預設交易與明確交易。
- [SQL Server 篩選索引](#)——篩選唯一索引的條件與限制。
- [SQL Server FOR JSON](#)——`PATH`、`WITHOUT_ARRAY_WRAPPER` 選項。

第 7 章

API 使用申請與授權審核

用途、申請版本、有效期間、額度、Scope；Approval 與 ApiGrant 分離；使用申請、個人進度、管理者核准與駁回、Grant 建立與撤銷



課程時數 2 小時（120 分鐘） 當堂成果 申請、審核、期限與額度設定 範例專案 範例/07_API使用申請與授權審核

學習目標

上完這個單元，你能夠：

- 讓會員對一個**已發布版本**提出使用申請，填寫用途、希望的有效期間、每日額度與權限範圍（Scope）。
- 說明「審核結果」（`ApiAccessRequests` + `ApprovalRecords`）與「使用權」（`ApiGrants`）為什麼分開存，並在「我的申請與授權」頁把兩者並排顯示。
- 在同一個交易內完成「申請標為核准 + 寫審核紀錄 + 建立 Grant」，並讓管理者核准比申請更短的期間或更少的額度。
- 用篩選唯一索引擋住重複送出，用 rowversion 與唯一索引擋住重複核准，並實測兩個並行請求只產生一筆 Grant。
- 撤銷 Grant 時只改授權狀態，原本的申請與核准紀錄完全不變。

先備知識

- 完成 Section 06：有已發布的版本可以申請（商品查詢 v1、課程查詢 v1、服務工單建立 v1、郵遞區號查詢 v1、天氣預報查詢 v2）。
- Section 06 的 rowversion 併發控制、`ConcurrencyConflictException`、`RowVersionField`；本單元照同樣的模式做一次。
- Section 03 的 `IX_ApiGrants_SourceAccessRequestId`（一件申請最多一筆 Grant）與 Section 06 預先建立的 `UX_ApprovalRecords_AccessRequestId`。
- 本單元有一個新 Migration。

觀念說明（20 分鐘）

為什麼註冊之後還要申請

Section 01 的驗收句：「註冊成功不代表取得 API 呼叫權」。本單元把它變成程式：會員要呼叫某個 API 版本，必須先提出申請，說明用途，由管理者判斷是否合理，核准後系統才建立一筆**使用權（ApiGrant）**。管理者看得到申請人的用途、期間、額度，可以核准、縮減或駁回。

申請與使用權都綁定**明確的版本**，不是綁 API 名稱：**v1** 核准了，提供者之後發布的 **v2** 仍要重新申請（大綱：新版本不能自動繼承舊版核准）。

申請的內容

欄位	合法值	為什麼需要
用途	10~500 字	管理者判斷是否核准的主要依據；撤銷時也常以「用途已結束」為由
希望的有效期間	1~365 天	使用權一定有到期日，沒有「永久有效」；到期要重新申請
希望的每日額度	1~10000 次	Section 11 以臺灣日期為單位計次，用完當天就拒絕
權限範圍（Scope）	目前只有 <code>tools:call</code>	使用權允許做什麼；Section 10 的 OAuth Access Token 也要帶這個 Scope 才能呼叫工具

管理者核准時可以把期間與額度**調低**，但**不能超過**申請的數字——會員沒要求的權限，不應該被給予。

審核結果與使用權分開存

下圖是一件使用申請的完整生命週期。看的時候注意：⑤ **撤銷或到期時，改變的只有 Grant；申請仍然是「已核准」**。

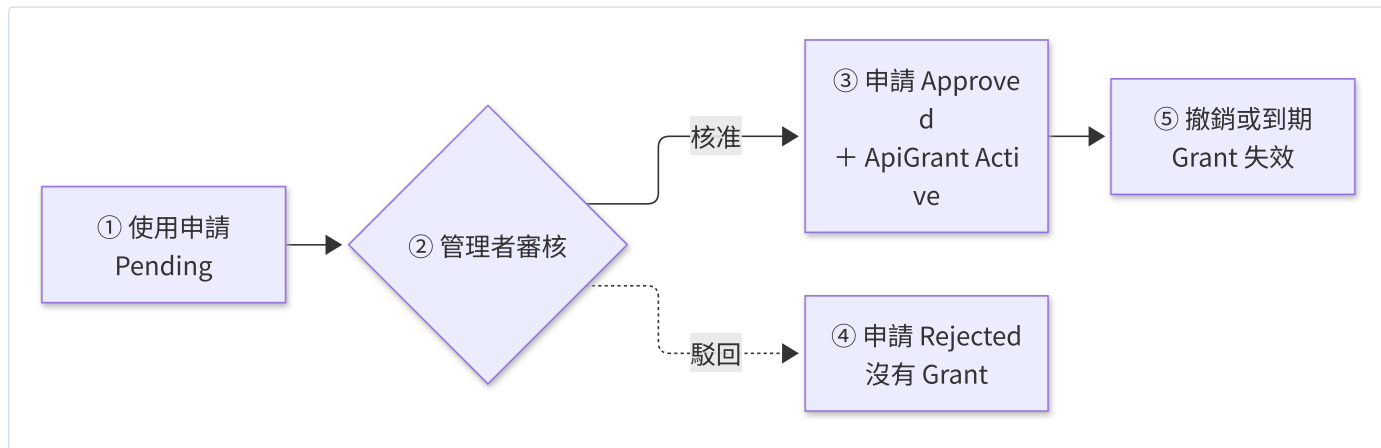


圖 07-1 使用申請與使用權的生命週期

編號	資料變化	誰觸發	對應方法
①	新增 <code>ApiAccessRequests</code> 一列， <code>Status = Pending</code>	會員	<code>AccessService.ApplyAsync</code>
②	管理者看用途、期間、額度	管理者	<code>AccessReviewsController.Details</code>
③	申請改 <code>Approved</code> 、新增 <code>ApprovalRecords</code> 、新增 <code>ApiGrants</code> （ <code>Active</code> 、到期日、額度）， 同一個交易	管理者	<code>AccessService.ApproveAsync</code>
④	申請改 <code>Rejected</code> 、新增 <code>ApprovalRecords</code> （含理由）， 不建立 Grant	管理者	<code>AccessService.RejectAsync</code>
⑤	撤銷：Grant 改 <code>Revoked</code> ，記下時間與原因；到期： <code>ExpiresAtUtc</code> 已過，Grant 自然失效	管理者／時間	<code>AccessService.RevokeAsync</code>

分開存的好處：

- **稽核**：半年後查「王小明當初為什麼能用？」，答案在 ③ 的審核紀錄與用途說明，就算 Grant 早已撤銷。
- **判斷「現在能不能用」只看一張表**：Section 11 每次呼叫都查 `ApiGrants` 有沒有 `Active` 且未到期的那一列，不必理會申請的歷史。
- **撤銷後重新申請**：Grant 撤銷後會員可以再申請，產生新的申請、新的 Grant；舊的那一組完整保留。

防止重複：送出與核准

情況	會發生什麼	防線
會員連接兩次「送出申請」	兩件一模一樣的待審申請	篩選唯一索引 <code>UX_ApiAccessRequests_OnePendingPerUserVersion</code> ：(UserId, ApiVersionId) 在 <code>Status = 'Pending'</code> 的列中唯一
會員已有有效 Grant 又申請	同一版本兩筆有效授權，額度加倍	<code>ApplyAsync</code> 先查 <code>HasActiveGrantAsync</code> ，有就拒絕
兩位管理者同時核准	兩筆 Grant	第一道：申請的 rowversion（同 Section 06）；第二道： <code>IX_ApiGrants_SourceAccessRequestId</code> 與 <code>UX_ApprovalRecords_AccessRequestId</code>
管理者事後再按一次核准	第二筆 Grant	<code>CheckReviewable</code> 檢查申請必須還是 <code>Pending</code>

篩選唯一索引擋的是「待審中」的重複；已核准或已駁回的申請不受限制，所以駁回後可以再申請、撤銷後也可以再申請。

限制與注意事項

- 撤銷與到期要到 **Section 11 才會真正擋住呼叫**：本單元只改資料；呼叫當下的檢查在 Section 11。
- 本單元不做「到期自動改狀態」的排程：Grant 的 `Status` 仍是 `Active`，但 `ExpiresAtUtc` 已過就視為無效；後臺以「🕒 已到期」顯示。判斷一律用「`Active` 且 未到期」。
- 管理者不因為有 `Admin` 角色就能呼叫 API（Section 01 的權限表）；管理者自己要用，也得申請，而且**不能審核自己的申請**。
- 資料庫的 IDENTITY 流水號在交易回復後不會回收，所以 Grant、申請的 Id 可能跳號（本單元實測出現 Grant #1、#4、#5）。這是正常現象，不要用 Id 是否連續判斷資料有沒有遺失。

名詞對照

名詞	白話說明	在本課程中是
使用申請	會員要求使用某個版本的正式請求	<code>ApiAccessRequests</code>
使用權（Grant）	核准後產生、有期限與額度的授權	<code>ApiGrants</code>
Scope	使用權允許做的事	<code>tools:call</code>
撤銷	管理者讓一筆有效授權提前失效	<code>GrantStatus.Revoked</code>
有效授權	<code>Status = Active</code> 且 <code>ExpiresAtUtc</code> 在未來	<code>HasActiveGrantAsync</code> 的條件

教師示範（25 分鐘）

示範情境：林小華對「天氣預報查詢 v2」提出申請；管理者審核王小明「商品查詢 v1」的重新申請，並縮減額度；最後在「我的申請與授權」與「Grant 管理」兩個畫面，看審核結果與使用權如何分開呈現。

示範步驟

- 1. 在 範例/07_API使用申請與授權審核/McpPlatform 執行 `dotnet ef database update ...`，輸出含 `Applying migration '20260926113344_AddAccessReview'`；再啟動網站。
- 2. 以林小華登入，開 `/Catalog/Details/5`（天氣預報查詢 v2），按「申請使用」。

MCP Server API 平台

API 目錄

會員中心

林小華 登出

會員中心

總覽

我的 API 與版本

新增 API

我的資料

我的申請與授權

呼叫紀錄 (Section 13)

申請使用 天氣預報查詢 v2 (提供者：王小明)

核准後才會建立使用授權 (Grant)。管理者可以核准比申請更短的期間或更少的額度。

用途說明

物流站點依天氣調整派車

希望的有效期間 (天)

希望的日常額度 (次)

權限範圍

30

100

tools:call (呼叫工具)

送出申請

© 2026 MCP Server API 平台 | 環境檢查

圖 07-2 使用申請表單

畫面重點：

- 1. ①標題旁「天氣預報查詢 v2 (提供者：王小明)」：申請綁定的是這個版本。
- 2. ②「用途說明」：至少 10 字，送出前 DataAnnotations 檢查一次，Service 再檢查一次。
- 3. ③期間預設 30 天、額度 100 次、權限範圍固定 `tools:call`：選單只有一個選項；有人竄改表單送出其他值，Service 回「目前只開放 tools:call 權限範圍」。
- 4. ④說明文字「管理者可以核准比申請更短的期間或更少的額度」：先讓會員知道審核可能縮減。
- 5. 以管理者登入，側欄「使用申請審核」→ 王小明的申請 #8。



圖 07-3 管理者審核一件使用申請

畫面重點：

1. ①**左欄申請內容**：用途、期間、額度、範圍，以及版本目前狀態「✓ 已發布」。版本若在申請後被下架，這裡會顯示「已下架」，核准會被拒絕。
2. ②**「核准天數」「核准每日額度」預設為申請的數字**：輸入框設了 `max`，但伺服器端仍會檢查。實測輸入 120 天（申請 90 天）得到「核准天數必須是 1～90（不可超過申請的天數）。」
3. ③**「核准並建立授權」**：按下後同一個交易內建立 Grant。
4. ④**「駁回理由（必填）」**：理由會出現在申請人的「我的申請與授權」。
5. 以王小明登入，會員中心 → 「我的申請與授權」。

會員中心

[總覽](#)

[我的 API 與版本](#)

[新增 API](#)

[我的資料](#)

[我的申請與授權](#)

[呼叫紀錄 \(Section 13 \)](#)

我的申請與授權

左邊是「申請當初的審核結果」，右邊是「現在能不能用」。兩者分開：授權被撤銷或到期後，核准紀錄仍在。

商品查詢 v1 申請 #8 | 2026-09-26 19:34 | 30 天、每日 100 次

申請結果

 審核中

目前授權

— 沒有授權

課程查詢 v1 申請 #2 | 2026-09-25 16:00 | 90 天、每日 500 次

申請結果

 已核准 核准 30 天、每日 200 次。

目前授權

 有效 Grant #4 · 每日 200 次 · 到 2026-10-26 19:35

商品查詢 v1 申請 #1 | 2026-09-21 16:00 | 30 天、每日 100 次

申請結果

 已核准 用途明確，額度合理。

目前授權

 已撤銷 Grant #1 · 每日 100 次 · 到 2026-10-21 17:00

圖 07-4 會員看自己的申請結果與目前授權

畫面重點：

- ①**每一筆分左右兩欄**：左欄「申請結果」，右欄「目前授權」。
- ②**商品查詢 v1 申請 #1**：左欄「 已核准」（理由「用途明確，額度合理。」），右欄「 已撤銷」—— 這正是圖 07-1 的 ⑤：申請結果沒變，授權失效了。
- ③**課程查詢 v1 申請 #2**：申請 90 天、每日 500 次，右欄 Grant #4 只有「每日 200 次，到 2026-10-26」—— 管理者縮減了額度與期間。
- ④**最上面的商品查詢 v1 申請 #8**：撤銷後重新申請，「 審核中」、「— 沒有授權」。
- 以管理者打開「Grant 管理」。

管理後臺

[總覽](#)[API 管理](#)[資料庫概況](#)[上架審核](#)[使用申請審核](#)[Grant 管理](#)[會員管理 \(Section 12 \)](#)

Grant 管理

共 3 筆使用授權。撤銷只改授權狀態，原本的申請與核准紀錄不變。

#	會員	API	有效期間	每日額度	狀態	撤銷
5 申請 #4	林小華	課程查詢 v1 tools:call	2026-09-26 19:35 ~ 2026-11-25 19:35	300	✓ 有效	原因 撤銷
4 申請 #2	王小明	課程查詢 v1 tools:call	2026-09-26 19:35 ~ 2026-10-26 19:35	200	✓ 有效	原因 撤銷
1 申請 #1	王小明	商品查詢 v1 tools:call	2026-09-21 17:00 ~ 2026-10-21 17:00	100	已撤銷 示範撤銷：用途已結束。	

圖 07-5 後臺 Grant 管理

畫面重點：

- ①「#」欄下方的「申請 #…」：每筆 Grant 都指回它的來源申請，一對一。
- ②有效期間：`ValidFromUtc` ~ `ExpiresAtUtc`，以臺灣時間顯示。
- ③狀態：「✓ 有效」或「已撤銷」（附原因）；`Active` 但已過期的會顯示「已到期」。
- ④撤銷欄：只有有效的 Grant 有「原因」輸入框與「撤銷」按鈕；原因空白送出會被拒絕。

程式碼走讀：核准就是建立 Grant

`ApproveAsync` 在 `McpPlatform.Core/Access/AccessService.cs`，由 `AccessReviewsController.Approve` 呼叫：

```
// 範例/07_API使用申請與授權審核/McpPlatform/src/McpPlatform.Core/Access/AccessService.cs (ApproveAsync)
public async Task<AccessResult> ApproveAsync(string reviewerId, int requestId, int days, int quota, string? note,
byte[] rowVersion, CancellationToken ct)
{
    var request = await repository.FindRequestAsync(requestId, ct);
    if (request is null) return AccessResult.Missing();
    var refusal = CheckReviewable(reviewerId, request, requirePublished: true);
    if (refusal is not null) return refusal;
    if (days < 1 || days > request.RequestedDays) return AccessResult.Refused($"核准天數必須是 1~{request.RequestedDays} (不可超過申請的天數)");
    if (quota < 1 || quota > request.RequestedQuota) return AccessResult.Refused($"核准額度必須是 1~{request.RequestedQuota} (不可超過申請的額度)");

    repository.UseClientRowVersion(request, rowVersion);
    var now = clock.GetUtcNow();
    request.Status = ReviewStatus.Approved;
    repository.Add(new ApprovalRecord
    {
        AccessRequestId = request.Id, ReviewerId = reviewerId, Decision = ReviewDecision.Approved,
        Reason = string.IsNullOrEmpty(note) ? $"核准 {days} 天、每日 {quota} 次。" : note.Trim(), DecidedAtUtc = now
    });
    repository.Add(new ApiGrant // ① 核准才建立 Grant
    {
        UserId = request.UserId, ApiVersionId = request.ApiVersionId, Scope = request.RequestedScope,
        ValidFromUtc = now, ExpiresAtUtc = now.AddDays(days), Quota = quota,
        Status = GrantStatus.Active, SourceAccessRequestId = request.Id // ② 指回來源申請 (唯一)
    });
    repository.Add(Audit(reviewerId, "ApproveAccess", "ApiAccessRequest", request.Id,
        $"核准 使用 {request.ApiVersion.ApiDefinition.Name} {request.ApiVersion.Version}，{days} 天、每日 {quota} 次", now));

    try { await repository.SaveChangesAsync(ct); } // ③ 申請、審核紀錄、Grant 一起成功或一起回復
    catch (ConcurrencyConflictException) { return AccessResult.Concurrent("這件申請剛剛已被其他人處理，請重新整理後再確認。"); }
    return AccessResult.Ok($"已核准，並建立 {days} 天、每日 {quota} 次的使用授權。");
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
—	<code>CheckReviewable(..., requirePublished: true)</code>	申請必須還是 <code>Pending</code> 、審核者不是申請人、版本仍是已發布 (見步驟三)
—	天數、額度不可超過申請的數字	只能縮減，不能加碼
①	在核准的同時新增 Grant：有效期從現在起算，到期日 = 現在 + 核准天數	Grant 只在這裡產生；程式其他地方都沒有 <code>new ApiGrant</code>
②	<code>SourceAccessRequestId = request.Id</code>	唯一索引保證一件申請最多一筆 Grant，也讓稽核能從 Grant 找回當初的申請與用途
③	一次存檔	rowversion 衝突或唯一索引衝突時，申請狀態、審核紀錄、Grant 三者一起回復

執行結果 (本單元實測，兩位管理者並行核准林小華的課程查詢申請 #4)：

管理者 A → /Admin/AccessReviews 「已核准，並建立 60 天、每日 300 次的使用授權。」
 管理者 B → /Admin/AccessReviews/Details/4 「這件申請剛剛已被其他人處理，請重新整理後再確認。」
 申請 #4 的 Grant 筆數：1；審核紀錄筆數：1

學員實作（60 分鐘）

範例專案：範例/07_API使用申請與授權審核/McpPlatform/（從 Section 06 複製後往上加）。本單元新增 Core 的 `Access/`（`AccessContracts.cs`、`AccessService.cs`）、Infrastructure 的 `Access/EfAccessRepository.cs` 與 Migration `....AddAccessReview`、Web 的會員中心 `AccessController` 與兩個頁面、後臺 `AccessReviewsController`、`GrantsController` 與三個頁面；公開詳細頁加上「申請使用」按鈕。

步驟一 申請的權限範圍、rowversion 與唯一索引（10 分鐘）

目的：`ApiAccessRequests` 多兩個欄位 `RequestedScope`、`RowVersion`，並加一個篩選唯一索引。

實體加 `public string RequestedScope { get; set; } = "tools:call"`；與 `public byte[] RowVersion { get; set; } = []`，設定檔改成：

```
// 範例/07_API使用申請與授權審核/McpPlatform/src/McpPlatform.Infrastructure/Data/Configurations/ReviewConfigurations.cs (ApiAccessRequest 部分)
e.Property(x => x.Purpose).HasMaxLength(500);
e.Property(x => x.RequestedScope).HasMaxLength(50);
e.Property(x => x.RowVersion).IsRowVersion();
e.HasOne<ApplicationUser>().WithMany().HasForeignKey(x => x.UserId);
e.HasIndex(x => new { x.UserId, x.ApiVersionId }).IsUnique()
    .HasFilter("[Status] = 'Pending'") // Section 07: 同一人對同一版本只能有一件待審申請
    .HasDatabaseName("UX_ApiAccessRequests_OnePendingPerUserVersion");
```

程式說明：

設定	效果	為什麼
<code>IsRowVersion()</code>	新增 <code>rowversion</code> 欄位，SQL Server 自動為既有 3 列填值	兩位管理者同時審核同一件申請時只有一位成功
篩選唯一索引	<code>(UserId, ApiVersionId)</code> 只在 <code>Pending</code> 列中唯一	擋住重複送出，但不妨礙駁回後、撤銷後再申請

產生 Migration `AddAccessReview`，把 `RequestedScope` 的 `defaultValue` 從 `""` 改成 `"tools:call"`（既有申請的權限範圍就是目前唯一開放的那一個）。

預期結果：`SELECT Id, RequestedScope, DATALENGTH(RowVersion) FROM ApiAccessRequests` 既有 3 列都是 `tools:call` 與 8。

步驟二 送出申請（15 分鐘）

目的：會員送出申請，Service 擋掉所有不該成立的情況。

```
// 範例/07_API使用申請與授權審核/McpPlatform/src/McpPlatform.Core/Access/AccessService.cs (ApplyAsync)
public async Task<AccessResult> ApplyAsync(string userId, AccessApplication form, CancellationToken ct)
{
    var version = await repository.FindVersionAsync(form.VersionId, ct);
    if (version is null) return AccessResult.Missing();
    if (version.Status != ApiVersionStatus.Published) // ㉑ 只能申請已發布的版本
        return AccessResult.Refused("只能申請已發布的版本。");
    if (form.Purpose.Trim().Length < 10) return AccessResult.Refused("請具體說明用途（至少 10 字），管理者依此判斷是否核准。");
    if (form.Days is < 1 or > MaxDays) return AccessResult.Refused($"有效期間必須是 1~{MaxDays} 天。");
    if (form.Quota is < 1 or > MaxQuota) return AccessResult.Refused($"每日額度必須是 1~{MaxQuota} 次。");
    if (form.Scope != DefaultScope) return AccessResult.Refused($"目前只開放 {DefaultScope} 權限範圍。");

    var now = clock.GetUtcNow();
    if (await repository.HasActiveGrantAsync(userId, form.VersionId, now, ct)) // ㉒ 已有有效授權就不必再申請
        return AccessResult.Refused("你對這個版本已有有效的授權，到期或被撤銷後才能再申請。");

    repository.Add(new ApiAccessRequest
    {
        UserId = userId, ApiVersionId = form.VersionId, Purpose = form.Purpose.Trim(),
        RequestedDays = form.Days, RequestedQuota = form.Quota, RequestedScope = form.Scope,
        RequestedAtUtc = now, Status = ReviewStatus.Pending
    });
    repository.Add(Audit(userId, "ApplyAccess", "ApiVersion", form.VersionId, $"申請使用 {version.ApiDefinition.Name} {version.Version}", now));

    try { await repository.SaveChangesAsync(ct); }
    catch (ConcurrencyConflictException) // ㉓ 撞到「一人一版本一件待審」的唯一索引
    {
        return AccessResult.Concurrent("你對這個版本已有一件待審的申請，請等待審核結果。");
    }
    return AccessResult.Ok($"已送出 {version.ApiDefinition.Name} {version.Version} 的使用申請。");
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	版本必須是 <code>Published</code>	審核中、已下架的版本不能申請；實測申請已下架的天氣預報 v1 得到「只能申請已發布的版本。」
—	用途、天數、額度、Scope 逐項檢查	表單的 DataAnnotations 可以被繞過（直接 POST），Service 不信任表單
②	已有「Active 且未到期」的 Grant 就拒絕	實測王小明申請商品查詢（當時 Grant #1 有效）得到「你對這個版本已有有效的授權…」
③	唯一索引衝突轉成友善訊息	沒有先查「有沒有待審申請」再新增 ：先查再寫在並行時兩個請求都會查到「沒有」；交給唯一索引決定才可靠

會員中心 `AccessController.Apply` 的 POST 把 `User.GetUserId()` 當申請人，並把表單欄位用 `?? ""` 補齊（Section 05 的教訓）。

預期結果（本單元實測）：林小華連送兩次課程查詢的申請——第一次「已送出 課程查詢 v1 的使用申請。」，第二次（繞過頁面直接 POST）「你對這個版本已有一件待審的申請，請等待審核結果。」；兩個請求**並行**送出服務工單的申請，資料庫只有 1 件待審。

步驟三 審核與撤銷（20 分鐘）

目的：核准（見教師示範的走讀）、駁回、撤銷共用的前提集中在一個方法。

```
// 範例/07_API使用申請與授權審核/McpPlatform/src/McpPlatform.Core/Access/AccessService.cs (CheckReviewable)
private static AccessResult? CheckReviewable(string reviewerId, ApiAccessRequest request, bool requirePublished)
{
    if (request.Status != ReviewStatus.Pending)
        return AccessResult.Refused($"這件申請已是「{request.Status}」，不能重複審核。");
    if (reviewerId == request.UserId)
        return AccessResult.Refused("不得審核本人提出的申請。");
    if (requirePublished && request.ApiVersion.Status != ApiVersionStatus.Published)
        return AccessResult.Refused("這個版本已不是「已發布」狀態，不能核准使用；請駁回。");
    return null;
}
```

程式說明：

檢查	核准	駁回	理由
申請必須是 <code>Pending</code>	✓	✓	事後再按一次核准，實測得到「這件申請已是「Approved」，不能重複審核。」
審核者不是申請人	✓	✓	與 Section 06 的「不得審核本人送出的案件」相同
版本仍是已發布	✓	✗	申請後版本被下架：不能核准，但 要能駁回 讓這件申請結案，所以用 <code>requirePublished</code> 參數區分

撤銷 `RevokeAsync` 只做三件事：Grant 的 `Status` 改 `Revoked`、記下 `RevokedAtUtc` 與 `RevokedReason`、寫一筆稽核紀錄。它**不碰**申請與審核紀錄。

預期結果（本單元實測，撤銷 Grant #1 後查資料庫）：

```
Grant #1 : Revoked ; 申請 #1 : Approved ; 申請 #1 的審核紀錄 : 1 筆
```

撤銷後王小明再申請商品查詢 v1，成功送出（新的申請 #8），因為已經沒有「有效」的 Grant。

步驟四 個人進度與 Grant 管理（15 分鐘）

目的：做出圖 07-4、07-5 兩個頁面。

「我的申請與授權」要把每件申請和它產生的 Grant（可能沒有）排在同一列，用 LINQ 的 left join：

```
// 範例/07_API使用申請與授權審核/McpPlatform/src/McpPlatform.Infrastructure/Access/EfAccessRepository.cs (ListMineAsync)
public async Task<IEnumerable<MyAccessDto>> ListMineAsync(string userId, CancellationToken ct) =>
    await (from r in db.ApiAccessRequests.AsNoTracking()
           where r.UserId == userId
           join a in db.ApprovalRecords on r.Id equals a.AccessRequestId into decisions
           from a in decisions.DefaultIfEmpty()
           join g in db.ApiGrants on r.Id equals g.SourceAccessRequestId into grants // Grant 以來源申請連回來
           from g in grants.DefaultIfEmpty()
           orderby r.RequestedAtUtc descending
           select new MyAccessDto(r.Id, r.ApiVersionId, r.ApiVersion.ApiDefinition.Name, r.ApiVersion.Version, r.Purpose,
                                r.RequestedDays, r.RequestedQuota, r.Status, r.RequestedAtUtc, a != null ? a.Reason : null,
                                g != null ? g.Id : null, g != null ? g.Status : null, g != null ? g.ExpiresAtUtc : null, g != null ? g.Quota : null))
        .ToListAsync(ct);
```

程式說明：

寫法	翻成的 SQL	為什麼
<code>where r.UserId == userId</code>	<code>WHERE r.UserId = @userId</code>	只看自己的申請
<code>join ... into x + from ... in x.DefaultIfEmpty()</code>	<code>LEFT JOIN</code>	待審的申請沒有審核紀錄、被駁回的沒有 Grant，仍要列出來
<code>g != null ? g.Id : null</code>	<code>CASE WHEN ... END</code>	LEFT JOIN 沒對到時整列是 NULL
<code>orderby</code> 寫在 <code>select new MyAccessDto(...)</code> 之前	<code>ORDER BY</code>	對建構子建立的 DTO 再做篩選或排序，EF Core 翻不成 SQL（見〈常見錯誤〉）

後臺 `GrantsController.Index` 把目前時間放進 `ViewData["Now"]`，畫面用「Active 且 `ExpiresAtUtc <= now`」顯示「🕒 已到期」；撤銷表單送到 `Revoke`，原因空白會被拒絕。

預期結果：畫面與圖 07-4、07-5 一致。

檢查點

做到這裡，你應該可以：

- ☐ 公開詳細頁有「申請使用」按鈕；未登入按下會先轉到登入頁
- ☐ 同一版本連送兩次申請，資料庫只有 1 件 `Pending`
- ☐ 核准後 `ApiGrants` 多一列，`SourceAccessRequestId` 指回申請；駁回後沒有
- ☐ 撤銷後「我的申請與授權」左欄仍是「✓ 已核准」、右欄變成「⊖ 已撤銷」

常見錯誤

以下都是撰寫本單元時實際遇到的原文。

症狀（錯誤訊息原文）	原因	處理
打開審核頁得到 500： <code>System.InvalidOperationException: The LINQ expression 'DbSet().Where(ti => new AccessCaseDto(...).RequestId == ...)' could not be translated.</code>	先 <code>select new AccessCaseDto(...)</code> 投影，再 <code>.Where(c => c.RequestId == id)</code> ：EF Core 不知道建構子參數對應哪個欄位	篩選、排序寫在投影之前： <code>CaseQuery(r => r.Id == requestId)</code> 把條件作用在實體上
重複申請時，網站終端機出現 <code>Microsoft.Data.SqlClient.SqlException (0x80131904): 無法以唯一索引 'UX_ApiAccessRequests_OnePendingPerUserVersion' 在物件 'dbo.ApiAccessRequests' 中插入重複的索引鍵資料列。重複的索引鍵值是 (member-b, 2)。</code>	同一人對同一版本已有待審申請（或兩個請求並行送出），第二筆撞到篩選唯一索引；EF Core 在丟出例外前會把原始錯誤記到日誌	這是預期的保護，使用者看到的是「你對這個版本已有一件待審的申請」。日誌裡的錯誤不必處理
Grant 的 Id 不連續（#1、#4、#5）	交易回復或唯一索引衝突時，SQL Server 已經配出去的 IDENTITY 值不會回收	正常現象；不要用 Id 連續與否判斷資料是否遺失
以 <code>CONVERT(varchar(20), RowVersion, 1)</code> 查詢，rowversion 顯示空白	顯示格式問題，不是資料問題	直接 <code>SELECT RowVersion</code> 或 <code>DATALENGTH(RowVersion)</code> （應為 8）

單元驗收（15 分鐘）

驗收標準：核准才建立 Grant；同案重複送出或重複核准不建立重複權限。

驗收方式：

1. **核准才建立 Grant：**送出一件申請後查 `SELECT COUNT(*) FROM ApiGrants WHERE SourceAccessRequestId = <申請 Id>` 得 0；核准後得 1；另一件被駁回的申請始終是 0（本單元實測駁回林小華的服務工單申請後為 0）。
2. **同案重複送出不建立重複申請：**同一版本連送兩次（包括並行送出），`Pending` 的申請只有 1 件。
3. **同案重複核准不建立重複權限：**兩個管理者工作階段並行核准同一件申請，一個成功、一個得到「剛剛已被其他人處理」；事後再核准得到「不能重複審核」。`ApiGrants` 與 `ApprovalRecords` 對該申請各只有 1 筆。
4. **反向驗證：**在 SSMS 對已有 Grant 的申請再新增一筆 Grant（`SourceAccessRequestId` 相同），必須得到 訊息 2601 ... `'IX_ApiGrants_SourceAccessRequestId'`（Section 03 已實測）。

回顧：申請綁定版本、記用途；核准的同一個交易建立 Grant；「當初核准」與「現在可用」分開存；重複由唯一索引與 rowversion 擋下。下一單元（Section 08）開始真的呼叫上游：API Gateway 依已核准的版本組出請求，並防範 SSRF。

延伸閱讀

- [EF Core 複雜查詢運算子](#) —— `GroupJoin` 與 `DefaultIfEmpty` 翻成 LEFT JOIN。
- [EF Core 用戶端與伺服器評估](#) —— 哪些 LINQ 能翻成 SQL、翻不動時的錯誤。
- [SQL Server IDENTITY 屬性](#) —— 為什麼識別值可能不連續。

Gateway、MCP 與 OAuth

S08～S11

以 API Gateway 安全呼叫已核准的上游，把已發布版本映射成 MCP Tools，透過 OAuth 與 PKCE 讓真實 Connector 連線並對應會員，並在每次工具呼叫時重新驗證授權、版本與額度。

第 8 章 APIGateway與安全呼叫

HttpClientFactory、逾時、取消、錯誤映射、機密保護、SSRF 防護與寫入幕等；ApiInvocationService、以核准版本組出請求、精確端點允許清單、攔截轉址

第 9 章 MCPServer與Tools映射

官方 C# SDK、Streamable HTTP、tools/list、tools/call、Tool 名稱、描述與 inputSchema；建立 /mcp、由已發布版本建立 ToolDefinition、工具呼叫交給 API Gateway

第 10 章 ConnectorOAuth與會員授權

網站 Cookie 與 Access Token 的差異；PKCE、Redirect URI、Scope、Audience 及資源中繼資料探索；套用 OpenIddict 授權服務骨架、登錄測試 Client、以 Issuer + Subject 對應會員

第 11 章 動態工具權限與呼叫管制

工具清單依權限過濾；每次呼叫重新驗證 Grant、Scope、版本、會員狀態、期限與剩餘額度；共用授權服務、原子額度計數、Rate Limit、撤銷時讓授權快取失效、錯誤回應一致化

第 8 章

APIGateway與安全呼叫

HttpClientFactory、逾時、取消、錯誤映射、機密保護、SSRF 防護與寫入冪等；ApiInvocationService、以核准版本組出請求、精確端點允許清單、攔截轉址



課程時數 2 小時 (120 分鐘) 當堂成果 受控上游呼叫與統一結果處理 範例專案 範例/08_APIGateway與安全呼叫

學習目標

上完這個單元，你能夠：

- 寫一個 `ApiInvocationService`，讓所有對上游的呼叫都走同一條路：呼叫端只給「版本 Id + 參數」，URL、Method、標頭、憑證全部由已核准的版本決定。
- 在送出前依序檢查：版本已發布、參數符合核准的 JSON Schema、端點在精確允許清單內；任何一項不通過都**不送出請求**。
- 說明 SSRF 是什麼，並用 `SocketsHttpHandler.ConnectCallback` 在連線當下擋住解析到私有、回送、雲端中繼資料位址的網域。
- 把上游的各種失敗（逾時、連不上、4xx／5xx、轉址、回應過大）映射成固定的錯誤代碼，並為每次呼叫寫一筆帶 CorrelationId 的 `ApiCallLogs`。
- 說明為什麼 POST 不自動重試、為什麼一律帶 `Idempotency-Key`，並實測同一把鍵重送得到同一張工單。

先備知識

- 完成 Section 07：已發布的版本與 Grant。本單元的 Gateway **還不檢查 Grant**（Section 11 才加上），測試台只開放管理者使用。
- Section 01 的三支示範 API（`http://localhost:5080`）與服務工單的 `Idempotency-Key`、`confirmed` 規則；Section 05 的 `ISchemaValidator` 與 `ISecretProtector`。
- 知道 `HttpClient` 與 `CancellationToken` 的基本用法；知道 HTTP 3xx 是轉址。
- 本單元範例的示範 API 多了兩個教師測試端點，要**重新啟動** `McpPlatform.DemoApis`。

觀念說明（20 分鐘）

為什麼需要 Gateway

平台代替會員（和他們的 AI）去呼叫別人的 API。如果每個地方各自 `new HttpClient()` 呼叫，會發生：逾時設定不一致、錯誤各自處理、憑證散落各處、呼叫紀錄漏寫。更危險的是：只要有一處讓呼叫端影響了網址或標頭，平台就成了攻擊者的跳板。

所以本課程只有一個地方會對上游送出 HTTP 請求 —— `ApiInvocationService`。測試台（本單元）、MCP 工具（Section 09）、呼叫前檢查（Section 11）都經過它。

呼叫端能決定什麼、不能決定什麼

項目	由誰決定	理由
要呼叫哪個版本	呼叫端（版本 Id）	會員選擇要用的工具
參數值	呼叫端，但必須符合版本的 <code>InputSchema</code>	參數是業務內容；Schema 是審核過的規格
URL、Method	已核准的版本	呼叫端能改網址，就能叫平台打任何地方
標頭（含 API Key）	Gateway	呼叫端能加標頭，就能偽造身分或覆蓋憑證
逾時、回應大小上限	Gateway 設定	避免一支慢 API 或巨大回應拖垮平台

`InvocationRequest` 這個型別裡根本沒有 URL 與 Header 欄位。測試台的表單就算多送 `Url=http://169.254.169.254/`、`Headers=Authorization: Bearer x`，也沒有任何程式會去讀它們。

SSRF：讓伺服器替攻擊者打內網

SSRF（Server-Side Request Forgery，伺服器端請求偽造）是指攻擊者讓伺服器替他發出請求，去存取攻擊者自己連不到的地方。典型目標：

目標	例子	後果
雲端中繼資料服務	<code>http://169.254.169.254/latest/meta-data/</code>	取得主機的雲端憑證
本機服務	<code>http://localhost:1433</code> 、 <code>http://127.0.0.1:8080/admin</code>	存取只綁在本機的管理介面
公司內網	<code>http://10.0.0.5/</code>	繞過防火牆

在本平台，攻擊的入口是「登錄一支 API」：會員填的端點 URL 會被平台呼叫。防線分四層：

1. **審核**（Section 06）：管理者看得到端點，可以退回。但人會看漏。
2. **精確端點允許清單**：放在伺服器設定 `appsettings.json` 的 `Gateway:AllowedEndpoints`，只有系統管理者能改。版本的端點範本（去掉查詢字串）必須與清單中某一項**逐字相同**，否則不送出。會員無法透過任何頁面擴張這份清單。
3. **連線當下檢查 IP**：網域在清單內，但 DNS 解析到私有或本機位址，照樣拒絕。這一步在 `ConnectCallback` 裡做 —— 由我們自己解析 DNS、自己連線，所以 DNS 在檢查之後才被改成指向內網（DNS rebinding）也沒用。
4. **不跟隨轉址**：上游回 `302 Location: http://169.254.169.254/...` 時，平台不會跟過去。

一次呼叫經過的檢查

下圖是 `InvokeAsync` 的檢查順序。注意：①～③ **任何一項不通過，請求根本不會送出**，耗時是 0 ms。

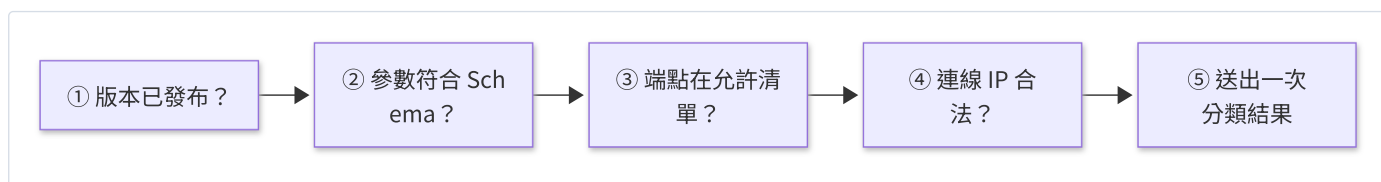


圖 08-1 Gateway 送出請求前的四道檢查

編號	不通過時的錯誤代碼	結果分類	本單元實測例子
①	VERSION_NOT_FOUND 、 VERSION_NOT_PUBLISHED	Denied	已下架的天氣預報 v1：「目前是「Retired」，不能呼叫。」
②	INVALID_ARGUMENTS	InvalidInput	{"keyword":123}：/keyword type:Value is "integer" but should be "string"
③	ENDPOINT_NOT_ALLOWED	Denied	雲端中繼資料位址、zip.example.test 都不在清單
④	PRIVATE_ADDRESS_BLOCKED	Denied	localtest.me 在清單內，但解析為 127.0.0.1
⑤	UPSTREAM_TIMEOUT 、 UPSTREAM_UNREACHABLE 、 UPSTREAM_HTTP_ERROR 、 REDIRECT_BLOCKED 、 RESPONSE_TOO_LARGE ； 成功為空字串	Timeout / UpstreamError / Success	見教師示範的完整結果表

錯誤代碼是**固定的英文字串**，給程式判斷（Section 11 決定要不要計入額度、Section 13 統計錯誤分布）；`Message` 是給人看的中文說明。

逾時、取消與重試

- **逾時**：`Gateway.TimeoutSeconds`（5 秒）。`HttpClient.Timeout` 設成無限，由 Service 自己建立一個 5 秒後取消的 `CancellationToken`。這樣才分得出「上游太慢」（逾時）與「呼叫端自己放棄」（例如使用者關掉頁面）。
- **取消**：呼叫端的 `CancellationToken` 與逾時的 Token 連結在一起，任一個觸發都會中止請求。
- **重試**：**POST 絕不自動重試**。平台不知道第一次是真的失敗還是「上游已經建立工單、只是回應在路上遺失」；自動重送可能建立兩張工單。GET 雖然理論上可以重試，本課程也不做，把重試決定留給呼叫端。
- **冪等鍵**：所有 POST 都帶 `Idempotency-Key`（呼叫端有給就用呼叫端的，沒給就用 `CorrelationId`），上游可以據此識別重送。

機密與紀錄

- API Key 在 `BuildRequest` 裡**才解密**，而且在允許清單檢查**之後**：不在清單的端點連解密都不會發生。
- `ApiCallLogs` 記使用者、Client、版本、結果分類、HTTP 狀態、耗時、`CorrelationId`、錯誤代碼；**不記**參數與回應內容（大綱：呼叫日誌不預設保存完整敏感請求／回應）。
- 每次呼叫都帶 `X-Correlation-Id` 標頭給上游，出問題時雙方可以用同一個代碼對日誌。

限制與注意事項

- 本單元的 Gateway **不檢查 Grant、Scope、額度**——這些在 Section 11 加在 Gateway 前面。所以測試台只放在後臺。
- 允許清單以「版本端點範本」比對，含 `{trackingNo}` 這類路徑參數的端點要把範本原樣寫進清單。路徑參數值一律 URL 編碼，`../` 會變成 `..%2F`，無法跳出路徑。
- 正式環境的允許清單應該只有真正上游網域，不要有 `localhost`；本機開發為了三支示範 API 才放行 `http://localhost:5080/...`。
- 連不上上游時不一定立刻失敗：本單元實測連線被拒的情況耗時 4066 ms（作業系統在回報失敗前會重試連線）。逾時上限仍是 5 秒。

名詞對照

名詞	白話說明	在本課程中是
API Gateway	所有對外呼叫的唯一出口	<code>ApiInvocationService</code>
SSRF	讓伺服器替攻擊者發請求去打內網	由允許清單與 <code>SsrfGuard</code> 防範
允許清單	只准列出的項目，其餘一律拒絕	<code>Gateway:AllowedEndpoints</code>
DNS rebinding	網域先解析到合法位址通過檢查，連線時改指內網	由 <code>ConnectCallback</code> 在連線當下檢查
冪等鍵	讓上游識別「這是同一個請求的重送」	<code>Idempotency-Key</code> 標頭
CorrelationId	一次呼叫的追查代碼	<code>ApiCallLogs.CorrelationId</code> 、 <code>X-Correlation-Id</code>

教師示範（25 分鐘）

示範情境：在後臺「API 測試台」依序呼叫 12 種情況，證明「成功、參數錯誤、上游逾時都有可辨識結果」，而且呼叫者無法指定 URL 或 Header；最後在資料庫看每次呼叫留下的紀錄。

示範步驟

1. 在 [範例/08_APIGateway與安全呼叫/McpPlatform](#) 對資料庫執行 `sql/05_gateway_test_apis.sql`：寫入四支教師測試 API（逾時測試、轉址測試、內網中繼資料、本機別名測試），直接是「已發布」狀態，只供本單元測試 Gateway。輸出最後列出它們的版本 Id（本機實測 15~18）。
2. 啟動示範 API 與網站，以管理者登入，側欄「API 測試台」。選「商品查詢 v1」，參數 `{"keyword": "茶"}`，送出。

The screenshot displays the MCP Server API Platform's API Testing Interface. The top navigation bar includes links for API 目錄, 會員中心, and 管理後臺, along with a system administrator login button. The sidebar on the left lists various management functions, with 'API 測試台' (API Testing) selected. The main content area is titled 'API 測試台' and contains a description of the testing process. It features a form for selecting an API version (商品查詢 v1) and entering parameters ({"keyword": "茶"}). A '送出呼叫' (Send Call) button is at the bottom of the form. To the right, the '呼叫結果' (Call Results) panel shows a successful outcome with a status of 'Success', HTTP status 200, and a response time of 16 ms. The response body is a JSON object containing product information.

管理後臺

- 總覽
- API 管理
- 資料庫概況
- 上架審核
- 使用申請審核
- Grant 管理
- API 測試台
- 會員管理 (Section 12)

API 測試台

經由 API Gateway 在伺服器端呼叫已發布的版本。網址、方法、標頭與憑證都由已核准的版本決定，這裡只能填參數。

已發布的版本

商品查詢 v1 (GET · 版本 #1)

參數 (JSON 物件)

`{"keyword": "茶"}`

幕等鍵 (POST 用 · 選填)

留空則以追查代碼代替

送出呼叫

呼叫結果

結果	✓ Success
說明	成功
HTTP	200
耗時	16 ms
追查	3615944f640e476f
回應	<pre>{ "count": 3, "items": [{ "sku": "P-1001", "name": "阿里山高山烏龍茶 150g", "category": "茶葉", "price": 680, "stock": 42 }, { "sku": "P-1002", "name": "日月潭紅玉紅茶 100g", "category": "茶葉", "price": 520, "stock": 0 }, { "sku": "P-3001", "name": "黑糖薑母茶磚", "category": "沖泡飲", "price": 250, "stock": 60 }] }</pre>

© 2026 MCP Server API 平台 | [環境檢查](#)

圖 08-2 API 測試台：成功呼叫商品查詢

畫面重點：

1. ①左欄只有三個欄位：版本、參數、幕等鍵。沒有網址、方法、標頭的輸入框——這就是「不可由呼叫者指定任意 URL 或 Header」在介面上的樣子。
2. ②右欄「結果」：✓ Success、HTTP 200、耗時 16 ms。
3. ③「追查」：16 碼 CorrelationId，同一個代碼寫進 `ApiCallLogs`，也以 `X-Correlation-Id` 標頭送給上游。
4. ④「回應」：上游回傳的 JSON 原文（3 筆茶類商品）。
5. 依序換成其他版本與參數。本單元實測的完整結果：

情況	結果	錯誤代碼	HTTP	耗時	
1 成功（商品查詢）	Success		200	47 ms	
2 參數型別錯 {"keyword":123}	InvalidInput	INVALID_ARGUMENTS	-	0 ms	
3 參數不在列舉 {"level":"高手"}	InvalidInput	INVALID_ARGUMENTS	-	0 ms	
4a POST 第一次（鍵 K-001）	Success		201	12 ms	T-00001
4b POST 同一把鍵重送	Success		200	21 ms	T-00001（同一張）
4c POST 沒有 confirmed	UpstreamError	UPSTREAM_HTTP_ERROR	400	7 ms	
5 上游逾時	Timeout	UPSTREAM_TIMEOUT	-	5014 ms	
6 上游轉址	UpstreamError	REDIRECT_BLOCKED	302	2 ms	
7 雲端中繼資料位址	Denied	ENDPOINT_NOT_ALLOWED	-	0 ms	
8 網域解析到 127.0.0.1	Denied	PRIVATE_ADDRESS_BLOCKED	-	0 ms	
9 郵遞區號（不在允許清單）	Denied	ENDPOINT_NOT_ALLOWED	-	0 ms	
10 偷塞 Url、Headers 欄位	Success		200	0 ms	回應與情況 1 相同
11 已下架的天氣預報 v1	Denied	VERSION_NOT_PUBLISHED	-	0 ms	
12 示範 API 沒開	UpstreamError	UPSTREAM_UNREACHABLE	-	4066 ms	

4. 情況 8 的畫面：

呼叫結果

結果	✕ Denied PRIVATE_ADDRESS_BLOCKED
說明	localtest.me 解析為 127.0.0.1，屬於私有、本機或連結本機位址，已拒絕連線。
HTTP	—（沒有送出或沒有回應）
耗時	0 ms
追查	c0d5d87e143c4520
回應	—

圖 08-3 允許清單內的網域，因為解析到 127.0.0.1 而被擋下

畫面重點：

- ① ✕ Denied 與 PRIVATE_ADDRESS_BLOCKED：這個版本的端點 `http://localtest.me:5080/api/products` 有在允許清單裡（教師刻意加入示範），所以通過了圖 08-1 的 ③。
- ② 說明「localtest.me 解析為 127.0.0.1...已拒絕連線」：SsrfGuard 在連線前自己查 DNS，發現是回送位址，而網域本身不是明寫的 localhost，所以拒絕。
- ③ HTTP「—」、耗時 0 ms：沒有任何 TCP 連線被建立。
- 在 SSMS 查最近的呼叫紀錄：

```
-- 在 MCPServerManag 執行；只讀
SELECT TOP 6 Id, UserId, ClientId, ApiVersionId, Outcome, HttpStatus, DurationMs, CorrelationId, ErrorCode
FROM dbo.ApiCallLogs ORDER BY Id DESC;
```

本單元實測（節錄 3 列）：

```
19|admin|admin-console|1|UpstreamError|NULL|4066|bcfe987a581e4ee1|UPSTREAM_UNREACHABLE
15|admin|admin-console|18|Denied|NULL|0|29eccbedfc2a4940|PRIVATE_ADDRESS_BLOCKED
14|admin|admin-console|17|Denied|NULL|0|674627035f434139|ENDPOINT_NOT_ALLOWED
```

被擋下的呼叫也有紀錄；表中沒有任何參數或回應內容的欄位。

程式碼走讀：一次呼叫的前半段

`InvokeAsync` 在 `McpPlatform.Core/Gateway/ApiInvocationService.cs`，由測試台 `InvokeController` 呼叫（Section 09 起也由 MCP 工具呼叫）。前半段是圖 08-1 的 ①～③：

```
// 範例/08_APIGateway與安全呼叫/McpPlatform/src/McpPlatform.Core/Gateway/ApiInvocationService.cs (InvokeAsync 前半：送出前檢查)
public async Task<InvocationResult> InvokeAsync(InvocationRequest request, CancellationToken ct)
{
    var correlationId = Guid.NewGuid().ToString("N")[..16]; // ① 每次呼叫一個追查代碼
    var target = await targets.GetAsync(request.VersionId, ct);
    if (target is null)
        return await FinishAsync(request, null, Fail(CallOutcome.Denied, GatewayErrors.VersionNotFound, "找不到這個版本。", correlationId), ct);
    if (target.Status != ApiVersionStatus.Published) // ② 未核准、已下架都不能呼叫
        return await FinishAsync(request, target, Fail(CallOutcome.Denied, GatewayErrors.VersionNotPublished, $"{target.ApiName} {target.Version} 目前是「{target.Status}」，不能呼叫。", correlationId), ct);

    var problems = schemas.ValidateInstance(target.InputSchema, request.ArgumentsJson); // ③ 參數必須符合核准的 Schema
    if (problems.Count > 0)
        return await FinishAsync(request, target, Fail(CallOutcome.InvalidInput, GatewayErrors.InvalidArguments, "參數不符合規格：" + string.Join("；", problems.Take(3)), correlationId), ct);

    if (!IsAllowed(target.EndpointUrl)) // ④ 精確端點允許清單（在解密憑證之前）
        return await FinishAsync(request, target, Fail(CallOutcome.Denied, GatewayErrors.EndpointNotAllowed, $"端點 {Template(target.EndpointUrl)} 不在允許清單內，平台不會送出請求。", correlationId), ct);

    UpstreamRequest upstreamRequest;
    try { upstreamRequest = BuildRequest(target, request, correlationId); }
    catch (ArgumentException ex)
    {
        return await FinishAsync(request, target, Fail(CallOutcome.InvalidInput, GatewayErrors.InvalidArguments, ex.Message, correlationId), ct);
    }
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	產生 16 碼 CorrelationId	被擋下的呼叫也有代碼，使用者回報問題時可以直接給這串
②	版本必須是 <code>Published</code>	審核中、被退回、已下架的版本一律不能呼叫（Section 06 的「未核准版本不可使用」在這裡落實到呼叫端）
③	用版本自己的 <code>InputSchema</code> 驗證參數	參數不合規格的請求在平台就被擋下，不會打到上游
④	允許清單比對，而且放在 <code>BuildRequest</code> 之前	<code>BuildRequest</code> 會解密憑證；不在清單的端點連解密都不必發生
—	每個失敗分支都經過 <code>FinishAsync</code>	寫呼叫紀錄的地方只有一個，不會漏

學員實作（60 分鐘）

範例專案：範例/08_APIGateway與安全呼叫/McpPlatform/（從 Section 07 複製後往上加）。本單元新增 Core 的 Gateway/（GatewayContracts.cs、ApiInvocationService.cs）、Infrastructure 的 Gateway/（SsrGuard.cs、HttpUpstreamClient.cs、EfGatewayData.cs）、Web 的後臺 InvokeController 與「API 測試台」頁、appsettings.json 的 Gateway 節點；示範 API 多了 /api/slow、/api/redirect 兩個測試端點；sql/05_gateway_test_apis.sql。本單元沒有新的 Migration。

步驟一 組出請求：參數只能填值（15 分鐘）

目的：依版本的端點範本與 Method 組出 UpstreamRequest；呼叫端的參數只能填進「路徑參數」「查詢字串」或「POST 本文」。

```
// 範例/08_APIGateway與安全呼叫/McpPlatform/src/McpPlatform.Core/Gateway/ApiInvocationService.cs (BuildRequest)
private UpstreamRequest BuildRequest(InvocationTarget target, InvocationRequest request, string correlationId)
{
    using var args = JsonDocument.Parse(request.ArgumentsJson);
    var root = args.RootElement;
    var used = new HashSet<string>(StringComparer.Ordinal);

    var url = PathParameter().Replace(target.EndpointUrl, m =>
    {
        var name = m.Groups[1].Value;
        if (!root.TryGetProperty(name, out var value))
            throw new ArgumentException($"缺少路徑參數 {name}。");
        used.Add(name);
        return Uri.EscapeDataString(Scalar(value, name)); // 「../」會變成「..%2F」，無法跳出路徑
    });

    string? body = null;
    if (target.HttpMethod == "GET")
    {
        var query = new List<string>();
        foreach (var name in DeclaredProperties(target.InputSchema).Where(n => !used.Contains(n))) // 只送 Schema 宣告過的參數
            if (root.TryGetProperty(name, out var value))
                query.Add($"{Uri.EscapeDataString(name)}={Uri.EscapeDataString(Scalar(value, name))}");
        if (query.Count > 0)
            url += (url.Contains('?') ? "&" : "?") + string.Join("&", query);
    }
    else
    {
        body = request.ArgumentsJson;
    }

    var headers = new Dictionary<string, string>
    {
        ["Accept"] = "application/json",
        ["X-Correlation-Id"] = correlationId // 上游的日誌也能用同一個代碼追查
    };
    if (target.HttpMethod == "POST")
        headers["Idempotency-Key"] = request.IdempotencyKey ?? correlationId; // 寫入一律帶冪等鍵
    if (target.AuthType == ApiAuthType.ApiKeyHeader && target.CredentialProtectedValue is { } secret)
        headers[target.CredentialHeader!] = secrets.Unprotect(secret); // 憑證只在這一刻解密

    return new UpstreamRequest(target.HttpMethod, new Uri(url), headers, body);
}
```

程式說明：

片段	做了什麼	為什麼這樣寫
PathParameter().Replace(...)	把端點範本中的 {trackingNo} 換成參數值，並用 Uri.EscapeDataString 編碼	值只能成為一個路徑片段，不能帶 /、?、# 改變網址結構
DeclaredProperties(...)	GET 只把 Schema properties 宣告過的參數放進查詢字串	Schema 沒宣告的參數（例如 url）不會被送出
POST 本文 = 參數原文	參數已通過 Schema 驗證	服務工單的 confirmed 由呼叫端在參數中給；平台不替它決定
headers 字典只在這裡建立	Accept、Correlation Id、冪等鍵、憑證	呼叫端沒有任何管道加入標頭
secrets.Unprotect(secret)	Section 05 加密的 API Key 在此解密	明文只存在於這一次請求的記憶體中

預期結果：呼叫商品查詢 {"keyword": "茶"} 時，組出的請求是 GET http://localhost:5080/api/products?keyword=%E8%8C%B6（「茶」經 URL 編碼），回應 3 筆資料（圖 08-2）。

步驟二 允許清單與連線當下的 IP 檢查 (20 分鐘)

目的：兩層 SSRF 防護 —— 設定檔的精確端點清單、連線時的 IP 檢查。

允許清單在 `src/McpPlatform.Web/appsettings.json` 的最外層：

```
"Gateway": {
  "TimeoutSeconds": 5,
  "MaxResponseBytes": 1048576,
  "AllowedEndpoints": [
    "http://localhost:5080/api/products",
    "http://localhost:5080/api/courses",
    "http://localhost:5080/api/tickets",
    "http://localhost:5080/api/slow",
    "http://localhost:5080/api/redirect",
    "http://localtest.me:5080/api/products"
  ]
}
```

鍵	值	說明
TimeoutSeconds	5	單次呼叫最多等 5 秒；實測逾時測試在 5014 ms 被中止
MaxResponseBytes	1048576 (1 MB)	超過就回 <code>RESPONSE_TOO_LARGE</code> ，不把整份回應讀進記憶體
AllowedEndpoints	6 個精確端點	前 5 個是示範與測試端點；最後一個 <code>localtest.me</code> 是刻意放進來示範 DNS 檢查，正式環境不要有

清單在伺服器設定檔裡，會員無論登錄什麼 API、審核通過與否，都不能讓這份清單多一項。

連線當下的檢查在 `SsrfGuard`：

```
// 範例/08_APIGateway與安全呼叫/McpPlatform/src/McpPlatform.Infrastructure/Gateway/SsrfGuard.cs (ConnectAsync)
public static async ValueTask<Stream> ConnectAsync(SocketsHttpContext context, CancellationToken ct)
{
    var host = context.DnsEndPoint.Host;
    var addresses = await Dns.GetHostAddressesAsync(host, ct); // ㉑ 自己解析，不交給作業系統自動挑
    var explicitLoopback = host.Equals("localhost", StringComparison.OrdinalIgnoreCase)
        || (IPAddress.TryParse(host, out var literal) && IPAddress.IsLoopback(literal));
    var allowed = addresses
        .Where(ip => explicitLoopback ? IPAddress.IsLoopback(ip) : !IsPrivateOrLocal(ip)) // ㉒ 只有「明寫 localhost」才能連本機
        .ToArray();
    if (allowed.Length == 0)
        throw new UpstreamBlockedException(
            $"{host} 解析為 {string.Join("、", addresses.Select(a => a.ToString()))}，屬於私有、本機或連結本機位址，已拒絕連線。");

    var socket = new Socket(SocketType.Stream, ProtocolType.Tcp) { NoDelay = true };
    try
    {
        await socket.ConnectAsync(allowed, context.DnsEndPoint.Port, ct); // ㉓ 只連檢查過的 IP
        return new NetworkStream(socket, ownsSocket: true);
    }
    catch
    {
        socket.Dispose();
        throw;
    }
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	在建立 TCP 連線前自己查 DNS	若先查一次 DNS 做檢查、再讓 HttpClient 自己連線，兩次查詢之間 DNS 可以改變 (DNS rebinding)
②	網域必須明寫為 <code>localhost</code> 或回送 IP，才准連本機；其他網域只要解析到私有範圍就排除	<code>IsPrivateOrLocal</code> 涵蓋 10/8、172.16/12、192.168/16、127/8、169.254/16 (含雲端中繼資料)、100.64/10、IPv6 的 <code>::1</code> 、 <code>fe80::/10</code> 、 <code>fc00::/7</code>
③	用檢查過的那批 IP 連線	連到的一定是剛剛驗證過的位址

`Program.cs` 把它接到專用的 `HttpClient`：

```
// 範例/08_APIGateway與安全呼叫/McpPlatform/src/McpPlatform.Web/Program.cs (上游 HttpClient)
builder.Services.AddHttpClient<IUpstreamClient, HttpUpstreamClient>() // ① 具型別 HttpClient
    .ConfigurePrimaryHttpMessageHandler(() => new SocketsHttpHandler
    {
        AllowAutoRedirect = false, // ② 不跟隨轉址
        ConnectCallback = SsrfGuard.ConnectAsync, // ③ 連線前檢查 IP
        PooledConnectionLifetime = TimeSpan.FromMinutes(2) // ④ 定期重建連線，DNS 變更才會生效
    })
    .ConfigureHttpClient(c => c.Timeout = Timeout.InfiniteTimeSpan); // ⑤ 逾時由 Service 以 CancellationToken 控制
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	<code>IHttpClientFactory</code> 管理的具型別用戶端	不必自己 <code>new HttpClient()</code> ，連線池由框架管理，不會耗盡通訊埠
②	<code>AllowAutoRedirect = false</code>	轉址會把請求帶到清單外； <code>HttpUpstreamClient</code> 收到 3xx 直接回報 <code>REDIRECT_BLOCKED</code>
③	每個新連線都經過 <code>SsrfGuard.ConnectAsync</code>	這是 HttpClient 唯一建立 TCP 連線的地方
④	連線最多重用 2 分鐘	連線池長期保留舊連線時，DNS 變更不會生效
⑤	<code>HttpClient.Timeout</code> 設無限	預設 100 秒的逾時會與 Service 的 5 秒互相干擾，也分不出逾時與取消

預期結果：雲端中繼資料位址 → `ENDPOINT_NOT_ALLOWED`（沒送出）；`localtest.me` → `PRIVATE_ADDRESS_BLOCKED`（圖 08-3）；轉址測試 → `REDIRECT_BLOCKED`，HTTP 302。

步驟三 送出、逾時與結果分類（15 分鐘）

目的：送出一次請求，把所有可能的結果分成固定代碼，並寫紀錄。

`InvokeAsync` 的後半段：

```
// 範例/08_APIGateway與安全呼叫/McpPlatform/src/McpPlatform.Core/Gateway/ApiInvocationService.cs (InvokeAsync 後半：送出與分類)
using var timeout = CancellationTokenSource.CreateLinkedTokenSource(ct);
timeout.CancelAfter(TimeSpan.FromSeconds(options.TimeoutSeconds)); // ⑥ 逾時與呼叫端取消分開判斷
var stopwatch = Stopwatch.StartNew();
InvocationResult result;
try
{
    var response = await upstream.SendAsync(upstreamRequest, options.MaxResponseBytes, timeout.Token); // ⑦ 只送一次，不自動重試
    result = Map(response, (int)stopwatch.ElapsedMilliseconds, correlationId);
}
catch (UpstreamBlockedException ex)
{
    result = Fail(CallOutcome.Denied, GatewayErrors.PrivateAddress, ex.Message, correlationId);
}
catch (OperationCanceledException) when (ct.IsCancellationRequested)
{
    result = Fail(CallOutcome.UpstreamError, GatewayErrors CallerCancelled, "呼叫端已取消。", correlationId, (int)stopwatch.ElapsedMilliseconds);
}
catch (OperationCanceledException)
{
    result = Fail(CallOutcome.Timeout, GatewayErrors.UpstreamTimeout, $"上游 {options.TimeoutSeconds} 秒內沒有回應。", correlationId, (int)stopwatch.ElapsedMilliseconds);
}
catch (HttpRequestException ex)
{
    result = Fail(CallOutcome.UpstreamError, GatewayErrors.UpstreamUnreachable, "連不上上游：" + ex.Message, correlationId, (int)stopwatch.ElapsedMilliseconds);
}
return await FinishAsync(request, target, result, CancellationToken.None); // ⑧ 取消了也要寫紀錄
```

程式說明：

標號	做了什麼	為什麼這樣寫
⑤	建立一個與呼叫端 Token 連結、5 秒後自動取消的 Token	兩個來源都能中止請求
⑥	<code>SendAsync</code> 只呼叫一次	POST 不盲目重試；結果交給 <code>Map</code> 依狀態碼分類（2xx 成功、3xx <code>REDIRECT_BLOCKED</code> 、其他 <code>UPSTREAM_HTTP_ERROR</code> ）
—	<code>catch ... when (ct.IsCancellationRequested)</code> 放在一般 <code>OperationCanceledException</code> 前面	呼叫端的 Token 被取消 = 呼叫端放棄；否則就是我們的 5 秒逾時
—	<code>HttpRequestException</code>	連不上、連線被拒；實測訊息「無法連線，因為目標電腦拒絕連線。(localhost:5080)」
⑦	寫紀錄時傳 <code>CancellationToken.None</code>	呼叫端已取消時仍要把這次呼叫記下來；傳原本的 <code>ct</code> 會讓寫入也被取消

`HttpUpstreamClient.SendAsync` 以 `HttpCompletionOption.ResponseHeadersRead` 先取得標頭：3xx 直接回報轉址、不讀本文；其他狀態碼只讀到 `MaxResponseBytes + 1` 位元組為止，超過就標記 `TooLarge`。

預期結果：逾時測試 → `Timeout`、`UPSTREAM_TIMEOUT`、約 5000 ms；關掉示範 API → `UPSTREAM_UNREACHABLE`；每一次都在 `ApiCallLogs` 多一列。

步驟四 API 測試台與幕等驗證（10 分鐘）

目的：做出圖 08-2 的頁面，並驗證寫入幕等。

`Admin/InvokeController` 的 POST 只從表單讀 `VersionId`、`ArgumentsJson`、`IdempotencyKey` 三個欄位，組成 `InvocationRequest`（呼叫者 = `User.GetUserId()`、`ClientId = "admin-console"`）交給 `ApiInvocationService`。Admin Area 的慣例（Section 04）讓只有管理者能用這一頁。

幕等驗證：選「服務工單建立 v1」，參數 `{"subject": "禮盒缺件", "confirmed": true}`，幕等鍵填 `K-001`，送出兩次。

預期結果（本單元實測）：第一次 `HTTP 201`、`T-000001`；第二次 `HTTP 200`、仍是 `T-000001`。拿掉 `"confirmed": true` 並換一把鍵，得到 `UpstreamError`、`UPSTREAM_HTTP_ERROR`、`HTTP 400`，回應內容是上游的「尚未確認」——平台不會替呼叫端補上確認。

檢查點

做到這裡，你應該可以：

- ☐ 測試台只有「版本、參數、幕等鍵」三個欄位；多送 `Url`、`Headers` 欄位，結果與沒送完全相同
- ☐ 參數型別錯、不在列舉內都在 0 ms 內回 `INVALID_ARGUMENTS`，上游完全沒收到請求
- ☐ 逾時、轉址、中繼資料位址、解析到本機四種情況各有自己的錯誤代碼
- ☐ `ApiCallLogs` 每次呼叫一列，包含被擋下的呼叫，且沒有參數與回應內容

常見錯誤

以下都是撰寫本單元時實際遇到的原文。

症狀（錯誤訊息原文）	原因	處理
執行測試 API 腳本：訊息 156 ... 接近關鍵字 'Schema' 之處的語法不正確。	SCHEMA 是 T-SQL 保留字，不能當欄位名稱	改名（本課程用 InputSchemaJson），或以 [Schema] 括起來
測試台結果 UPSTREAM_UNREACHABLE：連不上上游：無法連線，因為目標電腦拒絕連線。（localhost:5080）	示範 API 沒有啟動（或還在跑 Section 07 以前的舊版，沒有 /api/slow）	在本單元的範例目錄執行 dotnet run --project src/McpPlatform.DemoApis
自動化擷圖或測試腳本在送出後「被登出」	腳本用 form button[type=submit] 選取送出按鈕，先選到導覽列的「登出」表單	選取器指定表單：form[action*= "Invoke"] button[type=submit]
允許清單有 http://localhost.me:5080/api/products 卻得到 PRIVATE_ADDRESS_BLOCKED	localhost.me 解析為 127.0.0.1，不是明寫的 localhost	這正是 DNS 檢查在運作；本機開發請直接寫 localhost

單元驗收（15 分鐘）

驗收標準：成功、參數錯誤、上游逾時均有可辨識結果；不可由呼叫者指定任意 URL 或 Header。

驗收方式：

1. **成功有可辨識結果**：商品查詢 `{"keyword": "茶"}` → `Success`、HTTP 200、有 `CorrelationId` 與回應內容。
2. **參數錯誤有可辨識結果**：`{"keyword": 123}` → `InvalidInput`、`INVALID_ARGUMENTS`，說明指出 `/keyword` 的型別錯誤；耗時 0 ms，上游沒收到請求。
3. **上游逾時有可辨識結果**：逾時測試 → `Timeout`、`UPSTREAM_TIMEOUT`、耗時約 5000 ms（實測 5014 ms），說明「上游 5 秒內沒有回應。」
4. **不可由呼叫者指定任意 URL 或 Header**：送出商品查詢時額外夾帶 `Url=http://169.254.169.254/`、`Headers=Authorization: Bearer x`、`EndpointUrl=http://evil.test`，結果與正常呼叫相同（實測 `Success`、HTTP 200、回應為商品清單）。對端點在清單外的版本（雲端中繼資料、郵遞區號）→ `ENDPOINT_NOT_ALLOWED`，沒有送出任何請求。
5. **反向驗證**：清單內的 `localhost.me` 仍被 `PRIVATE_ADDRESS_BLOCKED` 擋下；上游回 302 時不跟隨，得到 `REDIRECT_BLOCKED`。

回顧：Gateway 是唯一的出口；呼叫端只給版本與參數；允許清單在設定檔、IP 檢查在連線當下、轉址不跟隨；錯誤都有固定代碼、每次都有紀錄。下一單元（Section 09）建立 `/mcp` 端點，把已發布的版本映射成 MCP 工具，工具的執行就是呼叫這個 Gateway。

延伸閱讀

- [OWASP SSRF 防範速查表](#) —— 允許清單、DNS rebinding、轉址等防範手法。
- [在 ASP.NET Core 使用 IHttpClientFactory](#) —— 具型別用戶端與 `ConfigurePrimaryHttpMessageHandler`。
- [SocketsHttpHandler.ConnectCallback](#) —— 自訂連線建立流程。
- [.NET 中的工作取消](#) —— 連結多個 `CancellationToken`。

第 9 章

MCP Server 與 Tools 映射

官方 C# SDK、Streamable HTTP、tools/list、tools/call、Tool 名稱、描述與 inputSchema；建立 /mcp、由已發布版本建立 ToolDefinition、工具呼叫交給 API Gateway



課程時數 2 小時 (120 分鐘) 當堂成果 /mcp、工具探索與呼叫示範 範例專案 範例/09_MCP Server 與 Tools 映射

學習目標

上完這個單元，你能夠：

- 用官方 MCP C# SDK (`ModelContextProtocol.AspNetCore` 2.2.0) 在網站上開出 `/mcp` 端點，並說明 Streamable HTTP 一次請求長什麼樣子。
- 在後臺把一個「已發布」的 API 版本映射成 MCP 工具：指定工具名稱，平台自動帶入說明與 `inputSchema`；名稱不合規則、Schema 不合 MCP 規定、名稱重複都會被拒絕。
- 寫出 `tools/list` 與 `tools/call` 兩個處理常式：工具清單從資料庫即時讀出，版本下架後工具自動消失；工具呼叫一律交給 Section 08 的 `ApiInvocationService`。
- 分辨協定錯誤 (JSON-RPC `error`，例如 `-32602` 沒有這個工具) 與工具錯誤 (`result.isError = true`，例如參數不符合 Schema)，並讓每個工具結果都帶 `CorrelationId`。
- 用測試用戶端 (官方 SDK 的 `McpClient`) 連上 `/mcp`，完成探索、成功呼叫、參數錯誤、不存在工具四種情況。

先備知識

- 完成 Section 08： `ApiInvocationService` 能呼叫已發布的版本，每次呼叫寫一筆 `ApiCallLogs`。本單元的 MCP 工具就是再多一個呼叫 Gateway 的入口。
- Section 03 已建好的 `ToolDefinitions` 資料表（ `ToolName`、 `ApiVersionId` 各有唯一索引）。之前一直是空的，本單元開始使用。
- Section 05 的版本 `InputSchema`（JSON Schema）與 Section 06 的「已發布」狀態。
- Section 01 講過 MCP 的三個角色（Host、Client、Server）與「工具」的概念；示範 API（ `http://localhost:5080` ）要啟動。
- 本單元沒有新的 Migration。

觀念說明（20 分鐘）

為什麼需要 /mcp

到 Section 08 為止，只有管理者能在「API 測試台」呼叫上游 API。專題的目標是讓會員的 AI 應用（Claude、ChatGPT、VS Code 裡的 Copilot 等）也能用這些 API。AI 應用不會讀我們的網頁，它們講的是 MCP（Model Context Protocol，模型脈絡協定）：一套規定「AI 應用怎麼問伺服器有哪些工具、怎麼呼叫工具」的公開協定。

所以平台要多開一個入口 /mcp，說 MCP 這種語言。這個入口不重做任何安全檢查：它只負責把 MCP 的請求翻譯成 `InvocationRequest`，再交給 Gateway。版本狀態、Schema、允許清單、SSRF、逾時、紀錄，全部沿用 Section 08。

下圖是一次工具呼叫在平台裡經過的路徑：



圖 09-1 一次 MCP 工具呼叫經過的元件

編號	做了什麼	對應程式碼
①	AI 應用（本單元用測試用戶端代替）送出 JSON-RPC 請求，標頭帶 <code>Authorization: Bearer <Token></code>	<code>tools/McpPlatform.McpTestClient/Program.cs</code>
②	沒帶或帶錯 Token 一律 401，請求進不到 MCP 處理常式	<code>DevTokenAuthenticationHandler</code> 、 <code>MapMcp("/mcp").RequireAuthorization(...)</code>
③	用工具名稱查 <code>ToolDefinitions</code> ，找出背後的 API 版本；版本已不是「已發布」就當作沒有這個工具	<code>McpToolHandlers.CallToolAsync</code> 、 <code>EfToolRepository.ListActiveToolsAsync</code>
④	版本、參數、允許清單、連線 IP 逐一檢查後才送出，並寫呼叫紀錄	<code>ApiInvocationService.InvokeAsync</code> （Section 08）
⑤	上游只收到 Gateway 組出來的請求，看不到 MCP 的任何東西	示範 API

官方 C# SDK

MCP 的訊息格式是 **JSON-RPC 2.0**：每個請求有 `jsonrpc`、`id`、`method`、`params`，回應有 `result` 或 `error`。自己手寫解析與驗證很容易出錯，本課程用官方維護的 SDK（版本 2.2.0，寫在 `Directory.Packages.props`）：網站用 `ModelContextProtocol.AspNetCore`，提供 `AddMcpServer()`、`WithHttpTransport()`、`MapMcp()`，把 MCP 伺服器掛到 ASP.NET Core 的路由上；測試用戶端用 `ModelContextProtocol`，提供 `McpClient` 與 `HttpClientTransport`。

SDK 有兩種定義工具的方式：一種是在 C# 方法上加 `[McpServerTool]` 屬性，工具在編譯時就固定；另一種是註冊 `WithListToolsHandler` 與 `WithCallToolHandler` 兩個處理常式，自己決定清單內容。本平台的工具來自資料庫（管理者映射、版本下架都會改變清單），所以用第二種。

Streamable HTTP：一次 POST 一個 JSON-RPC 請求

MCP 規定了幾種傳輸方式，遠端伺服器用的是 **Streamable HTTP**：Client 把 JSON-RPC 請求用 `POST` 送到同一個網址（本平台是 `/mcp`），標頭 `Accept` 要同時接受 `application/json` 與 `text/event-stream`；伺服器可以直接回 JSON，也可以用 SSE（Server-Sent Events）格式一則一則送回。

本平台設定 `Stateless = true`（無狀態）：伺服器不記任何連線狀態，每個 POST 各自獨立。下面是本單元實測的原始交換（用 PowerShell 直接 POST，沒有經過 SDK 的 Client）：

```
POST /mcp
Authorization: Bearer <你的 Token>
Accept: application/json, text/event-stream
Content-Type: application/json

{"jsonrpc": "2.0", "id": 1, "method": "tools/call", "params": {"name": "courses_search", "arguments": {"level": "高手"}}
```

```
200 Content-Type: text/event-stream
```

```
event: message
data: {"result":{"content":[{"type":"text","text":"[INVALID_ARGUMENTS] 參數不符合規格: /level enum: Value should match one of the values specified by the enum"}, {"type":"text","text":"CorrelationId: eff85696200d40c2"}], "isError":true, "id":1, "jsonrpc":"2.0"}
```

看這段回應時注意三件事：

1. **HTTP 是 200**。參數錯誤不是 HTTP 層的錯，是工具執行的結果，寫在 JSON 裡的 `isError`。
2. `event: message` / `data:` 是 SSE 的框架，裡面才是 JSON-RPC 回應；`id: 1` 對應請求的 `id`。
3. 實際收到的中文是 `\u53C3\u6578...` 這種寫法。這是 JSON 標準的跳脫字元，不是亂碼；任何 JSON 解析器讀出來都是「參數不符合規格」。上面為了好讀已經還原成中文。

tools/list 與 tools/call

MCP 的工具功能只有兩個方法：`tools/list` 問「有哪些工具」，`tools/call` 說「用這些參數執行這個工具」。下圖是測試用戶端的完整交談：

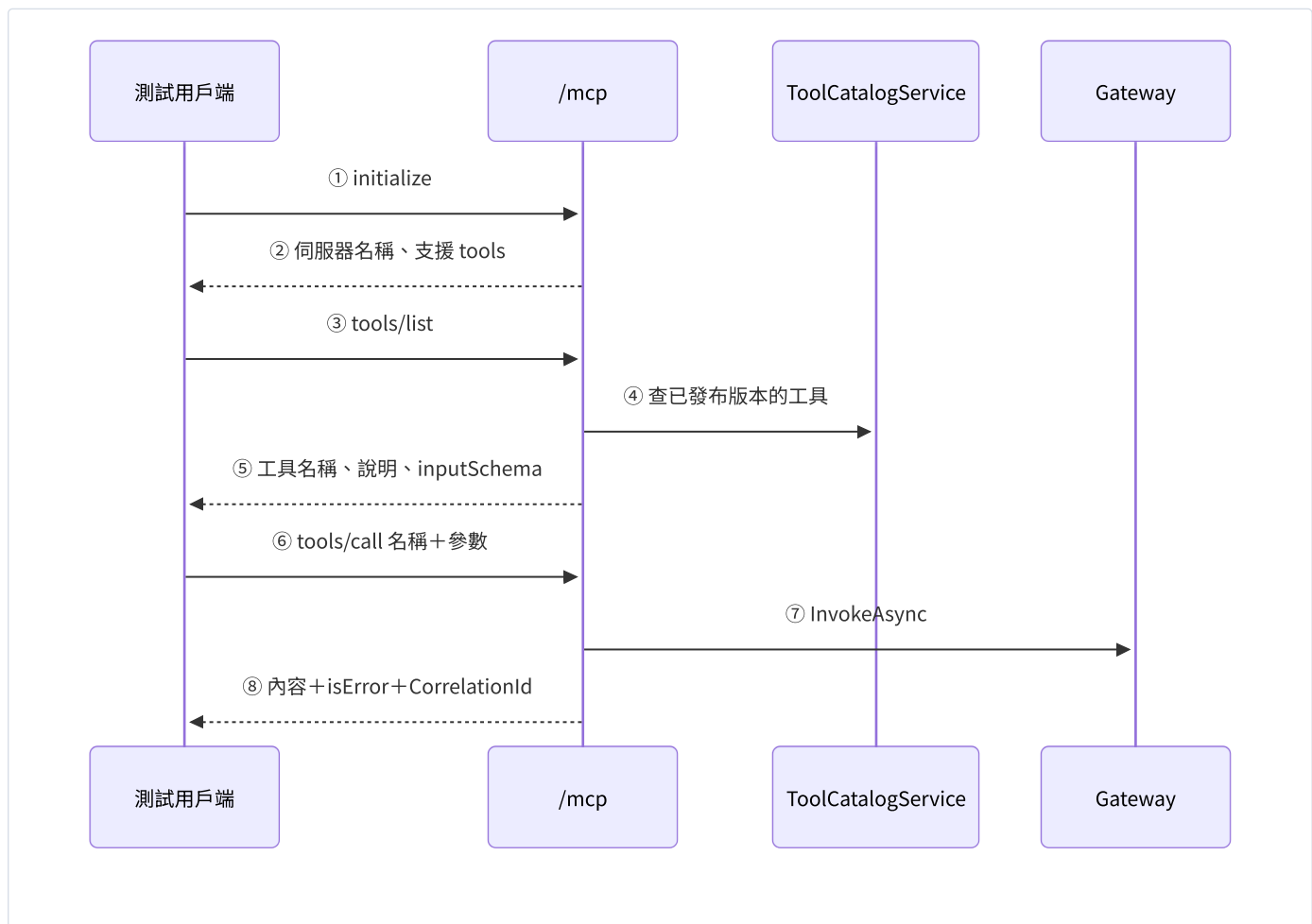


圖 09-2 測試用戶端與 /mcp 的交談順序

編號	內容
① ②	握手：Client 送名稱與協定版本；伺服器回實測 <code>protocolVersion: 2025-11-25</code> 、 <code>serverInfo: McpPlatform.Web 1.0.0.0</code> ， <code>capabilities</code> 有 <code>tools</code>
③ ④ ⑤	<code>tools/list</code> 沒有參數； <code>ListToolsAsync</code> 只取版本狀態是「已發布」的工具，回傳每個工具的 <code>name</code> 、 <code>title</code> 、 <code>description</code> 、 <code>inputSchema</code> 、 <code>annotations</code>
⑥ ⑦	<code>tools/call</code> 帶 <code>name</code> 與 <code>arguments</code> （JSON 物件）；組成 <code>InvocationRequest</code> （版本 Id、參數、會員、ClientId、工具名稱）交給 Gateway
⑧	<code>content</code> 第一段是上游回應或錯誤說明、第二段是 <code>CorrelationId</code> ，另有 <code>isError</code>

① ② 由 SDK 自動處理，我們只寫 ④ 與 ⑦ 兩段。無狀態模式下 Client 其實可以不作 `initialize` 直接送 `tools/call`（上一節的 PowerShell 實測就是這樣），但正式的 Client 都會先握手。

Tool 的名稱、描述與 inputSchema

`tools/list` 回給 AI 的每個工具有五個欄位。以實測的 `courses_search` 為例：`title` 是「課程查詢 v1」，`inputSchema` 是 `{ "type": "object", "properties": { "level": { "type": "string", "enum": [ "入門", "中階", "進階" ] } } }`，`annotations` 是 `{ "readOnlyHint": true }`。

欄位	來源	AI 怎麼用它	本課程的規則
<code>name</code>	管理者在映射頁指定	呼叫時用的識別字	<code>^[a-z][a-z0-9_]{2,63}\$</code> ：小寫英文開頭，3~64 字； 建立後不改 ，Client 會記住它
<code>title</code>	API 名稱 + 版本	顯示給人看	自動產生
<code>description</code>	API 名稱（版本）：API 說明	判斷什麼時候該用這個工具	映射當下從版本複製
<code>inputSchema</code>	版本的 <code>InputSchema</code>	決定要填哪些參數、什麼型別	與審核通過的版本相同； 最外層必須是 <code>"type": "object"</code>
<code>annotations.readOnlyHint</code>	版本的 Method	提示 Client 這個工具不會改資料	GET 為 <code>true</code> ，POST 為 <code>false</code>

`inputSchema` 最外層必須是 `"type": "object"`，這是 MCP 的規定（工具參數永遠是一個具名參數的物件）。SDK 在設定 `Tool.InputSchema` 時會檢查，不合格就丟出 `ArgumentException`。本單元實際踩到：天氣預報 v2 的 Schema 是 `{}`，一映射上去，**整個 `tools/list` 就失敗**（見〈常見錯誤〉）。所以映射時先檢查，不合格的版本根本映射不進來。

協定錯誤與工具錯誤

工具呼叫失敗有兩種回法，給 AI 的意義不同：

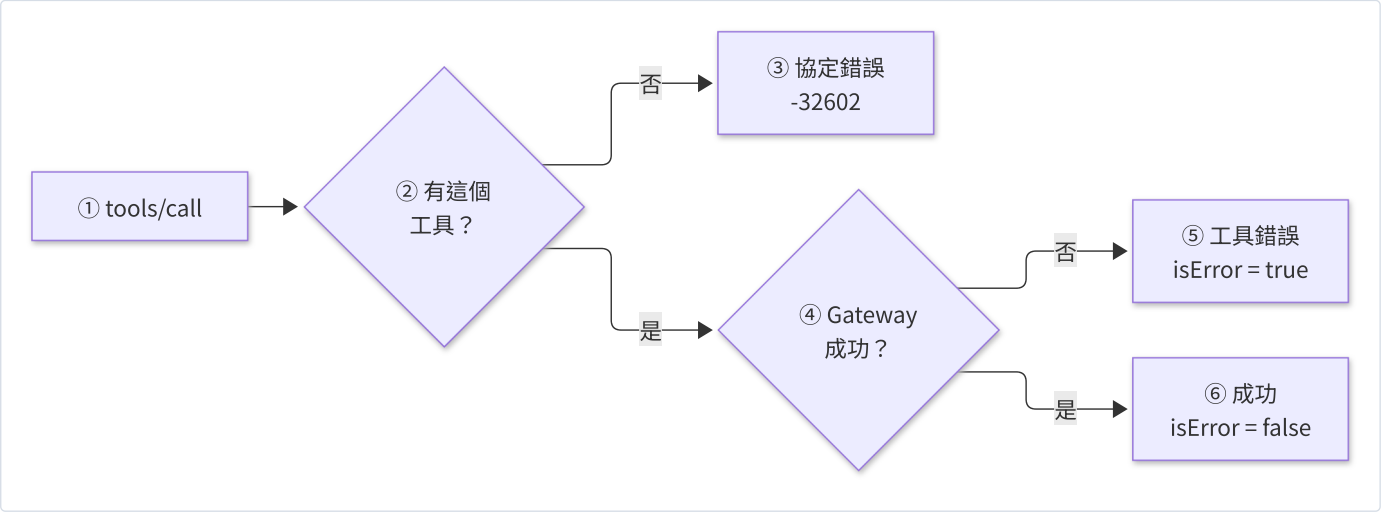


圖 09-3 tools/call 的三種結果

編號	觸發條件	回應長相	AI 會怎麼處理
② → ③	名稱不存在、或背後的版本已下架	<code>{ "error": { "code": "-32602", "message": "沒有名為 no_such_tool 的工具。" } }</code> ，沒有 <code>result</code>	請求本身就是錯的，重送也沒用；應重新 <code>tools/list</code>
④ → ⑤	找到工具，但 Gateway 回傳失敗： <code>INVALID_ARGUMENTS</code> 、 <code>VERSION_NOT_PUBLISHED</code> 、 <code>UPSTREAM_TIMEOUT</code> …	<code>result.isError = true</code> ，文字是【錯誤代碼】說明	AI 看得到錯誤內容，可以修正參數再試
④ → ⑥	Gateway 成功	<code>result.isError = false</code> ，文字是上游回應	讀回應內容回答使用者
⑤ ⑥	兩種結果都有	第二段文字 <code>CorrelationId: ...</code>	使用者回報問題時可以附上這串

`-32602` 是 JSON-RPC 規定的「Invalid params」代碼。MCP 規範建議：**找不到工具**屬於協定錯誤；**工具執行時的失敗**（包含參數不合商業規則）放在 `isError`，讓 AI 能自我修正。

限制與注意事項

- **本單元的 Token 是暫時的。** `/mcp` 認得一組固定的 Bearer Token（放在 User Secrets），並把它當成 `member-a`。它不會過期、不能撤銷、沒有 Scope，只適合在自
己的電腦上開發。Section 10 換成 OAuth 簽發的 Access Token。
- **本單元的 Gateway 仍不檢查 Grant。** `member-a` 沒有商品查詢的 Grant 也能呼叫 `products_search`；Section 11 才在 Gateway 前面加上 Grant、額度與停權檢
查。
- **一個版本對應一個工具。** 天氣預報出了 v3，要再映射一個新名稱，舊的工具在 v2 下架時自動消失。
- **無狀態模式不會主動通知。** 管理者新增映射後，Client 要再呼叫一次 `tools/list` 才看得到；伺服器不會推送「工具清單變了」。直接對 `/mcp` 發 `GET`（開啟伺服器
推送串流）會得到 404。

名詞對照

名詞	白話說明	在本課程中是
MCP	AI 應用與工具伺服器之間的公開協定	<code>/mcp</code> 端點
JSON-RPC 2.0	以 JSON 表示「呼叫哪個方法、參數、回應或錯誤」的格式	MCP 的訊息格式
Streamable HTTP	MCP 的遠端傳輸：POST 送請求，回應可以是 JSON 或 SSE	<code>WithHttpTransport()</code>
無狀態（Stateless）	伺服器不保存連線狀態，每個 POST 各自獨立	<code>options.Stateless = true</code>
Tool（工具）	AI 可以呼叫的一個功能，有名稱、說明、參數規格	一筆 <code>ToolDefinitions</code> 對應一個已發布版本
inputSchema	工具參數的 JSON Schema	複製自版本的 <code>InputSchema</code>
協定錯誤	JSON-RPC 的 <code>error</code> 回應	找不到工具 → <code>-32602</code>
工具錯誤	正常的 <code>result</code> ，但 <code>isError = true</code>	Gateway 回傳的失敗

教師示範 (25 分鐘)

示範情境：管理者把四個已發布的版本映射成工具，看到兩種映射被拒；測試用戶端探索並呼叫工具，得到成功、工具錯誤、協定錯誤三種結果；最後把郵遞區號查詢 v1 下架，工具從清單消失。

示範步驟

1. 產生一組隨機 Token，存進網站專案的 User Secrets（在 [範例/09_MCPServer與Tools映射/McpPlatform](#) 執行）：

```
$token = [Convert]::ToHexString([Security.Cryptography.RandomNumberGenerator]::GetBytes(32)) # 64 個十六進位字元
dotnet user-secrets set "Mcp:DevAuth:Token" $token --project src/McpPlatform.Web
$token # 記下來，步驟 4 要用
```

2. 啟動示範 API 與網站（`dotnet run --project src/McpPlatform.DemoApis`、`dotnet run --project src/McpPlatform.Web --urls http://localhost:5199`），以管理者登入，側欄點「MCP 工具映射」。這一頁列出所有「已發布」的版本；還沒映射的版本右邊有輸入框。
3. 依序映射，並看頁面上方的訊息。本單元實測：

版本	輸入的工具名稱	結果
1	Products Search	工具名稱只能用小寫英文、數字、底線，以英文字母開頭，3~64 字，例如 <code>products_search</code> 。
1	<code>products_search</code>	已將 商品查詢 v1 映射為工具 <code>products_search</code> 。
2	<code>courses_search</code>	已將 課程查詢 v1 映射為工具 <code>courses_search</code> 。
3	<code>ticket_create</code>	已將 服務工單建立 v1 映射為工具 <code>ticket_create</code> 。
5	<code>weather_forecast</code>	這個版本的輸入 Schema 最外層不是 <code>"type": "object"</code> ，MCP 不接受；請發布新版本修正 Schema 後再映射。
11	<code>zip_lookup</code>	已將 郵遞區號查詢 v1 映射為工具 <code>zip_lookup</code> 。
2	<code>products_search</code>	工具名稱 <code>products_search</code> 已被使用，或這個版本已經映射過。

如果你的資料庫是本單元才全新建立的，`DemoDataSeeder` 已經建好 `products_search`、`courses_search`、`ticket_create` 三個工具，映射頁上它們直接顯示名稱，你只要映射郵遞區號查詢即可。

映射天氣預報 v2 被拒後的畫面：

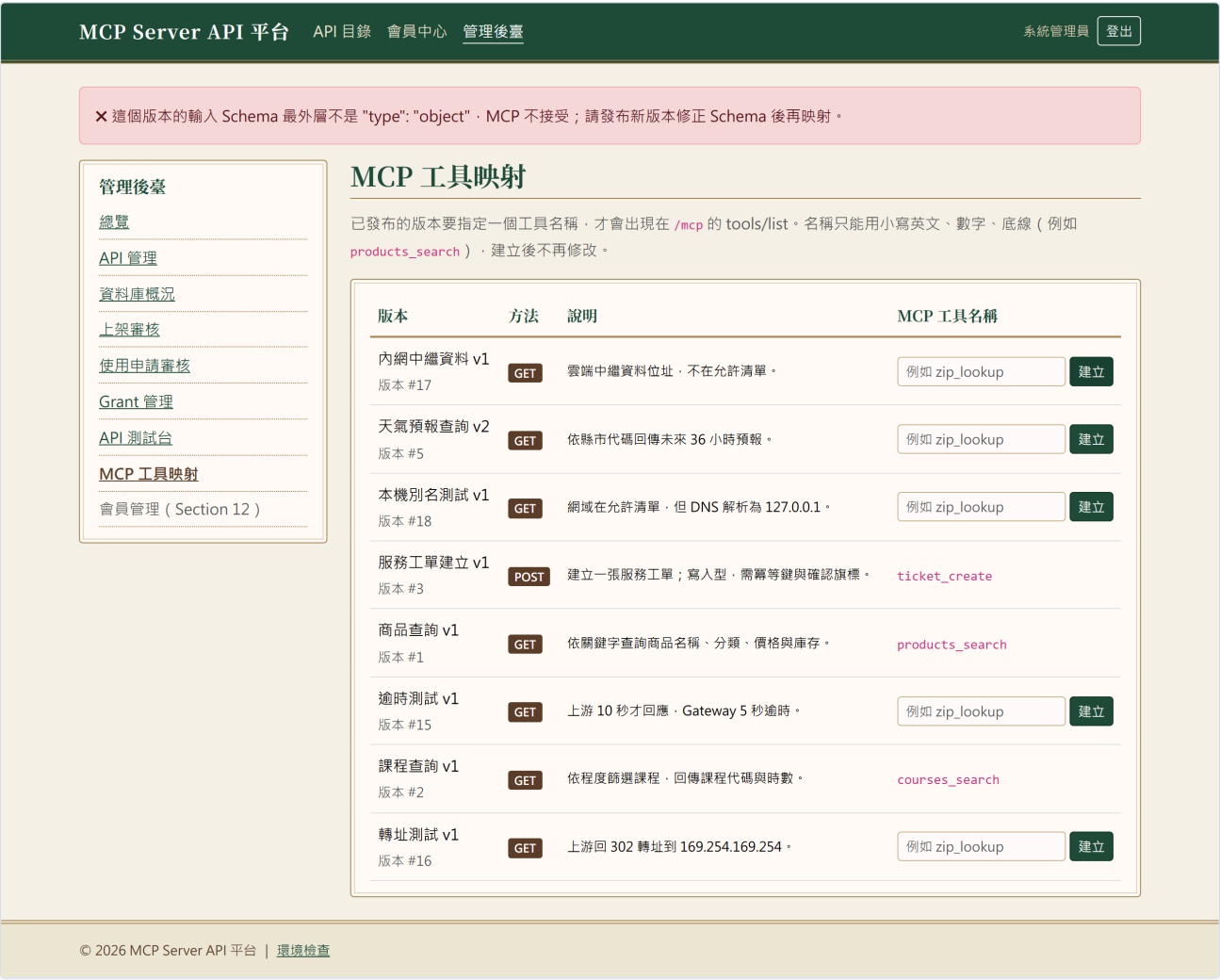


圖 09-4 後臺「MCP 工具映射」頁：Schema 不合 MCP 規定的版本被拒絕

畫面重點：

- ①上方紅色 ✕ 訊息：天氣預報 v2 的 Schema 是 `{}`，最外層沒有 `"type": "object"`，映射被拒；成功的訊息則是綠色 ✓。
- ②「MCP 工具名稱」欄已有名稱的列（`ticket_create`、`products_search`、`courses_search`）：已映射，只顯示名稱、不能再改。
- ③仍有輸入框的列：已發布、但還沒映射的版本。天氣預報 v2 因為被拒，仍在這裡。
- ④表格裡沒有「郵遞區號查詢 v1」：這張圖是在示範步驟 7 下架它之後擷取的；只有「已發布」的版本會出現在這一頁。
- 執行測試用戶端，把步驟 1 的 Token 當參數：

```
dotnet run --project tools/McpPlatform.McpTestClient -- $token
```

本單元實測輸出（下架郵遞區號查詢之前）：

```
已連線：McpPlatform.Web 1.0.0.0

== tools/list: 共 4 個工具
courses_search    課程查詢 (v1)：依程度篩選課程，回傳課程代碼與時數。
products_search    商品查詢 (v1)：依關鍵字查詢商品名稱、分類、價格與庫存。
ticket_create      服務工單建立 (v1)：建立一張服務工單；寫入型，需幕等鍵與確認旗標。
zip_lookup         郵遞區號查詢 (v1)：輸入地址，回傳 3+3 郵遞區號與行政區。

== tools/call products_search {"keyword":"茶"}
isError = False
{"count":3,"items":[{"sku":"P-1001","name":"阿里山高山烏龍茶 150g","category":"茶葉","price":680,"stock":42},{"sku":"P-1...
CorrelationId: 1cd188ad633048bf

== tools/call courses_search {"level":"高手"}
isError = True
[INVALID_ARGUMENTS] 參數不符合規格：/level enum: Value should match one of the values specified by the enum
CorrelationId: f26fbc3e8564e1d

== tools/call no_such_tool {}
協定錯誤 -32602 (InvalidParams): Request failed (remote): 沒有名為 no_such_tool 的工具。
```

逐段對照圖 09-3：`products_search` 是 ⑥ 成功；`courses_search` 的「高手」不在 Schema 的 `enum`（入門／中階／進階）裡，是 ⑤ 工具錯誤，仍然有 `CorrelationId`；`no_such_tool` 是 ③ 協定錯誤，沒有 `result`，所以也沒有 `CorrelationId`。天氣預報 v2 被拒，所以清單裡沒有它。

5. 在 SSMS 用 `CorrelationId` 查紀錄：

```
-- 在 MCPServerManag 執行；只讀
SELECT TOP 2 UserId, ClientId, ApiVersionId, ToolName, Outcome, HttpStatus, CorrelationId, ErrorCode
FROM dbo.ApiCallLogs ORDER BY Id DESC;
```

本單元實測：

```
member-a|classroom-mcp-client|2|courses_search|InvalidInput|NULL|f26fbc3e8564e1d|INVALID_ARGUMENTS
member-a|classroom-mcp-client|1|products_search|Success|200|1cd188ad633048bf|NULL
```

與 Section 08 測試台的紀錄（`admin` / `admin-console`、`ToolName` 空白）相比，MCP 呼叫的紀錄多了是哪位會員、哪個 Client、哪個工具。

6. 用錯的 Token 再跑一次：`dotnet run --project tools/McpPlatform.McpTestClient -- wrong-token`。實測在連線階段就失敗：`HttpRequestException: Response status code does not indicate success: 401 (Unauthorized).`

7. 回到後臺「API 管理」，把「郵遞區號查詢 v1」下架（填下架原因），再跑一次步驟 4。實測 `tools/list`：共 3 個工具，只剩 `courses_search`、`products_search`、`ticket_create`。`zip_lookup` 消失了，網站沒有重新啟動。`ToolDefinitions` 裡那一列還在，只是 `ListActiveToolsAsync` 不再選它。此時呼叫 `zip_lookup` 會得到 `-32602`。

學員實作（60 分鐘）

範例專案：範例/09_MCPServer與Tools映射/McpPlatform/（從 Section 08 複製後往上加）。本單元新增 Core 的 `Mcp/ToolCatalog.cs`、Infrastructure 的 `Mcp/EfToolRepository.cs`、Web 的 `Mcp/`（`DevTokenAuthenticationHandler.cs`、`McpToolHandlers.cs`）、後臺 `ToolsController` 與「MCP 工具映射」頁、`_Layout.cshtml` 的紅色錯誤訊息；`Program.cs` 加上 MCP 的註冊與 `MapMcp`；`appsettings.Development.json` 加上 `Mcp:DevAuth`；新專案 `tools/McpPlatform.McpTestClient`。 `DemoDataSeeder` 在全新資料庫多建三個工具。

步驟一 由已發布版本建立 ToolDefinition（15 分鐘）

目的：寫出映射規則。名稱合法、版本已發布、Schema 合 MCP 規定，三項都通過才寫入 `ToolDefinitions`。

`MapAsync` 在 `McpPlatform.Core/Mcp/ToolCatalog.cs`，由後臺 `ToolsController.Map` 呼叫：

```
// 範例/09_MCPServer與Tools映射/McpPlatform/src/McpPlatform.Core/Mcp/ToolCatalog.cs (MapAsync 前半：三項檢查)
/// <summary>為一個已發布的版本建立工具映射。名稱建立後不再修改 (Client 會記住工具名稱)。</summary>
public async Task<(bool Ok, string Message)> MapAsync(int versionId, string toolName, CancellationToken ct)
{
    if (!ToolNamePattern().IsMatch(toolName)) // ① 名稱規則
        return (false, "工具名稱只能用小寫英文、數字、底線，以英文字母開頭，3~64 字，例如 products_search。");
    var version = await repository.FindVersionAsync(versionId, ct);
    if (version is null) return (false, "找不到這個版本。");
    if (version.Status != ApiVersionStatus.Published) return (false, "只有已發布的版本可以映射成工具。");
    if (!IsObjectSchema(version.InputSchema)) // ② MCP 的硬性規定
        return (false, "這個版本的輸入 Schema 最外層不是 \"type\": \"object\"，MCP 不接受；請發布新版本修正 Schema 後再映射。");
```

三項檢查都通過，才寫入一筆 `ToolDefinition`：

```
// 範例/09_MCPServer與Tools映射/McpPlatform/src/McpPlatform.Core/Mcp/ToolCatalog.cs (MapAsync 後半)
repository.Add(new ToolDefinition
{
    ToolName = toolName, ApiVersionId = versionId,
    Description = $"{version.ApiDefinition.Name} ({version.Version}) : {version.ApiDefinition.Description}", // ③ AI 靠說明判斷何時使用
    InputSchema = version.InputSchema // ④ 與核准的版本相同
});
try { await repository.SaveChangesAsync(ct); }
catch (ConcurrencyConflictException) { return (false, $"工具名稱 {toolName} 已被使用，或這個版本已經映射過。"); }
return (true, $"已將 {version.ApiDefinition.Name} {version.Version} 映射為工具 {toolName}。");
```

同一個檔案裡的 Schema 檢查與名稱規則：

```
// 範例/09_MCPServer與Tools映射/McpPlatform/src/McpPlatform.Core/Mcp/ToolCatalog.cs (IsObjectSchema、ToolNamePattern)
/// <summary>MCP 規定工具的 inputSchema 最外層必須是 {\"type\": \"object\"}；SDK 遇到不符的 Schema 會讓整個 tools/list 失敗。</summary>
private static bool IsObjectSchema(string json)
{
    try
    {
        using var doc = JsonDocument.Parse(json);
        return doc.RootElement.ValueKind == JsonValueKind.Object
            && doc.RootElement.TryGetProperty(\"type\", out var type)
            && type.ValueKind == JsonValueKind.String && type.GetString() == \"object\";
    }
    catch (JsonException) { return false; }
}

[GeneratedRegex(\"^[a-z][a-z0-9_]{2,63}$\")]
private static partial Regex ToolNamePattern();
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	名稱必須符合 <code>^[a-z][a-z0-9_]{2,63}\$</code>	各家 AI 應用對工具名稱的字元限制不同，只用小寫英文、數字、底線最保險； <code>Products Search</code> 有大寫與空格，被拒
—	版本必須是 <code>Published</code>	草稿、審核中、被退回的版本不能給 AI 用（與 Gateway 的第一道檢查一致）
②	<code>IsObjectSchema</code> ：能解析、是物件、 <code>type</code> 是字串 <code>"object"</code>	不合格的 Schema 一旦寫進資料表， <code>tools/list</code> 就整個失敗；在入口擋掉
③	說明寫成 <code>API 名稱 (版本)：API 說明</code>	AI 只看得到名稱與說明，說明要講清楚「做什麼、回傳什麼」
④	<code>InputSchema</code> 從版本複製一份	工具的參數規格就是審核通過的規格；Gateway 呼叫時也用同一份驗證
—	<code>ConcurrencyConflictException</code> 轉成訊息	<code>EfToolRepository.SaveChangesAsync</code> 把 SQL 錯誤 2601／2627（唯一索引衝突）轉成這個例外；名稱或版本重複都走這裡
—	<code>[GeneratedRegex]</code>	編譯時產生比對程式碼，不必每次建立 <code>Regex</code> 物件；類別因此要宣告成 <code>partial</code>

`ToolsController.Map` 依 `ok` 把訊息放進 `TempData["Notice"]`（綠色✓）或 `TempData["Error"]`；後者由 `_Layout.cshtml` 新增的一段顯示成紅色✕、`role="alert"`（螢幕閱讀器會立即念出）。共用一個鍵的話，拒絕訊息也會顯示綠色✓，容易被看成成功。

預期結果：依示範步驟 3 的順序映射，得到同樣的七個訊息；`SELECT ToolName, ApiVersionId FROM ToolDefinitions` 有四列。

常見改錯：訊息都是空白 → 檢查 `_Layout.cshtml` 是否有 `TempData["Error"]` 那段；只加了 Notice 的話，拒絕訊息不會顯示。

步驟二 暫時的開發用 Token（10 分鐘）

目的：`/mcp` 不能用網站的登入 Cookie（AI 應用沒有瀏覽器），改認 `Authorization: Bearer <Token>`。本單元先用固定 Token，把「驗證」與「工具」分開做通。

```
// 範例/09_MCPServer與Tools映射/McpPlatform/src/McpPlatform.Web/Mcp/DevTokenAuthenticationHandler.cs (HandleAuthenticateAsync)
protected override Task<AuthenticateResult> HandleAuthenticateAsync()
{
    var header = Request.Headers.Authorization.ToString();
    if (!header.StartsWith("Bearer ", StringComparison.Ordinal))
        return Task.FromResult(AuthenticateResult.NoResult()); // 沒帶 Token：交給授權階段回 401

    var token = header["Bearer ".Length..].Trim();
    if (string.IsNullOrEmpty(Options.Token) || !CryptographicEquals(token, Options.Token))
        return Task.FromResult(AuthenticateResult.Fail("Token 不正確。"));

    var identity = new ClaimsIdentity(
    [
        new Claim(ClaimTypes.NameIdentifier, Options.UserId), // 這個 Token 代表的會員
        new Claim(ClientIdClaim, Options.ClientId)           // 哪一個 MCP Client
    ], SchemeName);
    return Task.FromResult(AuthenticateResult.Success(new AuthenticationTicket(new ClaimsPrincipal(identity), SchemeName)));
}
```

程式說明：

片段	做了什麼	為什麼這樣寫
<code>AuthenticateResult.NoResult()</code>	沒有 Bearer 標頭時「不表態」	由授權原則決定結果（回 401）；不要在驗證階段自己寫回應
<code>string.IsNullOrEmpty(Options.Token)</code>	沒設定 Token 的環境一律失敗	正式機沒有這個 User Secret， <code>/mcp</code> 就不可能用空字串通過
<code>CryptographicEquals</code>	同檔案的小方法，內部呼叫 <code>CryptographicOperations.FixedTimeEquals</code> ，耗時與內容無關	一般 <code>==</code> 在第一個不同字元就停，攻擊者可以量回應時間逐字猜出 Token
<code>ClaimTypes.NameIdentifier</code>	會員 Id（設定值 <code>member-a</code> ）	<code>CallToolAsync</code> 用它當 <code>InvocationRequest.UserId</code>
<code>client_id</code> Claim	Client 名稱	寫進 <code>ApiCallLogs.ClientId</code> ；Section 10 改成 OAuth 登錄的 Client

設定值綁定 `Mcp.DevAuth` 節點：

鍵	放哪裡	值	設錯的後果
<code>Mcp:DevAuth:Token</code>	User Secrets	自己產生的 64 字元隨機字串（示範步驟 1）	沒設：每個請求都 401；寫進 <code>appsettings.json</code> ：Token 跟著原始碼外洩
<code>Mcp:DevAuth:UserId</code>	<code>appsettings.Development.json</code>	<code>member-a</code> （示範資料的會員 Id）	填不存在的 Id：呼叫紀錄的 <code>UserId</code> 對不到會員
<code>Mcp:DevAuth:ClientId</code>	<code>appsettings.Development.json</code>	<code>classroom-mcp-client</code>	只影響紀錄上的 Client 名稱

鍵名必須和 `DevTokenOptions` 的屬性名稱**完全相同**（`Token`、`UserId`、`ClientId`），`Bind` 才對得上；對不上不會報錯，只是屬性維持預設值（見〈常見錯誤〉）。

預期結果：不帶 Token 或帶錯 Token 呼叫 `/mcp`，HTTP 401；帶對的 Token，`initialize` 成功。

步驟三 建立 /mcp 與兩個處理常式（25 分鐘）

目的：註冊 MCP 伺服器、掛上 `/mcp`，並寫出 `tools/list` 與 `tools/call`。

`Program.cs` 的註冊（放在 Section 08 的 Gateway 註冊之後）：

```
// 範例/09_MCPServer與Tools映射/McpPlatform/src/McpPlatform.Web/Program.cs（服務註冊）
// Section 09: MCP Server
builder.Services.AddScoped<ToolCatalogService>();
builder.Services.AddScoped<IToolRepository, EfToolRepository>();
builder.Services.AddAuthentication()
    .AddScheme<DevTokenOptions, DevTokenAuthenticationHandler>(DevTokenAuthenticationHandler.SchemeName,
        options => builder.Configuration.GetSection("Mcp:DevAuth").Bind(options)); // ㉞ 暫時的開發用 Token（Section 10 換成 OAuth）
builder.Services.AddMcpServer()
    .WithHttpTransport(options => options.Stateless = true) // ㉞ 無狀態：每個 HTTP 請求各自獨立
    .WithListToolsHandler(McpToolHandlers.ListToolsAsync) // ㉞ tools/list
    .WithCallToolHandler(McpToolHandlers.CallToolAsync); // ㉞ tools/call
```

端點（放在 `areas` 路由之後、`default` 路由之前）：

```
// 範例/09_MCPServer與Tools映射/McpPlatform/src/McpPlatform.Web/Program.cs（端點）
// /mcp 端點：只接受 McpDevToken 驗證通過的呼叫，未帶或帶錯 Token 一律 401。
app.MapMcp("/mcp").RequireAuthorization(new AuthorizationPolicyBuilder(DevTokenAuthenticationHandler.SchemeName)
    .RequireAuthenticatedUser().Build()); // ㉞
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	新增一個名為 <code>McpDevToken</code> 的驗證方案	<code>AddAuthentication()</code> 不帶參數， 不會 改變網站預設的 Cookie 驗證；只有指定這個方案的端點才用它
②	Streamable HTTP、無狀態	不必保存工作階段，多台主機部署也不用共用狀態（Section 15）
③ ④	註冊兩個處理常式	工具清單來自資料庫，不用 <code>[McpServerTool]</code> 靜態宣告
⑤	授權原則 只接受 <code>McpDevToken</code> 方案，且要已驗證	網站的登入 Cookie 不能拿來呼叫 <code>/mcp</code> ；沒 Token 時由這條原則回 401（ <code>/mcp</code> 不是網頁，不會轉址到登入頁）

`tools/list` 處理常式與 `Tool` 的轉換（同在 `McpToolHandlers.cs`）：

```
// 範例/09_MCPServer與Tools映射/McpPlatform/src/McpPlatform.Web/Mcp/McpToolHandlers.cs（ListToolsAsync、ToProtocolTool）
public static async ValueTask<ListToolsResult> ListToolsAsync(RequestContext<ListToolsRequestParams> context, CancellationToken ct)
{
    var catalog = context.Services!.GetRequiredService<ToolCatalogService>();
    var tools = await catalog.ListActiveToolsAsync(ct);
    return new ListToolsResult { Tools = tools.Select(ToProtocolTool).ToList() }; // ㉞ 資料庫的工具 → MCP 的 Tool
}

private static Tool ToProtocolTool(McpToolDto t)
{
    using var schema = JsonDocument.Parse(t.InputSchemaJson);
    return new Tool
    {
        Name = t.ToolName,
        Title = t.Title,
        Description = t.Description,
        InputSchema = schema.RootElement.Clone(), // 與核准的版本相同的 JSON Schema
        Annotations = new ToolAnnotations { ReadOnlyHint = t.HttpMethod == "GET" } // GET 是唯讀；POST 會寫入
    };
}
```


清單的查詢在 `EfToolRepository.ListActiveToolsAsync`：`..Where(t => t.ApiVersion.Status == ApiVersionStatus.Published)` 只選已發布版本的工具，依 `ToolName` 排序，最後才投影成 `McpToolDto`（Section 07 踩過：先投影再 `Where`，EF Core 翻譯不成 SQL）。下架不必刪工具列，歷史紀錄的 `ToolName` 仍對得上。

程式說明：

片段	做了什麼	為什麼這樣寫
① <code>tools.Select(ToProtocolTool)</code>	每一筆 <code>McpToolDto</code> 轉成 SDK 的 <code>Tool</code>	Core 層不引用 MCP SDK，轉換放在 Web 層
<code>JsonDocument.Parse(...) + Clone()</code>	把 Schema 字串解析成 <code>JsonElement</code>	<code>Tool.InputSchema</code> 的型別是 <code>JsonElement</code> ； <code>using</code> 會釋放文件，所以要 <code>Clone()</code> 一份獨立的副本
<code>InputSchema = ...</code>	SDK 在這裡檢查最外層是不是 <code>"type": "object"</code>	不合格會丟 <code>ArgumentException</code> ，整個 <code>tools/list</code> 失敗；步驟一的 <code>IsObjectSchema</code> 就是為了不讓這種資料進來
<code>ReadOnlyHint = t.HttpMethod == "GET"</code>	標示唯讀工具	Client 可以據此決定寫入型工具（ <code>ticket_create</code> ）要不要先問使用者

`FindActiveToolAsync`（`CallToolAsync` 用來找工具）直接在這份清單裡找名稱，所以「清單裡看不到的工具」一定也呼叫不到。

最後是 `tools/call`。`CallToolAsync` 同樣在 `McpToolHandlers.cs`，由 SDK 在收到 `tools/call` 時呼叫（上面的 ④ 註冊）：

```
// 範例/09_MCPServer與Tools映射/McpPlatform/src/McpPlatform.Web/Mcp/McpToolHandlers.cs (CallToolAsync 前半)
public static async ValueTask<CallToolResult> CallToolAsync(RequestContext<CallToolRequestParams> context, CancellationToken ct)
{
    var name = context.Params?.Name ?? "";
    var catalog = context.Services!.GetRequiredService<ToolCatalogService>();
    var tool = await catalog.FindActiveToolAsync(name, ct)
        ?? throw new McpProtocolException($"沒有名為 {name} 的工具。", McpErrorCode.InvalidParams); // ㊦ 協定錯誤：JSON-RPC -32602

    var userId = context.User?.FindFirstValue(ClaimTypes.NameIdentifier)
        ?? throw new McpProtocolException("未驗證的呼叫。", McpErrorCode.InvalidRequest);
    var argumentsJson = JsonSerializer.Serialize(context.Params!.Arguments ?? new Dictionary<string, JsonElement>());
```

後半段把呼叫交給 Gateway，並把結果包成 MCP 的 `CallToolResult`：

```
// 範例/09_MCPServer與Tools映射/McpPlatform/src/McpPlatform.Web/Mcp/McpToolHandlers.cs (CallToolAsync 後半)
var gateway = context.Services!.GetRequiredService<ApiInvocationService>();
var result = await gateway.InvokeAsync(new InvocationRequest( // ㊦ 與測試台同一條路
    tool.VersionId, argumentsJson, userId,
    ClientId: context.User?.FindFirstValue(DevTokenAuthenticationHandler.ClientIdClaim),
    ToolName: tool.ToolName), ct);

return new CallToolResult
{
    IsError = !result.Succeeded, // ㊦ 工具錯誤：結果裡的 isError
    Content =
    [
        new TextContentBlock { Text = result.Succeeded ? result.ResponseBody ?? "" : $"[{result.ErrorCode}] {result.Message}" },
        new TextContentBlock { Text = $"CorrelationId: {result.CorrelationId}" } // ㊦ 每個結果都帶追查代碼
    ]
};
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
—	<code>context.Services</code> 取得這次請求的 DI 容器	處理常式是靜態方法，Scoped 服務（DbContext）要從請求的範圍取，不能在啟動時注入
②	找不到工具就丟 <code>McpProtocolException</code> ，代碼 <code>InvalidParams</code>	SDK 把它轉成 JSON-RPC <code>error</code> ， <code>code: -32602</code> ；訊息原樣送給 Client
—	<code>context.User</code> 取出會員 Id	這是 <code>DevTokenAuthenticationHandler</code> 驗證 Token 後建立的使用者；少了它就拒絕，不用匿名身分呼叫
—	<code>Arguments</code> 重新序列化 JSON 字串	Gateway 的 <code>InvocationRequest.ArgumentsJson</code> 本來就吃字串，Schema 驗證在 Gateway 裡做
③	交給 <code>ApiInvocationService.InvokeAsync</code> ，並帶 <code>ClientId</code> 、 <code>ToolName</code>	所有檢查與紀錄只在 Gateway 做一次；兩個欄位寫進 <code>ApiCallLogs</code> （Section 08 就預留了）
④	<code>IsError = !result.Succeeded</code>	Gateway 的每一種失敗都是工具錯誤，AI 讀得到錯誤代碼與說明
⑤	第二段文字固定是 <code>CorrelationId</code>	成功與失敗都有，對得上 <code>ApiCallLogs.CorrelationId</code>

預期結果：帶 Token 送 `tools/list`，回傳步驟一映射的工具，依名稱排序；把其中一個版本下架，下一次 `tools/list` 就少一個；`tools/call` 的三種結果如示範步驟 4。

步驟四 測試用戶端（10 分鐘）

目的：用官方 SDK 寫一個主控台程式，當作「AI 應用」驗收 `/mcp`。

```
dotnet new console -o tools/McpPlatform.McpTestClient
dotnet sln add tools/McpPlatform.McpTestClient
```

版本已寫在 `Directory.Packages.props`（`<PackageVersion Include="ModelContextProtocol" Version="2.2.0" />`），所以在新專案的 csproj 裡手動加一行、**不寫版本**：`<ItemGroup><PackageReference Include="ModelContextProtocol" /></ItemGroup>`。

```
// 範例/09_MCPServer與Tools映射/McpPlatform/tools/McpPlatform.McpTestClient/Program.cs（連線與探索）
var Readable = new JsonSerializerOptions { Encoder = System.Text.Encodings.Web.JavaScriptEncoder.UnsafeRelaxedJsonEscaping }; // 中文照原樣印出
var token = args.Length > 0 ? args[0] : Environment.GetEnvironmentVariable("MCP_DEV_TOKEN") ?? "";
var endpoint = new Uri(args.Length > 1 ? args[1] : "http://localhost:5199/mcp");

var transport = new HttpClientTransport(new HttpClientTransportOptions // ① Streamable HTTP 傳輸
{
    Endpoint = endpoint,
    Name = "McpPlatform 課堂測試用戶端",
    AdditionalHeaders = new Dictionary<string, string> { ["Authorization"] = $"Bearer {token}" } // ② Section 09 的開發用 Token
});

await using var client = await McpClient.CreateAsync(transport); // ③ initialize 握手
Console.WriteLine($"已連線：{client.ServerInfo.Name} {client.ServerInfo.Version}");

var tools = await client.ListToolsAsync(); // ④ tools/list
Console.WriteLine($"n== tools/list：共 {tools.Count} 個工具");
foreach (var tool in tools)
    Console.WriteLine($" {tool.Name,-18} {tool.Description}");

// 範例/09_MCPServer與Tools映射/McpPlatform/tools/McpPlatform.McpTestClient/Program.cs（三種呼叫）
await CallAsync("products_search", new() { ["keyword"] = "茶" }); // ⑤ tools/call
await CallAsync("courses_search", new() { ["level"] = "高手" }); // 參數不符合 Schema → 工具錯誤
await CallAsync("no_such_tool", new()); // 不存在的工具 → 協定錯誤

async Task CallAsync(string name, Dictionary<string, object?> arguments)
{
    Console.WriteLine($"n== tools/call {name} {JsonSerializer.Serialize(arguments, Readable)}");
    try
    {
        var result = await client.CallToolAsync(name, arguments);
        Console.WriteLine($" isError = {result.IsError ?? false}");
        foreach (var text in result.Content.OfType<TextContentBlock>())
            Console.WriteLine($" {Truncate(text.Text)}");
    }
    catch (McpProtocolException ex) // ⑥ JSON-RPC 錯誤回應
    {
        Console.WriteLine($" 協定錯誤 {(int)ex.ErrorCode} ({ex.ErrorCode})：{ex.Message}");
    }
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
—	<code>Readable</code> 使用 <code>UnsafeRelaxedJsonEscaping</code>	預設序列化會把「茶」印成 <code>茶</code> ；這只用在主控台顯示，不送給伺服器
①	<code>HttpClientTransport</code> 指定端點	SDK 會用 Streamable HTTP 送 POST
②	<code>AdditionalHeaders</code> 加上 <code>Authorization</code>	每個請求都帶 Token；少了它 <code>CreateAsync</code> 就失敗在 401
③	<code>McpClient.CreateAsync</code> 完成 <code>initialize</code>	回傳的 <code>ServerInfo</code> 就是伺服器的組件名稱與版本
④	<code>ListToolsAsync</code>	回傳的每個工具都有 <code>Name</code> 、 <code>Description</code> ，與 AI 看到的一樣
⑤	<code>CallToolAsync(name, arguments)</code>	參數用字典，SDK 序列化化成 <code>arguments</code> 物件
—	<code>result.IsError ?? false</code>	協定裡 <code>isError</code> 可以省略，省略代表 <code>false</code>
⑥	捕捉 <code>McpProtocolException</code>	協定錯誤不會回 <code>result</code> ，SDK 以例外呈現； <code>ErrorCode</code> 是 <code>-32602</code>

預期結果：與示範步驟 4 的輸出相同（CorrelationId 每次不同）。

檢查點

做到這裡，你應該可以：

- ☐ 映射真的七個測試案例得到正確訊息，拒絕訊息是紅色 ✕
- ☐ 測試用戶端列出工具，工具說明是「API 名稱（版本）：說明」
- ☐ `products_search` 成功、`courses_search` 高手 `isError = True`、`no_such_tool` 得到 `-32602`
- ☐ 成功與工具錯誤都印出 `CorrelationId`，且在 `ApiCallLogs` 查得到，`ClientId` 是 `classroom-mcp-client`
- ☐ 帶錯 Token 得到 401；下架一個版本後，它的工具從 `tools/list` 消失

常見錯誤

以下都是撰寫本單元時實際遇到的原文。

症狀（錯誤訊息原文）	原因	處理
測試用戶端： <code>McpProtocolException: Request failed (remote): An error occurred.</code> （在 <code>ListToolsAsync</code> 那一行）	伺服器端處理常式丟出例外，SDK 只把籠統的訊息回給 Client，不外洩細節	看網站的主控台輸出，找 <code>method 'tools/list' request handler failed</code> 那一段的例外
網站輸出： <code>System.ArgumentException: The specified document is not a valid MCP tool input JSON schema. (Parameter 'InputSchema')</code>	某個工具的 Schema 最外層不是 <code>"type": "object"</code> （本例是天氣預報 v2 的 <code>{}</code> ）。只要一筆不合格，整個 <code>tools/list</code> 都失敗	刪掉那一筆 <code>ToolDefinitions</code> ，並在 <code>MapAsync</code> 加上 <code>IsObjectSchema</code> 檢查（步驟一）
帶了正確 Token 仍然每次 401	設定鍵與屬性名稱不同（例如設定寫 <code>Mcp:DevToken</code> ，屬性叫 <code>Token</code> ）， <code>Bind</code> 對不上時不報錯， <code>Options.Token</code> 維持空字串	設定節點用 <code>Mcp:DevAuth</code> ，鍵名 <code>Token</code> 、 <code>UserId</code> 、 <code>ClientId</code> ；User Secrets 用 <code>Mcp:DevAuth:Token</code>
測試用戶端： <code>HttpRequestException: Response status code does not indicate success: 401 (Unauthorized).</code> ，內層另有一個 <code>404 (Not Found)</code>	Token 錯誤或沒帶，看外層的 401。內層的 404 來自 <code>GET /mcp</code> ：無狀態模式不提供 GET 串流，直接 GET 也是 404	檢查 Token 是否與 User Secrets 相同；內層 404 不必處理

單元驗收（15 分鐘）

驗收標準：測試 Client 能探索並呼叫示範工具；輸入格式錯誤得到適當協定錯誤或工具錯誤；Tool 結果保留可追查識別碼。

驗收方式：

1. **能探索：**測試用戶端 `tools/list` 列出映射過的工具（實測 4 個），每個都有名稱與「API 名稱（版本）：說明」。
2. **能呼叫：**`products_search {"keyword":"茶"}` → `isError = False`，內容是 3 筆茶類商品。
3. **輸入格式錯誤 → 工具錯誤：**`courses_search {"level":"高手"}` → `isError = True`、`[INVALID_ARGUMENTS]` 參數不符合規格：`/level enum...`；上游沒有收到請求。
4. **輸入格式錯誤 → 協定錯誤：**呼叫不存在的 `no_such_tool` → `-32602 (InvalidParams)`：沒有名為 `no_such_tool` 的工具。
5. **保留可追查識別碼：**第 2、3 項的結果都有 `CorrelationId: ...`，在 `ApiCallLogs` 查得到同一串，且 `UserId = member-a`、`ClientId = classroom-mcp-client`、`ToolName` 是工具名稱。
6. **反向驗證：**帶錯 Token → 401，連 `initialize` 都做不了；把郵遞區號查詢 v1 下架 → `tools/list` 剩 3 個，`zip_lookup` 呼叫得到 `-32602`；後臺映射 `Products Search`（名稱不合規則）、天氣預報 v2（Schema 不合規定）、重複的 `products_search` 都被拒絕。

本單元結束時，郵遞區號查詢 v1 已經下架。

回顧：`/mcp` 只是一個翻譯層：`tools/list` 從資料庫讀出已發布版本的工具，`tools/call` 把工具名稱換成版本 Id 交給 Gateway。找不到工具是協定錯誤，Gateway 的失敗是工具錯誤，兩種都讓 AI 看得懂；每次呼叫都能用 `CorrelationId` 追到紀錄。下一單元（Section 10）把固定 Token 換成 OAuth：真實的 MCP Client 登入平台會員、取得有期限與 Audience 的 Access Token。

延伸閱讀

- [MCP 規範：Tools \(2025-11-25\)](#)：`tools/list`、`tools/call`、`isError` 與協定錯誤的區分。
- [MCP 規範：Transports \(2025-11-25\)](#)：Streamable HTTP 的請求與回應格式。
- [MCP C# SDK \(GitHub\)](#)：`ModelContextProtocol`、`ModelContextProtocol.AspNetCore` 的原始碼與範例。
- [JSON-RPC 2.0 規格](#)：`-32602 Invalid params` 等標準錯誤代碼。

第 10 章

ConnectorOAuth與會員授權

網站 Cookie 與 Access Token 的差異；PKCE、Redirect URI、Scope、Audience 及資源中繼資料探索；套用 OpenIdDict 授權服務骨架、登錄測試 Client、以 Issuer + Subject 對應會員



課程時數 2 小時 (120 分鐘) 當堂成果 真實 Client 連線及身分對應 範例專案 範例/10_ConnectorOAuth與會員授權

學習目標

上完這個單元，你能夠：

- 說明為什麼 `/mcp` 不能用網站的登入 Cookie，而要用 OAuth 簽發、有期限與 Audience 的 Access Token。
- 在網站裡套用教師提供的 OpenIdDict 授權服務骨架：授權碼流程 + PKCE、`/connect/authorize` 同意頁、`/connect/token`，並在後臺登錄一個測試 Client（Redirect URI、公開用戶端）。
- 讓 `/mcp` 提供資源中繼資料（`/.well-known/oauth-protected-resource`），並驗證 Token 的簽章、Issuer、Audience 與期限。
- 用 Token 裡的 **Issuer + Subject** 查 `ExternalIdentityLinks` 對應到平台會員，而不是相信 Client 傳來的任何 UserId。
- 用官方 SDK 的測試用戶端完成「探索 → 登入 → 同意 → 換 Token → 列出並呼叫工具」，並實測未登入、過期 Token、錯誤 Audience 都被拒絕。

先備知識

- 完成 Section 09：`/mcp` 能列出並呼叫工具，但只認一組固定的開發用 Token（本單元把它拿掉，換成 OAuth）。
- Section 04 的 Identity 登入（Cookie `McpPlatform.Auth`）與示範會員 `a@demo.test`（Id `member-a`）。
- Section 03 已建好但一直是空的兩張表：`ExternalIdentityLinks`（`Issuer + Subject` 唯一）、`ConnectorClients`。
- 知道 HTTP 302 轉址、查詢字串；知道 JWT 是「用點分成三段的 Base64 字串」即可。

觀念說明（20 分鐘）

網站 Cookie 與 Access Token

會員用瀏覽器操作網站時，登入後瀏覽器保存 `McpPlatform.Auth` Cookie，之後每個請求自動帶上。AI 應用（Connector）不是這個瀏覽器，它需要的是一張可以交給程式、而且只能用在 `/mcp` 的通行證：Access Token。

比較	網站登入 Cookie	MCP Access Token
誰拿著	會員的瀏覽器	Connector（AI 應用、測試用戶端）
怎麼取得	在登入頁輸入帳密	會員在同意頁按「允許」後，Connector 用授權碼換取
能用在哪裡	整個網站	只有 <code>aud</code> 寫的資源（本平台 <code>http://localhost:5199/mcp</code> ）
有效期	8 小時，操作中會延長	10 分鐘（ <code>OAuth:AccessTokenMinutes</code> ），到期就失效
Connector 知道會員密碼嗎	—	不知道：密碼只輸入在平台自己的登入頁

本單元實測：以 `a@demo.test` 登入網站後，帶著那個 Cookie 對 `/mcp` 送 `tools/list`，得到 `401`。`/mcp` 只認 Access Token。

整個流程：從 401 到拿到 Token

測試用戶端第一次連 `/mcp` 時手上沒有 Token。下圖是它自動完成的交談（官方 SDK 幫你做了大部分）：

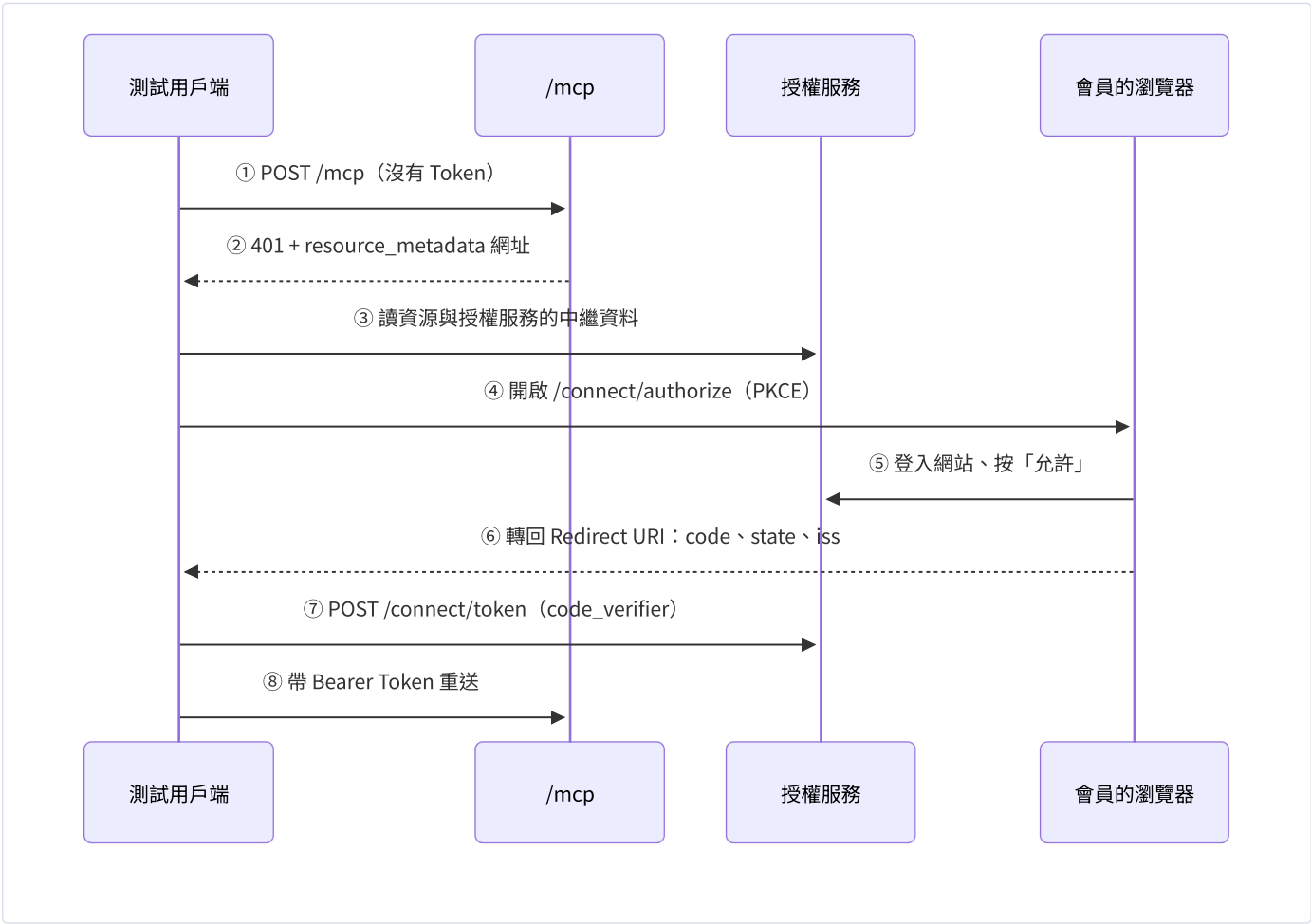


圖 10-1 測試用戶端取得 Access Token 的交談順序

編號	送出／拿回什麼	本單元實測
① ②	沒有 Token 被拒；回應標頭告訴 Client 去哪裡讀資源說明	401、WWW-Authenticate: Bearer resource_metadata="http://localhost:5199/.well-known/oauth-protected-resource/mcp"
③	資源中繼資料說「我是 http://localhost:5199/mcp，Token 找 http://localhost:5199/ 要」；再讀授權服務的 /.well-known/oauth-authorization-server 得到兩個端點	見教師示範步驟 2
④	授權網址帶 client_id、redirect_uri、scope=mcp:tools、resource=.../mcp、code_challenge (S256)、state	見教師示範步驟 3
⑤	會員在平台自己的頁面登入並同意；Connector 看不到密碼	圖 10-4
⑥	瀏覽器被轉回 http://127.0.0.1:7071/callback?code=...&state=...&iss=http://localhost:5199/	測試用戶端的 HttpListener 接住
⑦	用授權碼 + 當初的 code_verifier 換 Token	200，expires_in: 599
⑧	之後每個請求都帶 Authorization: Bearer ...	tools/list 回 3 個工具

PKCE、Redirect URI、Scope、Audience、資源中繼資料

名詞	解決什麼問題	本平台的設定
Redirect URI	授權碼要送到哪裡。只送到登錄過、一字不差的網址，攻擊者不能把授權碼導去自己的網站	http://127.0.0.1:7071/callback；只有本機迴路位址可以用 http，其餘一律 https
PKCE	測試用戶端是公開用戶端（命令列程式存不住 Client Secret）。它先產生隨機的 code_verifier，只送出它的 SHA-256 雜湊 code_challenge；換 Token 時才交出原文。半路偷到授權碼的人沒有原文，換不到 Token	RequireProofKeyForCodeExchange()，每個 Client 也標記必須 PKCE
Scope	Connector 要求的權限範圍，會列在同意頁上給會員看	mcp:tools（呼叫 MCP 工具）；reports:read 只用來示範另一個資源
Audience（aud）	Token 是給哪個資源用的。給報表服務的 Token 拿來呼叫 /mcp 必須被拒絕	Client 用 resource 參數指定（RFC 8707）；/mcp 只收 aud = http://localhost:5199/mcp
資源中繼資料	Client 不必事先知道授權服務在哪裡，從 401 的標頭一路找得到（RFC 9728）	SDK 的 AddMcp() 提供 /.well-known/oauth-protected-resource
state / iss	state 防止別人把他的授權結果塞給你；iss 確認授權碼來自預期的授權服務（RFC 9207）	SDK 比對兩者都相符才去換 Token

Access Token 裡有什麼

本平台的 Token 是只簽章、不加密的 JWT，中間那段解開後是（本單元實測）：

```
{
  "iss": "http://localhost:5199/",
  "exp": 1790426538,
  "iat": 1790425938,
  "aud": "http://localhost:5199/mcp",
  "scope": "mcp:tools",
  "sub": "member-a",
  "name": "a@demo.test",
  "client_id": "classroom-mcp-client",
  "jti": "..."
}
```

宣告	意思	/mcp 怎麼用
iss	誰簽的	必須是設定的 Issuer，且與 sub 一起查會員
aud	給哪個資源用	必須是 http://localhost:5199/mcp
exp、iat	到期、簽發時間（Unix 秒）	過期就 401；兩者相差 600 秒 = 10 分鐘
sub	簽給誰（授權服務眼中的使用者）	與 iss 一起查 ExternalIdentityLinks
client_id	哪一個 Connector	寫進 ApiCallLogs.ClientId
scope	會員同意的權限	記錄用；Section 11 才依 Scope 限制工具

簽章讓任何人都改不了內容：把 sub 改成別人，簽章就對不上，/mcp 直接 401。

Issuer + Subject 對應會員

`sub = member-a` 看起來就是會員 Id，為什麼還要查表？因為 `sub` 只在簽發它的授權服務裡有意義。將來若接入學校或公司的登入服務，別的授權服務也可能簽出 `sub = member-a`，那是另一個人。所以平台只用 `(iss, sub)` 這一對去查：

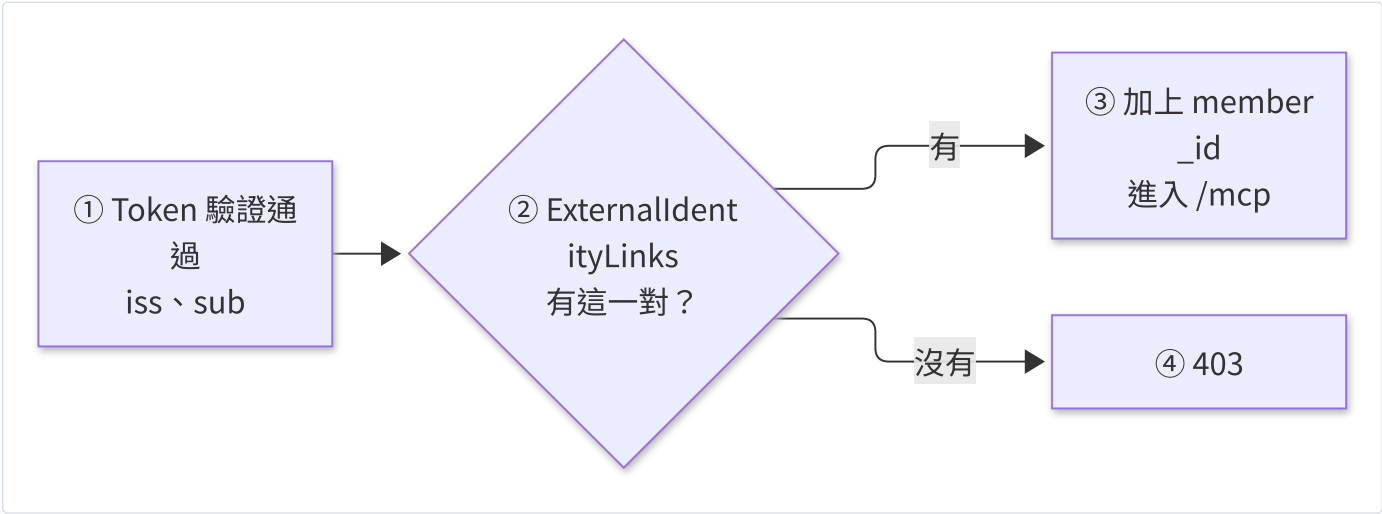


圖 10-2 Issuer + Subject 對應會員

編號	做了什麼	對應程式碼
①	OpenIddict 驗證簽章、Issuer、Audience、期限；任一不符直接 401，不會走到 ②	<code>AddValidation(...)</code>
②	以 <code>(iss, sub)</code> 查對應表	<code>McpMemberClaimsTransformation</code>
③	查到的會員 Id 成為 <code>member_id</code> 宣告； <code>tools/call</code> 用它當呼叫者	<code>McpToolHandlers.CallToolAsync</code>
④	Token 合法但對應不到會員	<code>/mcp</code> 授權原則 <code>RequireClaim(member_id)</code>

對應是在同意頁按「允許」時建立的：那一刻會員剛用網站帳密登入，平台確定「這個 `sub` 就是這位會員」。

限制與注意事項

- 雲端 Connector 連不到你電腦的 localhost。本單元用本機的測試用戶端；要給雲端 AI 服務用，必須有外部連得到的 https 網址（Section 15 部署）。資料庫仍可維持網站主機上的 localhost。
- 本機用 http：`DisableTransportSecurityRequirement()` 只為了課堂；部署一律 https。簽章用的是 OpenIddict 開發憑證（存在目前 Windows 使用者的憑證存放區），部署要換正式憑證。
- 沒有 Refresh Token：Token 10 分鐘到期後，Client 要重新走一次授權。
- 不開放動態登錄（DCR）：Client 只能由管理者在後臺登錄。
- 本單元的 `tools/list` 還沒依會員的 Grant 過濾，`tools/call` 也還不檢查 Grant（Section 11）；Connector 的停用（`IsEnabled`）在 Section 12 處理。

名詞對照

名詞	白話說明	在本課程中是
授權服務	負責登入同意、簽發 Token 的服務	OpenIddict，與網站同一個程式
資源伺服器	收 Token、提供功能的一方	<code>/mcp</code>
Connector／Client	代表會員呼叫 <code>/mcp</code> 的程式	測試用戶端 <code>classroom-mcp-client</code>
公開用戶端	無法安全保存密碼（Client Secret）的程式	命令列、桌面程式；必須 PKCE
授權碼	同意後拿到的一次性代碼，只能換一次 Token	<code>code</code>
Access Token	呼叫資源用的通行證	10 分鐘的 JWT
Issuer + Subject	「誰簽的」+「簽給誰」，合起來才唯一	<code>ExternalIdentityLinks</code> 的唯一鍵

教師示範（25 分鐘）

示範情境：在後臺登錄測試用戶端；測試用戶端自動找到授權服務，會員在瀏覽器登入並按「允許」，然後列出並呼叫工具；最後逐一示範未登入、錯誤 Audience、過期、對應不到會員都被拒絕。

示範步驟

1. 在 `範例/10_ConnectorOAuth與會員授權/McpPlatform` 套用 Migration（`dotnet ef database update --project src/McpPlatform.Infrastructure --startup-project src/McpPlatform.Web`），啟動示範 API 與網站。以管理者登入，側欄「Connector 登錄」，依序送出四組資料。本單元實測訊息：

Client Id	Redirect URI	結果
Classroom Client	<code>http://127.0.0.1:7071/callback</code>	✗ Client Id 只能用小寫英文、數字、連字號，以英文字母開頭，3~64 字...
<code>classroom-mcp-client</code>	<code>http://example.com/callback</code>	✗ Redirect URI 必須是完整網址：一律 <code>https</code> ；只有本機迴路位址（ <code>127.0.0.1</code> 、 <code>localhost</code> ）可以用 <code>http</code> ...
<code>classroom-mcp-client</code>	<code>http://127.0.0.1:7071/callback</code>	✓ 已登錄 課堂 MCP 測試用戶端（ <code>classroom-mcp-client</code> ），Redirect URI： <code>http://127.0.0.1:7071/callback</code>
<code>classroom-mcp-client</code>	<code>http://127.0.0.1:7071/callback</code>	✗ Client Id <code>classroom-mcp-client</code> 已經登錄過。

MCP Server API 平台

API 目錄 會員中心 管理後臺

系統管理員 登出

管理後臺

總覽

API 管理

資料庫概況

上架審核

使用申請審核

Grant 管理

API 測試台

MCP 工具映射

Connector 登錄

會員管理 (Section 12)

Connector 登錄

MCP Client 要先在這裡登錄，才能向平台要求授權。登錄後的用戶端是公開用戶端（沒有 Client Secret），一定要用 PKCE；授權碼只會送到登錄的 Redirect URI。

Client 端要填的設定

MCP 端點 (資源)

授權服務 (Issuer)

授權端點

Token 端點

Scope

http://localhost:5199/mcp

http://localhost:5199/

http://localhost:5199/connect/authorize

http://localhost:5199/connect/token

mcp:tools

Client Id

名稱

Redirect URI

狀態

classroom-mcp-client

課堂 MCP 測試用戶端

http://127.0.0.1:7071/callback

啟用

登錄新的 Connector

Client Id

名稱 (同意頁上顯示)

Redirect URI

classroom-mcp-client

課堂 MCP 測試用戶端

http://127.0.0.1:7071/callback

登錄

© 2026 MCP Server API 平台 | 環境檢查

圖 10-3 後臺「Connector 登錄」頁

畫面重點：

1. ①「Client 端要填的設定」：MCP 端點、Issuer、授權端點、Token 端點、Scope，全部來自 `appsettings.json` 的 `OAuth` 節點。Client 程式裡只需要寫 MCP 端點、Client Id 與 Redirect URI；授權服務與兩個端點它會自己探索到。
2. ②已登錄的 Client：名稱「課堂 MCP 測試用戶端」會顯示在同意頁；Redirect URI 必須和 Client 程式裡寫的一字不差。
3. ③登錄表單：只有三個欄位。沒有 Client Secret：登錄的一律是公開用戶端，一律 PKCE。
4. 看探索資訊（不帶 Token 對 `/mcp` 送 POST，再讀資源中繼資料）：

```
POST /mcp → 401
WWW-Authenticate: Bearer resource_metadata="http://localhost:5199/.well-known/oauth-protected-resource/mcp"

GET /.well-known/oauth-protected-resource/mcp → 200
{"resource": "http://localhost:5199/mcp", "authorization_servers": ["http://localhost:5199/"],
 "bearer_methods_supported": ["header"], "scopes_supported": ["mcp:tools"], "resource_name": "MCP Server API 平台"}

GET /.well-known/oauth-authorization-server → 200
{"issuer": "http://localhost:5199/", "authorization_endpoint": "http://localhost:5199/connect/authorize",
 "token_endpoint": "http://localhost:5199/connect/token", ...}
```

3. 執行測試用戶端（不帶參數＝瀏覽器模式）：`dotnet run --project tools/McpPlatform.McpTestClient`。它先印出授權網址並開啟瀏覽器（實測網址，節錄）：

```
請在瀏覽器登入並按「允許」：
http://localhost:5199/connect/authorize?client_id=classroom-mcp-client&redirect_uri=http%3a%2f%2f127.0.0.1%3a7071%2fcallback
&response_type=code&code_challenge=s-lfDjSA...&code_challenge_method=S256&state=EWTSAxmh...&resource=http%3a%2f%2flocalhost%3a5199%2fmcp&scope=mc
p%3atools
```

瀏覽器先被轉到登入頁；以 `a@demo.test` 登入後回到同意頁：



圖 10-4 同意頁：會員決定是否讓 Connector 代表自己

畫面重點：

- ①**Connector 名稱與會員帳號**：「課堂 MCP 測試用戶端 要求以你的會員身分（a@demo.test）…」，會員清楚知道是誰、以誰的身分。
- ②**權限清單**：`mcp:tools`，就是授權網址的 `scope`。
- ③**說明文字**：Token 有限期；Connector 仍只能用管理者核准給會員的 API（Section 11 落實）。
- ④**「允許」「拒絕」**：按「允許」後瀏覽器被轉到 `http://127.0.0.1:7071/callback`，頁面顯示「授權完成，可以關閉這個分頁。」
5. 回到主控台，測試用戶端已換到 Token 並繼續（實測）：

已連線：McpPlatform.Web 1.0.0.0

```
== tools/list: 共 3 個工具
  courses_search    課程查詢 (v1): 依程度篩選課程, 回傳課程代碼與時數。
  products_search   商品查詢 (v1): 依關鍵字查詢商品名稱、分類、價格與庫存。
  ticket_create     服務工單建立 (v1): 建立一張服務工單; 寫入型, 需寫等鍵與確認旗標。

== tools/call products_search {"keyword": "茶"}
  isError = False
  {"count": 3, "items": [{"sku": "P-1001", "name": "阿里山高山烏龍茶 150g", ...
  CorrelationId: 13fc2825b66e4762
```

5. 用 SSMS 看身分對應與呼叫紀錄：

```
-- 在 MCPServerManag 執行; 只讀
SELECT UserId, Issuer, Subject, LinkedAtUtc FROM dbo.ExternalIdentityLinks;
SELECT TOP 2 UserId, ClientId, ToolName, Outcome, CorrelationId FROM dbo.ApiCallLogs ORDER BY Id DESC;
```

```
member-a|http://localhost:5199/|member-a|2026-09-26 12:34:30.2809550 +00:00
member-a|classroom-mcp-client|courses_search|InvalidInput|c23d63387b1d4626
member-a|classroom-mcp-client|products_search|Success|910aaa97938c4527
```

6. 反向示範。範例的 `tools/oauth-flow.ps1` 不經過 SDK，一步一步走授權流程並印出每一步的結果（密碼放環境變數 `MCP_LOGIN_PASSWORD`）。本單元實測：

情況	結果
未登入就開授權網址	302 → /Account/Login (登入後回到同意頁)
同意頁按「拒絕」	302 → .../callback?error=access_denied&error_description=會員拒絕授權。
resource=http://localhost:5199/reports	拿到 aud=.../reports 的 Token; 呼叫 /mcp → 401
resource=http://evil.test/mcp	400 invalid_target: One of the specified 'resource' parameters is invalid.
Token 到期 (AccessTokenMinutes=1)	0 秒時 200; 62 秒時 401
刪除 ExternalIdentityLinks 那一列	同一張 Token → 403; 重新同意後恢復 200
帶網站登入 Cookie、不帶 Token	401

網站主控台會寫出被拒的原因：過期是 `The authentication demand was rejected because the token was expired.`，Audience 不符是 `The token was rejected because it had no valid audience.`。

學員實作（60 分鐘）

範例專案：範例/10_ConnectorOAuth與會員授權/McpPlatform/（從 Section 09 複製）。本單元新增：Core 的 `Connectors/`（`ConnectorService.cs`、`IdentityLinkService.cs`）；Infrastructure 的 `Connectors/`（`OpenIddictConnectorRegistry.cs`、`EfIdentityLinkRepository.cs`、`OAuthScopeSeeder.cs`）與 Migration `AddOpenIddict`；Web 的 `AuthorizationController`、同意頁、後臺 `ConnectorsController`、`Mcp/McpMemberClaimsTransformation.cs`、`Options/OAuthOptions.cs`；測試用戶端的 `BrowserLogin.cs`、`ScriptedLogin.cs`；`tools/oauth-flow.ps1`。刪除 Section 09 的 `DevTokenAuthenticationHandler.cs` 與 `Mcp:DevAuth` 設定。

步驟一 套用 OpenIddict 骨架與 Migration（15 分鐘）

目的：讓網站同時是授權服務。OpenIddict 管 Client、授權碼、Token 這些協定資料，我們不自己寫。

`Directory.Packages.props` 加入 `OpenIddict.AspNetCore`、`OpenIddict.EntityFrameworkCore`（都是 7.7.1）；Infrastructure 參考後者、Web 參考前者。`AppDbContext.OnModelCreating` 加一行，並讓「不串聯刪除」的規則略過 OpenIddict 的表：

```
// 範例/10_ConnectorOAuth與會員授權/McpPlatform/src/McpPlatform.Infrastructure/Data/AppDbContext.cs (OnModelCreating 節錄)
builder.UseOpenIddict(); // Section 10: OpenIddict 的四張協定資料表

// 所有外鍵都不串聯刪除：下架、停權都不應該連帶刪掉歷史紀錄。
// OpenIddict 的資料表除外：授權碼、Token 由它自己管理與清除，保留它原本的刪除規則。
foreach (var fk in builder.Model.GetEntityTypes()
    .Where(e => e.ClrType.Namespace?.StartsWith("OpenIddict") != true)
    .SelectMany(e => e.GetForeignKeys()))
    fk.DeleteBehavior = DeleteBehavior.Restrict; // ④
```

```
dotnet ef migrations add AddOpenIddict --project src/McpPlatform.Infrastructure --startup-project src/McpPlatform.Web
dotnet ef database update --project src/McpPlatform.Infrastructure --startup-project src/McpPlatform.Web
```

預期結果：Migration 只有四個 `CreateTable`：`OpenIddictApplications`、`OpenIddictScopes`、`OpenIddictAuthorizations`、`OpenIddictTokens`，沒有動到其他表。

接著在 `Program.cs` 註冊授權服務與 Token 驗證：

```
// 範例/10_ConnectorOAuth與會員授權/McpPlatform/src/McpPlatform.Web/Program.cs (OpenIddict)
builder.Services.AddOpenIddict()
    .AddCore(options => options.UseEntityFrameworkCore().UseDbContext<AppDbContext>()) // ① Client、授權碼、Token 存在 OpenIddict 的資料表
    .AddServer(options =>
    {
        options.SetIssuer(new Uri(oauth.Issuer)); // ② Token 的 iss，固定寫死，不從請求的網址推算
        options.SetAuthorizationEndpointUri("connect/authorize")
            .SetTokenEndpointUri("connect/token");
        options.AllowAuthorizationCodeFlow()
            .RequireProofKeyForCodeExchange(); // ③ 只允許授權碼 + PKCE
        options.RegisterScopes(OAuthScopes.McpTools, OAuthScopes.ReportsRead);
        options.RegisterResources(oauth.McpResource, oauth.ReportsResource); // Client 可以用 resource 參數指定的資源 (RFC 87
07)
        options.SetAccessTokenLifetime(TimeSpan.FromMinutes(oauth.AccessTokenMinutes)); // ④ Token 有效期
        options.AddDevelopmentEncryptionCertificate()
            .AddDevelopmentSigningCertificate();
        options.DisableAccessTokenEncryption(); // 開發用憑證；部署時換正式憑證 (Section 15)
        options.UseAspNetCore(); // Token 只簽章不加密，課堂上可以解開來看
            .EnableAuthorizationEndpointPassthrough() // ⑤ 授權端點交給 AuthorizationController
            .DisableTransportSecurityRequirement(); // 本機用 http；部署一律 https
    })
    .AddValidation(options =>
    {
        options.UseLocalServer(); // ⑥ 同一個網站簽的 Token，直接用本機的金鑰驗證
        options.AddAudiences(oauth.McpResource); // ⑦ aud 必須是 /mcp 的網址
        options.UseAspNetCore();
    });
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	用 EF Core 存協定資料，DbContext 就是平台的 AppDbContext	同一個資料庫、同一個交易，登錄 Client 時能和 ConnectorClients 一起寫入
②	Issuer 從設定讀，不讓 OpenIddict 依請求網址推算	ExternalIdentityLinks.Issuer 要與 Token 的 iss 一字不差；少一個結尾斜線就對應不到會員
③	只開授權碼流程，且強制 PKCE	其他流程（例如直接送帳密）在 MCP 不需要，不開就不會被濫用
—	RegisterResources	官方 SDK 一定會送 resource 參數；沒登錄的資源一律 invalid_target （見〈常見錯誤〉）
④	Token 10 分鐘	洩漏時的損害有時間上限
⑤	授權端點由我們的 Controller 決定「誰、同不同意」	Token 端點沒有開 passthrough：授權碼換 Token 完全由 OpenIddict 處理，也不受全站防偽權杖的影響
⑥ ⑦	驗證簽章、Issuer、期限，且 aud 必須是 /mcp	給其他資源的 Token 在這裡就被擋掉

appsettings.json 的 **OAuth** 節點：

鍵	值	設錯的後果
Issuer	http://localhost:5199/ （結尾有斜線）	與網站網址不符：Client 探索到的授權服務連不到
McpResource	http://localhost:5199/mcp	與 Client 連線的網址不符：SDK 送的 resource 被拒，或 Token 的 aud 對不上而 401
ReportsResource	http://localhost:5199/reports	只用於示範錯誤 Audience
AccessTokenMinutes	10	設 1 可以在課堂上測過期

步驟二 登錄測試 Client（10 分鐘）

目的：後臺表單檢查規則後，同時寫入 OpenIddict 的用戶端登錄與 **ConnectorClients**。

```
// 範例/10_ConnectorOAuth與會員授權/McpPlatform/src/McpPlatform.Core/Connectors/ConnectorService.cs (Redirect URI 規則)
/// <summary>授權碼會送到 Redirect URI；http 只允許送回使用者自己的電腦（命令列、桌面程式的標準做法）。</summary>
private static bool IsAllowedRedirect(Uri uri) =>
    string.IsNullOrEmpty(uri.Fragment)
    && (uri.Scheme == Uri.UriSchemeHttps || (uri.Scheme == Uri.UriSchemeHttp && uri.IsLoopback));

// 範例/10_ConnectorOAuth與會員授權/McpPlatform/src/McpPlatform.Infrastructure/Connectors/OpenIddictConnectorRegistry.cs (RegisterAsync 前半)
var descriptor = new OpenIddictApplicationDescriptor
{
    ClientId = registration.ClientId,
    DisplayName = registration.Name,
    ClientType = ClientTypes.Public,
    ConsentType = ConsentTypes.Explicit,
    RedirectUri = { registration.RedirectUri },
    Permissions =
    {
        Permissions.Endpoints.Authorization,
        Permissions.Endpoints.Token,
        Permissions.GrantTypes.AuthorizationCode,
        Permissions.ResponseTypes.Code,
        Permissions.Prefixes.Scope + OAuthScopes.McpTools,
        Permissions.Prefixes.Scope + OAuthScopes.ReportsRead
    },
    Requirements = { Requirements.Features.ProofKeyForCodeExchange }
};
foreach (var resource in resources)
    descriptor.Permissions.Add(Permissions.Prefixes.Resource + resource);

await using var transaction = await db.Database.BeginTransactionAsync(ct);
await applications.CreateAsync(descriptor, ct);
```

// 同意頁上給會員看的名稱

// ④ 公開用戶端：沒有 Client Secret

// ⑤ 每次都要會員按「允許」

// ⑥ 授權碼只會送到這個網址

// ⑦ 一定要用 PKCE

// ⑧ 允許用 resource 參數要求這些資源

// OpenIddict 用同一個 AppDbContext 寫入

程式說明：

標號	做了什麼	為什麼這樣寫
<code>IsAllowedRedirect</code>	https 一律可以；http 只限 <code>uri.IsLoopback</code> （ <code>127.0.0.1</code> 、 <code>localhost</code> ）；不可有 <code>#</code>	授權碼放在網址上，送到別人的 http 網站會被看見
①	<code>ClientTypes.Public</code>	命令列程式的檔案誰都拿得到，放 <code>Secret</code> 等於公開
②	<code>ConsentTypes.Explicit</code>	不會因為「以前同意過」就自動發 Token
③	登錄的 Redirect URI	OpenIddict 比對請求的 <code>redirect_uri</code> ，不同就拒絕，不轉址
—	<code>Permissions</code> 清單	這個 Client 只能用授權端點、Token 端點、授權碼流程與兩個 Scope；清單外的一律拒絕
④ ⑤	必須 PKCE；允許要求的資源	資料表中存成 <code>rsrc:http://localhost:5199/mcp</code> 這樣的權限字串
<code>transaction</code>	後半段再寫 <code>ConnectorClients</code> 與 <code>AuditLogs</code> 後一起 Commit	兩邊資料不會一邊有、一邊沒有

預期結果：示範步驟 1 的四個訊息；`OpenIddictApplications` 有一列 `classroom-mcp-client`，`ClientType = public`。

步驟三 授權端點與同意頁（15 分鐘）

目的：OpenIddict 驗完 Client、Redirect URI、PKCE、Scope、resource 之後，把請求交給 `AuthorizationController`；我們只決定「會員是誰、同不同意」。

GET `/connect/authorize` 沒登入就轉到登入頁（`Challenge`，登入後回到原網址），登入了就顯示同意頁。按下按鈕是 POST：

```
// 範例/10_ConnectorOAuth與會員授權/McpPlatform/src/McpPlatform.Web/Controllers/AuthorizationController.cs (Decide)
[HttpPost("~/connect/authorize")]
public async Task<IActionResult> Decide(string? decision, CancellationToken ct)
{
    var request = HttpContext.GetOpenIddictServerRequest()
    ?? throw new InvalidOperationException("這不是 OpenIddict 驗證過的授權請求。");
    var user = await users.GetUserAsync(User);
    if (user is null)
        return Challenge(IdentityConstants.ApplicationScheme);

    if (decision != "accept") // ㊦ 拒絕：帶 access_denied 回到 Client
        return Forbid(new AuthenticationProperties(new Dictionary<string, string?>
        {
            [OpenIddictServerAspNetCoreConstants.Properties.Error] = Errors.AccessDenied,
            [OpenIddictServerAspNetCoreConstants.Properties.ErrorDescription] = "會員拒絕授權。"
        }), OpenIddictServerAspNetCoreDefaults.AuthenticationScheme);

    var identity = new ClaimsIdentity(OpenIddictServerAspNetCoreDefaults.AuthenticationScheme, Claims.Name, Claims.Role);
    identity.SetClaim(Claims.Subject, user.Id) // ㊦ sub = 會員 Id
        .SetClaim(Claims.Name, user.UserName);
    identity.SetScopes(request.GetScopes());
    identity.SetResources(request.Resources is { Length: > 0 } requested // ㊦ aud: Client 用 resource 參數指定的資源；
        ? requested : await ResourcesAsync(request.GetScopes(), ct)); // 沒指定才依 Scope 推算
    identity.SetDestinations(_ => [Destinations.AccessToken]);

    await links.LinkAsync(user.Id, oauth.Issuer, user.Id, ct); // ㊦ 記下 (Issuer, Subject) → 會員
    return SignIn(new ClaimsPrincipal(identity), OpenIddictServerAspNetCoreDefaults.AuthenticationScheme); // ㊦ OpenIddict 產生授權碼並轉回 Client
}
```

同意頁的表單把原本的 OAuth 參數原樣送回，OpenIddict 在 POST 時會再驗證一次：

```
@foreach (var parameter in Context.Request.Query)
{
    <input type="hidden" name="@parameter.Key" value="@parameter.Value" />
}

<button type="submit" name="decision" value="accept" class="btn btn-classic">允許</button>
<button type="submit" name="decision" value="deny" class="btn btn-outline-secondary">拒絕</button>
```

程式說明：

標號	做了什麼	為什麼這樣寫
<code>GetOpenIdDictServerRequest()</code>	取得 OpenIdDict 已驗證過的授權請求	走到這裡代表 Client、Redirect URI、PKCE、Scope、resource 都合法
<code>users.GetUserAsync(User)</code>	由網站 Cookie 取得會員	這是唯一用到 Cookie 的地方：證明「按允許的人」是誰
③	<code>Forbid + access_denied</code>	瀏覽器被轉回 Client，網址帶 <code>error=access_denied</code> ，Client 知道會員拒絕了
④	<code>sub</code> 用會員 Id， <code>name</code> 只是顯示用	不用 Email 當 <code>sub</code> ：Email 可以改，Id 不會
⑤	<code>aud</code> 優先用 Client 的 <code>resource</code> 參數	MCP 規範要求 Client 指定資源；舊式 Client 沒指定時才由 Scope 推算（ <code>OAuthScopeSeeder</code> 登錄的對應）
<code>SetDestinations</code>	所有宣告都放進 Access Token	本課程不發 ID Token
⑥	建立（ <code>http://localhost:5199/</code> ， <code>member-a</code> ）→ <code>member-a</code> 的對應	此刻會員剛登入，對應可信；已存在就不重建
⑦	<code>SignIn</code>	OpenIdDict 產生授權碼，302 轉到 Redirect URI，網址帶 <code>code</code> 、 <code>state</code> 、 <code>iss</code>
<code>hidden</code> 欄位	送回 <code>client_id</code> 、 <code>redirect_uri</code> 、 <code>code_challenge</code> …	表單另外還有 <code><form></code> Tag Helper 自動加上的防偽權杖，全站防偽檢查照常運作

預期結果：用瀏覽器開示範步驟 3 的授權網址 → 登入 → 圖 10-4 → 允許後網址變成 `http://127.0.0.1:7071/callback?code=...&state=...&iss=http%3A%2F%2Flocalhost%3A5199%2F`。

步驟四 /mcp：資源中繼資料、Token 驗證、會員對應（10 分鐘）

目的：/mcp 改收 OAuth Token，401 時告訴 Client 去哪裡找授權服務。

```
// 範例/10_ConnectorOAuth與會員授權/McpPlatform/src/McpPlatform.Web/Program.cs（資源中繼資料）
builder.Services.AddAuthentication()
    .AddMcp(options =>
    {
        options.ForwardAuthenticate = OpenIdDictValidationAspNetCoreDefaults.AuthenticationScheme; // ⑧ 驗 Token 交給 OpenIdDict
        options.ResourceMetadata = new ProtectedResourceMetadata // ⑨ /.well-known/oauth-protected-resource
        {
            Resource = oauth.McpResource,
            AuthorizationServers = [oauth.Issuer],
            ScopesSupported = [OAuthScopes.McpTools],
            ResourceName = "MCP Server API 平台"
        };
    });

// 範例/10_ConnectorOAuth與會員授權/McpPlatform/src/McpPlatform.Web/Program.cs（端點）
// /mcp 端點：Token 由 OpenIdDict 驗證（簽章、Issuer、Audience、期限）；沒有或無效 → 401，查不到會員 → 403。
app.MapMcp("/mcp").RequireAuthorization(new AuthorizationPolicyBuilder(McpAuthenticationDefaults.AuthenticationScheme)
    .RequireAuthenticatedUser()
    .RequireClaim(McpClaims.MemberId) // Issuer + Subject 對應得到會員
    .Build());
```

會員對應寫成 `IClaimsTransformation`，在 Token 驗證成功後執行：

```
// 範例/10_ConnectorOAuth與會員授權/McpPlatform/src/McpPlatform.Web/Mcp/McpMemberClaimsTransformation.cs (TransformAsync)
public async Task<ClaimsPrincipal> TransformAsync(ClaimsPrincipal principal)
{
    var issuer = principal.FindFirstValue(Claims.Issuer); // ⑩ 誰簽的
    var subject = principal.FindFirstValue(Claims.Subject); // ⑪ 簽給誰
    if (issuer is null || subject is null || principal.HasClaim(c => c.Type == McpClaims.MemberId))
        return principal; // ⑫ 不是 OAuth Token，或已經轉換過

    var userId = await links.FindMemberAsync(issuer, subject, Cancellation.Token.None); // ⑬ 查對應表
    if (userId is null)
        return principal;

    var result = principal.Clone(); // ⑭ 不修改原本的物件
    ((ClaimsIdentity)result.Identity!).AddClaim(new Claim(McpClaims.MemberId, userId));
    return result;
}
```

程式說明：

片段	做了什麼	為什麼這樣寫
⑧ <code>ForwardAuthenticate</code>	McpAuth 方案自己不驗 Token，轉給 OpenIdDict	不設定時它預設轉給名為 <code>Bearer</code> 的方案，本平台沒有，會 500（見〈常見錯誤〉）
⑨ <code>ResourceMetadata</code>	提供資源中繼資料；401 時加上 <code>resource_metadata</code>	Client 只要知道 <code>/mcp</code> 網址就能找到授權服務
授權原則只列 McpAuth 方案	驗證轉給 OpenIdDict，Challenge 由 McpAuth 回應	401 只有一個 <code>WWW-Authenticate</code> 標頭，而且帶 <code>resource_metadata</code> ；網站 Cookie 不在清單內，所以帶 Cookie 也是 401
<code>RequireClaim(member_id)</code>	Token 合法但對應不到會員 → 403	401 表示「你是誰？」，403 表示「知道你是誰，但不讓你進來」
① ②	用 <code>iss</code> 、 <code>sub</code> 查 <code>ExternalIdentityLinks</code>	網站 Cookie 沒有 <code>iss</code> 、 <code>sub</code> ，第一個 <code>if</code> 就略過，不會多查資料庫
③ <code>Clone()</code>	回傳新的 Principal	ASP.NET Core 可能對同一個請求執行多次轉換，不改原物件，且已有 <code>member_id</code> 就不重查

`McpToolHandlers.CallToolAsync` 只改兩行：呼叫者從 `McpClaims.MemberId` 取得，`ClientId` 從 Token 的 `OpenIdDictConstants.Claims.ClientId`（`client_id`）取得。

預期結果：不帶 Token → 401 並帶 `resource_metadata`；GET `/.well-known/oauth-protected-resource/mcp` 回示範步驟 2 的 JSON。

步驟五 測試用戶端改用 OAuth（10 分鐘）

目的：把 Section 09 的固定 Token 換成 SDK 的 OAuth 設定。

```
// 範例/10_ConnectorOAuth與會員授權/McpPlatform/tools/McpPlatform.McpTestClient/Program.cs (OAuth 設定)
var transport = new HttpClientTransport(new HttpClientTransportOptions
{
    Endpoint = endpoint,
    Name = "McpPlatform 課堂測試用戶端",
    OAuth = new ClientOAuthOptions // ① 收到 401 就自動走 OAuth
    {
        ClientId = "classroom-mcp-client", // ② 後臺「Connector 登錄」的 Client Id
        RedirectUri = new Uri("http://127.0.0.1:7071/callback"), // ③ 必須和登錄的一字不差
        Scopes = ["mcp:tools"],
        AuthorizationCallbackHandler = login // ④ 讓會員登入、同意，拿回授權碼
    }
});
```

login 預設是 `BrowserLogin.HandleAsync`：

```
// 範例/10_ConnectorOAuth與會員授權/McpPlatform/tools/McpPlatform.McpTestClient/BrowserLogin.cs (HandleAsync 節錄)
using var listener = new HttpListener();
listener.Prefixes.Add(redirectUri.GetLeftPart(UriPartial.Path).TrimEnd('/') + "/"); // ① 在 Redirect URI 上等瀏覽器回來
listener.Start();

Console.WriteLine($"請在瀏覽器登入並按「允許」:\n{authorizationUri}");
Process.Start(new ProcessStartInfo(authorizationUri.ToString()) { UseShellExecute = true }); // ② 開預設瀏覽器

var context = await listener.GetContextAsync().WaitAsync(ct); // ③ 等轉回來的那一個請求
var query = HttpUtility.ParseQueryString(context.Request.Url!.Query);
```

```
// 範例/10_ConnectorOAuth與會員授權/McpPlatform/tools/McpPlatform.McpTestClient/BrowserLogin.cs (HandleAsync 結尾)
if (query["code"] is null)
    throw new InvalidOperationException(message); // 會員按了「拒絕」：停止
return new AuthorizationResult { Code = query["code"], State = query["state"], Iss = query["iss"] }; // ④ SDK 會比對 state 與 iss
```

程式說明：

標號	做了什麼	為什麼這樣寫
① (Program)	設定 OAuth 後，SDK 遇到 401 會自己讀中繼資料、產生 PKCE、組授權網址、換 Token	我們只提供「怎麼讓會員登入」這一段
② ③ (Program)	Client Id 與 Redirect URI 與後臺登錄的一致	任一不符，OpenIdDict 直接拒絕，不會轉址
④ (Program)	AuthorizationCallbackHandler	舊的 AuthorizationRedirectDelegate 在 SDK 2.2.0 已標記過時（只能回傳 code，無法回傳 iss）
① (BrowserLogin)	HttpListener 聽 http://127.0.0.1:7071/callback/	請求 /callback?code=... 也符合這個前綴（實測）
②	用預設瀏覽器開授權網址	會員在自己熟悉的瀏覽器、平台自己的網頁輸入密碼
④ (BrowserLogin)	回傳 code、state、iss	SDK 要求 state 完全相符才換 Token；iss 用來防止授權服務混淆攻擊

自動化驗收時，設定環境變數 MCP_LOGIN_EMAIL、MCP_LOGIN_PASSWORD，Program 會改用 ScriptedLogin：以 HttpClient 代替瀏覽器完成登入與按「允許」，結果與瀏覽器模式相同。

預期結果：與示範步驟 4 相同；ApiCallLogs.ClientId 是 classroom-mcp-client。

檢查點

- ☐ 後臺登錄的四組資料得到示範步驟 1 的四個訊息
- ☐ 不帶 Token 呼叫 /mcp：401，WWW-Authenticate 帶 resource_metadata
- ☐ 測試用戶端完成授權，列出 3 個工具並呼叫成功
- ☐ ExternalIdentityLinks 有 (http://localhost:5199/, member-a)；ApiCallLogs.ClientId 是 classroom-mcp-client
- ☐ 拒絕、錯誤 Audience、過期、刪除對應、只帶 Cookie 都被拒絕

常見錯誤

以下都是撰寫本單元時實際遇到的原文。

症狀（錯誤訊息原文）	原因	處理
呼叫 <code>/mcp</code> 得到 500： <code>No authentication handler is registered for the scheme 'Bearer'. The registered schemes are: ... OpenIddict.Validation.AspNetCore, McpAuth.</code>	<code>AddMcp()</code> 的方案預設把驗證轉給 <code>Bearer</code> （JwtBearer），本平台用的是 OpenIddict	<code>AddMcp</code> 設定 <code>ForwardAuthenticate = OpenIddictValidationAspNetCoreDefaults.AuthenticationScheme</code>
授權網址回 400： <code>error:invalid_target</code> 、 <code>One of the specified 'resource' parameters is invalid.</code> （ID2190）	SDK 送了 <code>resource</code> 參數，但 OpenIddict 沒登錄這個資源，或這個 Client 沒有 <code>rsrc</code> 權限	<code>RegisterResources(...)</code> ，登錄 Client 時加 <code>Permissions.Prefixes.Resource + 資源</code> ；已登錄的舊 Client 要重新登錄
編譯： <code>CS1061: 'HttpContext' 未包含 'GetOpenIddictServerRequest' 的定義</code>	這個擴充方法在 <code>Microsoft.AspNetCore</code> 命名空間	加 <code>using Microsoft.AspNetCore;</code>
編譯警告： <code>MCP9007: 'AuthorizationRedirectDelegate' 已經過時</code>	SDK 2.2.0 改用能回傳 <code>iss</code> 的新 API	改用 <code>AuthorizationCallbackHandler</code> ，回傳 <code>AuthorizationResult</code>
編譯警告： <code>CS8619 ... 'Task<AuthorizationResult>' ... 與目標型別 'Task<AuthorizationResult?>' 不符合</code>	處理常式的回傳型別宣告成為 null	方法簽章寫 <code>Task<AuthorizationResult?></code>
自己寫腳本送出同意表單得到 400	只抓了 <code>type="hidden" name=...</code> 順序的欄位，漏掉防偽權杖（它的屬性順序是 <code>name</code> 在前）	另外擷取 <code>__RequestVerificationToken</code> 一起送出
<code>/mcp</code> 回 401，網站主控台： <code>The token was rejected because it had no valid audience.</code>	Token 的 <code>aud</code> 不是 <code>http://localhost:5199/mcp</code>	Client 的 <code>resource</code> 或 <code>OAuth:McpResource</code> 要與實際連線的網址一致
<code>/mcp</code> 回 401，網站主控台： <code>The authentication demand was rejected because the token was expired.</code>	Token 過期（實測沒有寬限時間）	重新授權；調整 <code>AccessTokenMinutes</code>

單元驗收（15 分鐘）

驗收標準：Client 完成授權、列出工具並呼叫；未登入、過期 Token、錯誤 Audience 被拒絕。

驗收方式：

1. **Client 完成授權**：`dotnet run --project tools/McpPlatform.McpTestClient`，在瀏覽器登入 `a@demo.test` 並按「允許」，頁面顯示「授權完成」。
2. **列出工具並呼叫**：主控台列出 3 個工具，`products_search` 成功並有 `CorrelationId`；`ApiCallLogs` 那一列的 `UserId = member-a`、`ClientId = classroom-mcp-client`。
3. **未登入被拒絕**：不帶 Token 呼叫 `/mcp` → 401 + `resource_metadata`；未登入網站就開授權網址 → 302 到登入頁；只帶網站 Cookie → 401。
4. **過期 Token 被拒絕**：以 `OAuth__AccessTokenMinutes=1` 啟動網站，取得 Token 後等 60 秒以上再呼叫 → 401（實測 62 秒）。
5. **錯誤 Audience 被拒絕**：`pwsh tools/oauth-flow.ps1 -Resource http://localhost:5199/reports` 取得 `aud` 為報表資源的 Token → 呼叫 `/mcp` 401。
6. **反向驗證**：刪除 `ExternalIdentityLinks` 中 `member-a` 那一列，同一張合法 Token → 403；同意頁按「拒絕」→ `error=access_denied`；登錄 `http://example.com/callback` → 被拒。

回顧：Cookie 給瀏覽器、Token 給 Connector；Client 從 401 一路探索到授權服務，會員在平台自己的頁面登入並同意，PKCE 讓公開用戶端也安全；`/mcp` 檢查簽章、Issuer、Audience、期限，再以 Issuer + Subject 對應會員。下一單元（Section 11）在每次 `tools/list`、`tools/call` 加上 Grant、額度、版本與停權檢查，讓會員只看得到、也只叫得動被核准的工具。

延伸閱讀

- [MCP 規範：Authorization \(2025-11-25\)](#)：資源中繼資料、PKCE、`resource` 參數的要求。
- [RFC 7636 PKCE](#)、[RFC 8707 Resource Indicators](#)、[RFC 9728 Protected Resource Metadata](#)、[RFC 9207 Issuer Identification](#)。
- [OpenIddict 文件](#)：授權碼流程、Passthrough 模式、資源指示符。

第 11 章

動態工具權限與呼叫管制

工具清單依權限過濾；每次呼叫重新驗證 Grant、Scope、版本、會員狀態、期限與剩餘額度；共用授權服務、原子額度計數、Rate Limit、撤銷時讓授權快失效、錯誤回應一致化



課程時數 2 小時 (120 分鐘)

當堂成果

即時撤銷、額度、版本與停權檢查

範例專案

範例/11_動態工具權限與呼叫管制

學習目標

上完這個單元，你能夠：

- 寫一個共用的 `ToolAccessService`，讓 `tools/list` 與 `tools/call` 用同一套規則判斷會員能不能用某個工具。
- 在每次 `tools/call` 依序重新檢查 Token 的 Scope、會員狀態、Grant 狀態與期限，並說明為什麼這裡不能用快取。
- 用一個 SQL `UPDATE ... WHERE UsedCount < Quota` 做原子的額度預留，實測 20 個並行呼叫搶 10 次額度，剛好 10 個成功。
- 讓「從未獲准」的工具與「不存在」的工具回應完全相同，猜工具名稱得不到任何資訊。
- 加上每位會員每分鐘的 Rate Limit，並在撤銷 Grant 時讓 `tools/list` 的快取立即失效。

先備知識

- 完成 Section 10： `/mcp` 以 OAuth Token 驗證， `member_id` 宣告由 Issuer + Subject 對應而來。
- Section 07 的 `ApiGrants`（狀態 Active／Revoked／Expired、有效期間、每日額度 `Quota`）與後臺撤銷功能； `ApiUsageCounters` 表已建好， `(ApiGrantId, PeriodStart)` 有唯一索引。
- 示範資料：`member-a`（王小明）有課程查詢 v1 的 Grant #4（每日 200 次）與已撤銷的商品查詢 Grant #1； `member-b`（林小華）有課程查詢 v1 的 Grant #5（每日 300 次）；沒有人有服務工單的 Grant。
- Section 10 的 `tools/oauth-flow.ps1`（ `-SaveToken` 會把 Token 存到 `%TEMP%\mcp-access-token.txt`）。

觀念說明（20 分鐘）

為什麼每次呼叫都要重新檢查

Access Token 簽發後 10 分鐘內都有效，Client 也可能把工具清單記在自己那裡。如果只在發 Token 或列工具時檢查一次，這段時間裡發生的事平台都擋不住：

這段時間可能發生	只檢查一次的後果	本單元的做法
管理者撤銷 Grant	舊 Token 還能呼叫 10 分鐘	<code>tools/call</code> 每次查資料庫的 Grant
會員被停權	已發出的 Token 照樣可用	每次檢查 <code>AspNetUsers.IsActive</code>
Grant 到期	過了期限還能呼叫	每次比對 <code>ValidFromUtc</code> 、 <code>ExpiresAtUtc</code>
額度用完	超量呼叫	每次呼叫前原子地預留一次
版本下架	呼叫已下架的 API	Section 09 起 <code>FindActiveToolAsync</code> 只找已發布版本

Token 回答的是「你是誰、哪個 Connector」；**能不能用某個工具，是平台資料庫的即時狀態**，每次都要重看。

一次 tools/call 的檢查順序

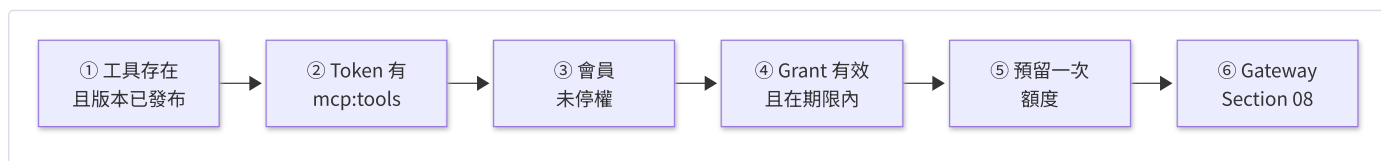


圖 11-1 tools/call 在送出上游之前的檢查

編號	不通過時	回應	計入額度
①	名稱不存在、版本已下架	協定錯誤 <code>-32602</code> 「沒有名為 … 的工具。」	否
②	<code>SCOPE_MISSING</code>	工具錯誤 (<code>isError = true</code>)	否
③	<code>MEMBER_SUSPENDED</code>	工具錯誤	否
④	從未獲准： <code>GRANT_REQUIRED</code> ；其他： <code>GRANT_REVOKED</code> 、 <code>GRANT_EXPIRED</code> 、 <code>GRANT_NOT_YET_VALID</code>	從未獲准回應與 ① 完全相同；其他是工具錯誤	否
⑤	<code>QUOTA_EXCEEDED</code>	工具錯誤	否
⑥	Gateway 的各種失敗（參數、逾時、上游錯誤）	工具錯誤	是 (⑤ 已預留)

每一種拒絕都照樣寫一筆 `ApiCallLogs`（`Outcome = Denied`、固定的 `ErrorCode`、`CorrelationId`），Section 13 的儀表板與稽核查詢靠這些紀錄。

從未獲准 = 不存在

`member-a` 從未申請過服務工單。如果 `ticket_create` 回「你沒有權限」，等於告訴他「這個工具存在，只是你不能用」；攻擊者可以逐一猜名字，拼出平台有哪些工具。所以「從未獲准」與「不存在」回同一句話：

```
tools/call ticket_create → 協定錯誤 -32602：沒有名為 ticket_create 的工具。
tools/call no_such_tool → 協定錯誤 -32602：沒有名為 no_such_tool 的工具。
```

對外看不出差別，但 `ApiCallLogs` 裡 `ticket_create` 那一列寫的是 `GRANT_REQUIRED`，管理者查得到誰在猜。會員曾經獲准、後來被撤銷的工具則直接說明原因（`GRANT_REVOKED`）：那是他自己知道的事，說清楚比較好處理。

額度的計次規則

大綱要求「重試是否計次需在文件中固定」。本平台固定如下：

情況	計次	理由
通過 ①～④，Gateway 成功	1 次	—
通過 ①～④，參數不符合 Schema（ <code>INVALID_ARGUMENTS</code> ）	1 次	預留發生在 Gateway 之前；參數由 AI 產生，錯了也是一次呼叫
通過 ①～④，上游逾時、連不上、4xx/5xx	1 次	請求可能已經到過上游，平台無法確定，一律不退還
Client 用同一個 <code>Idempotency-Key</code> 重試	每次都算 1 次	冪等鍵保證上游不重複寫入，但每次仍是一次完整的呼叫與檢查
在 ①～⑤ 被拒絕	0 次	沒有使用任何上游資源

計量期間是**臺灣日期**（UTC+8）的一天，午夜 0 點換日，不受伺服器所在時區影響。

原子計數：兩個人不能同時拿到最後一次

最直覺的寫法是「先讀已用次數，小於額度就加一再存回去」。兩個請求同時讀到 199（額度 200），兩個都判斷「還剩 1 次」，都存回 200，額度就被突破了。

正確做法是讓**比較與加一**發生在同一個 SQL 陳述式：

```
UPDATE ApiUsageCounters SET UsedCount = UsedCount + 1
WHERE ApiGrantId = @grant AND PeriodStart = @today AND UsedCount < @quota;
```

SQL Server 在更新那一列時會鎖住它；第二個請求等第一個完成後，看到的已經是 200，`WHERE` 不成立，影響 0 列。程式只要看「影響了幾列」：1 列＝預留成功，0 列＝已用完（或今天還沒有這一列）。

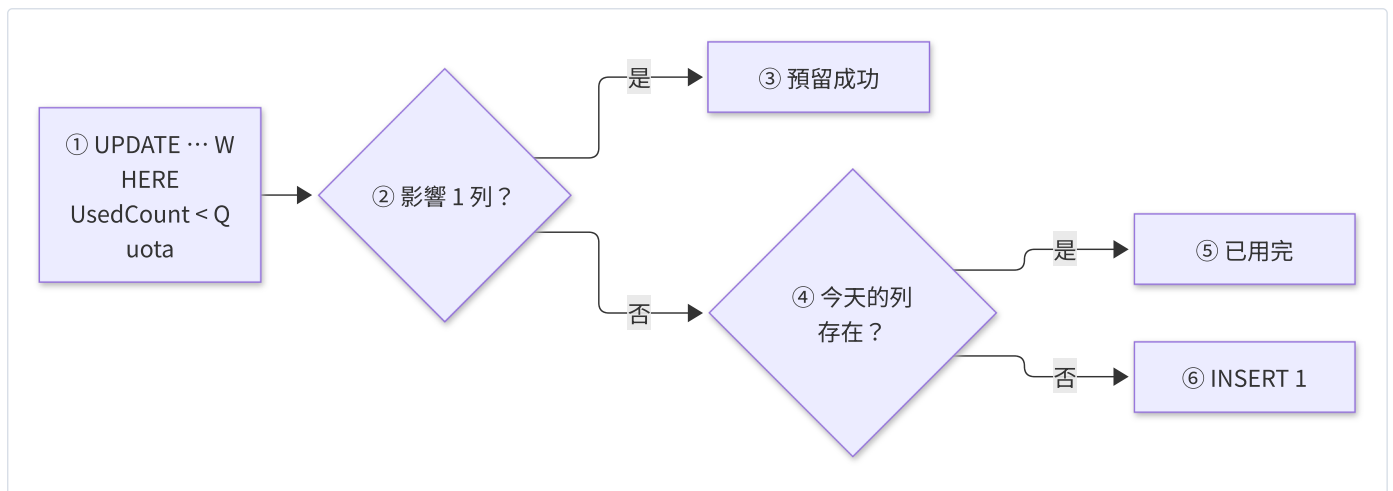


圖 11-2 原子的額度預留

編號	做了什麼	並行時會怎樣
① ②	一個 UPDATE 同時比較與加一	同一列的更新互相排隊，不可能兩個都成功拿到最後一次
④ ⑤	列存在卻沒更新到 → 已達上限	—
⑥	今天第一次呼叫，新增 <code>UsedCount = 1</code>	兩個請求同時新增，唯一索引讓其中一個失敗（2601），它回到 ① 再做一次 UPDATE

快取與 Rate Limit

- `tools/list` 可以用快取：AI 應用常常反覆列工具，每次都查 Grant 很浪費。會員的 Grant 放在記憶體快取 60 秒；**核准、撤銷時立即清除**，所以正常操作下清單是即時的。
- `tools/call` 不用快取：萬一 Grant 是被其他途徑改掉的（例如直接改資料庫），快取最多 60 秒後才更新；清單晚一點消失只是顯示問題，但呼叫必須當下就被擋下。本單元實測過這個情況（教師示範步驟 5）。
- Rate Limit**：額度管的是「一天多少次」，Rate Limit 管的是「一分鐘內不能太密集」，防止失控的 AI 迴圈瞬間用光額度或拖垮上游。`/mcp` 的每個請求（包含 `tools/list`）都算，每位會員每分鐘 60 次，超過回 HTTP 429。

限制與注意事項

- 快取在每台伺服器的記憶體裡；部署多台時，撤銷只清掉處理那個請求的那一台。這不影響安全（`tools/call` 不用快取），只影響清單最慢 60 秒才更新。
- Rate Limit 的計數也在單台記憶體裡，多台部署時每台各算各的。
- 額度預留後即使上游失敗也不退還；要退還就必須能確定上游沒有處理，本課程不處理這種情況。

- 會員停權的操作介面在 Section 12；本單元以 SQL 修改 `IsActive` 來測試。

名詞對照

名詞	白話說明	在本課程中是
共用授權服務	權限規則只寫一次，所有入口都呼叫它	<code>ToolAccessService</code>
原子操作	「檢查 + 修改」一次完成，中間不會被別人插隊	<code>UPDATE ... WHERE UsedCount < Quota</code>
預留 (Reserve)	呼叫之前先扣一次額度	<code>TryReserveAsync</code>
快取失效	資料變更時把舊的快取丟掉	<code>IToolAccessCache.InvalidateMember</code>
Rate Limit	單位時間內的請求次數上限	每位會員每分鐘 60 次，超過 429
列舉攻擊	逐一猜名稱，從回應差異推出存在哪些資源	從未獲准與不存在回應相同

教師示範 (25 分鐘)

示範情境：`member-a` 只看得到自己獲准的工具；猜別的工具名稱得到與不存在相同的回應；20 個並行呼叫搶最後 10 次額度；撤銷後舊 Token 立刻被拒；停權、Scope 不足、呼叫太密集都被擋下。

示範步驟

1. 啟動示範 API 與網站，執行測試用戶端（`member-a` 登入並按「允許」）。本單元實測：

```
== tools/list: 共 1 個工具
   courses_search    課程查詢 (v1): 依程度篩選課程，回傳課程代碼與時數。

== tools/call courses_search {"level": "進階"}
   isError = False
   {"count": 2, "items": [{"code": "C-301", "title": "MCP Server 與 AI 工具串接", "hours": 18, "level": "進階"}, ...
   CorrelationId: 425df3f993b34276

== tools/call courses_search {"level": "高手"}
   isError = True
   [INVALID_ARGUMENTS] 參數不符合規格: /level enum: Value should match one of the values specified by the enum
   CorrelationId: 41c45ef7704c473c

== tools/call products_search {"keyword": "茶"}
   isError = True
   [GRANT_REVOKED] 使用授權已被撤銷。
   CorrelationId: e0161ec897cd4efb

== tools/call no_such_tool {}
   協定錯誤 -32602 (InvalidParams): Request failed (remote): 沒有名為 no_such_tool 的工具。
```

Section 10 列出 3 個工具，現在只剩 1 個。`products_search` 是 `member-a` 曾獲准、已被撤銷的 Grant #1，所以明白告知原因。

2. 越權猜測。先取得 Token，再呼叫從未獲准的服務工單（`tools/mcp-call.ps1` 用存下的 Token 直接送 JSON-RPC）：

```
$env:MCP_LOGIN_PASSWORD = '<示範會員密碼>'
pwsh tools/oauth-flow.ps1 -Resource http://localhost:5199/mcp -SaveToken
pwsh tools/mcp-call.ps1 -Tool ticket_create -Arguments '{"subject": "猜猜看", "confirmed": true}'
```

協定錯誤 -32602：沒有名為 `ticket_create` 的工具。

`ApiCallLogs` 最新一列：`member-a` | 3 | `ticket_create` | Denied | GRANT_REQUIRED。

3. 並行搶額度。把 Grant #4 今天的已用次數設為 190（只剩 10 次），同時送 20 個呼叫：

```
-- 在 MCPServerManag 執行；僅供課堂測試
UPDATE ApiUsageCounters SET UsedCount = 190
WHERE ApiGrantId = 4 AND PeriodStart = CAST(SWITCHOFFSET(SYSDATETIMEOFFSET(), '+08:00') AS date);
```

```
pwsh tools/mcp-call.ps1 -Tool courses_search -Arguments '{"level": "入門"}' -Parallel 20
```

```
[QUOTA_EXCEEDED] × 10
SUCCESS × 10
```

`UsedCount` 最後是 200，一次都沒有多。只剩 1 次、同時送 2 個的情況也實測過：一個成功、一個 `QUOTA_EXCEEDED`。

4. 撤銷後使用舊 Token。先 `-List` 確認看得到 `courses_search`，再由管理者在「Grant 管理」撤銷 Grant #4，**不換 Token** 再試：

```
撤銷前 tools/list: 工具 (1 個): courses_search
撤銷後 tools/list: 工具 (0 個):
撤銷後 tools/call courses_search: [GRANT_REVOKED] 使用授權已被撤銷。 | CorrelationId: 0c7ca7bd68344311
```

管理後臺

[總覽](#)[API 管理](#)[資料庫概況](#)[上架審核](#)[使用申請審核](#)[Grant 管理](#)[API 測試台](#)[MCP 工具映射](#)[Connector 登錄](#)[會員管理 \(Section 12 \)](#)

Grant 管理

共 3 筆使用授權。撤銷只改授權狀態，原本的申請與核准紀錄不變。

#	會員	API	有效期間	今日已用／額度	狀態	撤銷
5 申請 #4	林小華	課程查詢 v1 tools:call	2026-09-26 19:35 ~ 2026-11-25 19:35	0 / 300	✓ 有效	原因 撤銷
4 申請 #2	王小明	課程查詢 v1 tools:call	2026-09-26 19:35 ~ 2026-10-26 19:35	200 / 200 今日已用完	已撤銷 課堂示範：撤銷後舊 Token 必須被拒	
1 申請 #1	王小明	商品查詢 v1 tools:call	2026-09-21 17:00 ~ 2026-10-21 17:00	0 / 100	已撤銷 示範撤銷：用途已結束。	

圖 11-3 Grant 管理：今日已用次數與撤銷狀態

畫面重點：

- ①「今日已用／額度」欄：本單元新增。Grant #4 顯示 200 / 200 與紅字「今日已用完」，就是步驟 3 的結果；數字來自 `ApiUsageCounters` 今天（臺灣日期）的那一列。
- ②Grant #4 的「已撤銷」與原因：「課堂示範：撤銷後舊 Token 必須被拒」。撤銷只改 Grant 狀態，申請 #2 的紀錄不變。
- ③Grant #5（林小華）仍有效：右側有撤銷欄位；停權與 Scope 的測試用的就是這位會員。
- 快取示範（說明為什麼 `tools/call` 不能用快取）。用 `member-b` 的 Token 先列一次工具（填入快取），再直接用 SQL 把 Grant #5 改成 Revoked（沒有經過撤銷功能，所以快取沒被清掉）。實測：

```
20:52:13 list (填入快取) : courses_search
20:52:13 SQL 直接改 Revoked
20:52:13 tools/list : courses_search          ← 快取還沒過期
                tools/call : [GRANT_REVOKED] 使用授權已被撤銷。    ← 呼叫當下就被擋
20:52:43 tools/list : (空)                  ← 快取在填入後 60 秒過期
```

示範完把 Grant #5 改回 Active。

6. 停權、Scope、Rate Limit (`member-b`)：

情況	結果
SQL 設 <code>IsActive = 0</code> 後 <code>tools/list</code>	工具 (0 個)
SQL 設 <code>IsActive = 0</code> 後 <code>tools/call</code>	[MEMBER_SUSPENDED] 會員已停權，不能呼叫工具。
Token 的 scope 是 <code>reports:read (resource=/mcp)</code>	<code>tools/list</code> 0 個； <code>tools/call</code> [SCOPE_MISSING] Access Token 沒有 <code>mcp:tools</code> 權限。
<code>Mcp_RateLimitPerMinute=5</code> ，連續送 <code>tools/list</code>	前 5 次 200，第 6 次起 HTTP 429
此時 <code>member-a</code> 送 <code>tools/list</code>	200 (每位會員各自計算)
不帶 Token	401 (先驗證身分，才輪到 Rate Limit)

學員實作（60 分鐘）

範例專案：範例/11_動態工具權限與呼叫管制/McpPlatform/（從 Section 10 複製）。本單元新增 Core 的 `Mcp/ToolAccessService.cs`、`Mcp/QuotaPeriod.cs`；Infrastructure 的 `Mcp/EfToolAccessRepository.cs`（含 `GrantCache`）；`tools/mcp-call.ps1`。修改：`McpToolHandlers.cs`、`ApiInvocationService`（新增 `DenyAsync`）、`AccessService`（核准與撤銷後清快取）、Grant 管理頁（今日已用次數）、`Program.cs`、`appsettings.json`（`Mcp:RateLimitPerMinute`）、測試用戶端的呼叫清單。本單元沒有新的 Migration。

步驟一 共用授權服務（20 分鐘）

目的：把「會員能不能用這個工具」寫成一個服務，`tools/list` 與 `tools/call` 都呼叫它。

```
// 範例/11_動態工具權限與呼叫管制/McpPlatform/src/McpPlatform.Core/Mcp/ToolAccessService.cs (AuthorizeCallAsync)
public async Task<AccessDecision> AuthorizeCallAsync(string userId, IEnumerable<string> tokenScopes, int versionId, CancellationToken ct)
{
    if (!tokenScopes.Contains(RequiredTokenScope)) // ① Token 必須有 mcp:tools
        return AccessDecision.Deny(AccessErrors.ScopeMissing, $"Access Token 沒有 {RequiredTokenScope} 權限。");
    if (!await repository.IsMemberActiveAsync(userId, ct)) // ② 會員停權
        return AccessDecision.Deny(AccessErrors.MemberSuspended, "會員已停權，不能呼叫工具。");

    var now = clock.UtcNow();
    var grants = (await repository.ListGrantsAsync(userId, allowCached: false, ct)) // ③ 一定查資料庫，撤銷立刻生效
        .Where(g => g.VersionId == versionId).ToList();
    if (grants.Count == 0)
        return AccessDecision.Hide(); // 從未獲准：與「沒有這個工具」相同

    var usable = grants.FirstOrDefault(g => Problem(g, now) is null);
    if (usable is null)
    {
        var (code, message) = Problem(grants[0], now)!.Value; // 最新一筆為什麼不能用
        return AccessDecision.Deny(code, message, grants[0].GrantId);
    }

    if (!await repository.TryReserveAsync(usable.GrantId, QuotaPeriod.For(now), usable.Quota, ct)) // ④ 以臺灣日期計，先預留一次；上游失敗也不
        return AccessDecision.Deny(AccessErrors.QuotaExceeded, // 退還
            $"今日額度 {usable.Quota} 次已用完，臺灣時間明天 0 點重新計算。", usable.GrantId);
    return AccessDecision.Allow(usable.GrantId);
}
```

同一個檔案裡判斷單筆 Grant 的規則，`tools/list` 的過濾也用它：

```
// 範例/11_動態工具權限與呼叫管制/McpPlatform/src/McpPlatform.Core/Mcp/ToolAccessService.cs (Problem)
private static (string Code, string Message)? Problem(GrantSnapshot g, DateTimeOffset now) => g switch
{
    { Status: GrantStatus.Revoked } => (AccessErrors.GrantRevoked, "使用授權已被撤銷。"),
    { Status: GrantStatus.Expired } => (AccessErrors.GrantExpired, "使用授權已到期。"),
    _ when now >= g.ExpiresAtUtc => (AccessErrors.GrantExpired, "使用授權已到期。"),
    _ when now < g.ValidFromUtc => (AccessErrors.GrantNotYetValid, "使用授權尚未生效。"),
    _ when !g.Scope.Split(' ').Contains(RequiredGrantScope) => (AccessErrors.ScopeMissing, $"使用授權沒有 {RequiredGrantScope} 權限。"),
    _ => null
};
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	Token 的 scope 必須含 <code>mcp:tools</code>	Section 10 的 <code>reports:read</code> Token 只要 <code>aud</code> 對了也能進 <code>/mcp</code> ，在這裡擋下
②	查 <code>AspNetUsers.IsActive</code>	停權的會員登不進網站，但手上的 Token 還沒過期
③	<code>allowCached: false</code>	撤銷、到期要從下一次呼叫起立即生效
	<code>Hide()</code>	這個版本完全沒有 Grant 處理常式會回「沒有名為 … 的工具」
	<code>grants[0]</code>	有 Grant 但都不能用時，說明最新一筆的原因 會員可能曾有多筆（撤銷後重新申請），最新一筆最有參考價值
④	<code>QuotaPeriod.For(now)</code>	取臺灣日期後預留 所有檢查都通過才扣額度，被拒絕的呼叫不計次
	<code>Problem</code>	依序檢查撤銷、到期、尚未生效、Grant 的 scope 狀態仍是 Active 但時間已過期（還沒有排程把它改成 Expired）也算到期

`FilterVisibleAsync` 用同一個 `Problem`，只是 Grant 允許用快取、不預留額度：Token 沒有 `mcp:tools` 或會員停權就回空清單，否則只留下有可用 Grant 的工具。

預期結果：`member-a` 的 `tools/list` 只剩 `courses_search`。

步驟二 原子的額度預留（15 分鐘）

目的：實作圖 11-2。

```
// 範例/11_動態工具權限與呼叫管制/McpPlatform/src/McpPlatform.Infrastructure/Mcp/EfToolAccessRepository.cs (TryReserveAsync)
public async Task<bool> TryReserveAsync(int grantId, DateOnly period, int quota, CancellationToken ct)
{
    for (var attempt = 0; attempt < 2; attempt++)
    {
        var updated = await db.ApiUsageCounters
            .Where(c => c.ApiGrantId == grantId && c.PeriodStart == period && c.UsedCount < quota) // ㉞ 一個 UPDATE 同時比較與加一：
            .ExecuteUpdateAsync(s => s.SetProperty(c => c.UsedCount, c => c.UsedCount + 1), ct); // 兩個請求不可能都看到「還剩 1 次」
        if (updated == 1)
            return true;
        if (await db.ApiUsageCounters.AnyAsync(c => c.ApiGrantId == grantId && c.PeriodStart == period, ct))
            return false; // ㉞ 今天的列已存在：表示已達上限
        if (quota < 1)
            return false;

        try
        {
            db.ApiUsageCounters.Add(new ApiUsageCounter { ApiGrantId = grantId, PeriodStart = period, UsedCount = 1 }); // ㉞ 今天第一次呼
            await db.SaveChangesAsync(ct);
            return true;
        }
        catch (DbUpdateException ex) when (ex.InnerException is SqlException { Number: 2601 or 2627 })
        {
            db.ChangeTracker.Clear(); // ㉞ 另一個請求先建好今天的列：回到 ㉞ 再試一次
        }
    }
    return false;
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
① <code>ExecuteUpdateAsync</code>	EF Core 直接送出一個 <code>UPDATE ... SET UsedCount = UsedCount + 1 WHERE ... AND UsedCount < @quota</code> ，回傳影響列數	不先把資料讀進記憶體；讀出來再存回去的寫法在並行時會突破額度
②	列存在但沒更新到	一定是已達上限
③	今天第一次，新增一列	計量期間換日時自然產生新的一列，舊的列留作歷史
④	唯一索引衝突（2601／2627）	另一個請求剛好同時新增；清掉失敗的追蹤，回到 ① 用 UPDATE 預留
<code>attempt < 2</code>	最多兩輪	第二輪時今天的列一定已存在，只會走 ① 或 ②

`QuotaPeriod.For` 在 `McpPlatform.Core/Mcp/QuotaPeriod.cs`：用 `TimeZoneInfo.FindSystemTimeZoneById("Asia/Taipei")` 把 UTC 時間轉成臺灣日期。Grant 管理頁的「今日已用」也用它，兩邊的「今天」一定一致。

預期結果：示範步驟 3 的 `SUCCESS × 10`、`[QUOTA_EXCEEDED] × 10`，`UsedCount = 200`。

步驟三 處理常式與一致的錯誤回應（15 分鐘）

目的：`tools/call` 先問授權服務，依結果決定呼叫上游、回工具錯誤，或假裝工具不存在。

```
// 範例/11_動態工具權限與呼叫管制/McpPlatform/src/McpPlatform.Web/Mcp/McpToolHandlers.cs (CallToolAsync 節錄)
var decision = await access.AuthorizeCallAsync(userId, context.User!.GetScopes(), tool.VersionId, ct); // ㉞ 每次呼叫都重新檢查
if (decision.Hidden)
{
    await gateway.DenyAsync(request, decision.ErrorCode, decision.Message, ct); // 紀錄照寫，但回應與「沒有這個工具」一模一樣
    throw NoSuchTool(name);
}
var result = decision.Allowed
    ? await gateway.InvokeAsync(request with { GrantId = decision.GrantId }, ct) // ㉞ 通過才呼叫上游，紀錄帶 GrantId
    : await gateway.DenyAsync(request with { GrantId = decision.GrantId }, decision.ErrorCode, decision.Message, ct);
```



```
// 範例/11_動態工具權限與呼叫管制/McpPlatform/src/McpPlatform.Core/Gateway/ApiInvocationService.cs (DenyAsync)
/// <summary>呼叫前的權限檢查沒有通過 (Section 11)：不送出請求，但一樣產生追查代碼、寫一筆紀錄。</summary>
public async Task<InvocationResult> DenyAsync(InvocationRequest request, string errorCode, string message, CancellationToken ct)
{
    var target = await targets.GetAsync(request.VersionId, ct);
    return await FinishAsync(request, target, Fail(CallOutcome.Denied, errorCode, message, Guid.NewGuid().ToString("N")[..16]), ct);
}
```

程式說明：

片段	做了什麼	為什麼這樣寫
<code>context.User!.GetScopes()</code>	OpenIddict 的擴充方法，取出這張 Token 的 scope 清單	不必自己解析 JWT 的 <code>scope</code> 宣告
<code>decision.Hidden</code>	先寫紀錄，再丟出與不存在相同的 <code>NoSuchTool(name)</code>	外部看不出差別，內部查得到 <code>GRANT_REQUIRED</code>
<code>request with { GrantId = ... }</code>	記錄用了哪一筆 Grant	<code>ApiCallLogs.ApiGrantId</code> 從 Section 03 就預留了，Section 13 依它統計
<code>DenyAsync</code>	不送出請求，但經過同一個 <code>FinishAsync</code> 寫紀錄	寫紀錄的地方仍然只有一個；被拒的呼叫一樣有 <code>CorrelationId</code>
回傳 <code>CallToolResult</code>	被拒與 Gateway 失敗都是 <code>【錯誤代碼】說明</code> 加 <code>CorrelationId</code>	錯誤回應格式一致，AI 與使用者都用同一種方式判讀

撤銷時清快取，在 `AccessService.RevokeAsync` 存檔之後加一行（核准也一樣）：

```
// 範例/11_動態工具權限與呼叫管制/McpPlatform/src/McpPlatform.Core/Access/AccessService.cs (RevokeAsync 結尾)
await repository.SaveChangesAsync(ct);
toolAccessCache.InvalidateMember(grant.UserId); // Section 11：工具立刻從 tools/list 消失
return AccessResult.Ok("已撤銷。申請與核准的歷程仍保留。");
```

預期結果：示範步驟 4：撤銷後 `tools/list` 立即是 0 個，`tools/call` 得到 `GRANT_REVOKED`。

步驟四 Rate Limit（10 分鐘）

目的：每位會員每分鐘最多 60 個 `/mcp` 請求。

```
// 範例/11_動態工具權限與呼叫管制/McpPlatform/src/McpPlatform.Web/Program.cs (Rate Limit)
var mcpPermitPerMinute = builder.Configuration.GetValue("Mcp:RateLimitPerMinute", 60);
builder.Services.AddRateLimiter(options =>
{
    options.RejectionStatusCode = StatusCodes.Status429TooManyRequests; // ① 超過頻率回 429
    options.AddPolicy("mcp", http => RateLimitPartition.GetFixedWindowLimiter(
        http.User.FindFirstValue(McpClaims.MemberId) ?? "anonymous", // ② 每位會員各自計算
        _ => new FixedWindowRateLimiterOptions { PermitLimit = mcpPermitPerMinute, Window = TimeSpan.FromMinutes(1), QueueLimit = 0 }));
});
```

中介軟體放在授權之後，`/mcp` 端點加上 `.RequireRateLimiting("mcp")`：

```
// 範例/11_動態工具權限與呼叫管制/McpPlatform/src/McpPlatform.Web/Program.cs (中介軟體順序)
app.UseRateLimiter(); // Section 11：放在授權之後，才認得出是哪位會員
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	拒絕時回 429 Too Many Requests	預設是 503，會被誤會成伺服器故障
②	以 <code>member_id</code> 分區	一位會員用光不影響別人；實測 <code>member-b</code> 被限制時 <code>member-a</code> 仍是 200
<code>FixedWindow</code> 、 <code>QueueLimit = 0</code>	每分鐘一個窗口，超過直接拒絕，不排隊	AI 迴圈失控時應該立刻知道，而不是越排越久
<code>UseRateLimiter()</code> 在 <code>UseAuthorization()</code> 之後	授權中介軟體驗完 Token 才把使用者放進 <code>HttpContext.User</code>	放在前面的話，每個請求都落在 <code>anonymous</code> 同一區

`appsettings.json` 的 `Mcp:RateLimitPerMinute` 預設 60；課堂上用環境變數 `Mcp__RateLimitPerMinute=5` 啟動網站來測。

預期結果：示範步驟 6：第 6 次起 429；另一位會員不受影響；不帶 Token 仍是 401。

檢查點

- ☐ `member-a` 的 `tools/list` 只有 `courses_search`；呼叫 `ticket_create` 與 `no_such_tool` 的回應相同
- ☐ 20 個並行呼叫搶 10 次額度，剛好 10 個成功，`UsedCount` 不超過額度
- ☐ 撤銷 Grant 後不換 Token，`tools/list` 立即 0 個、`tools/call` 得到 `GRANT_REVOKED`
- ☐ 停權、Scope 不足、額度用完都有固定的錯誤代碼與 `CorrelationId`，`ApiCallLogs` 都有紀錄
- ☐ Grant 管理頁顯示今日已用次數

常見錯誤

以下都是撰寫本單元時實際遇到的情況。

症狀	原因	處理
設定每分鐘 5 次，測試腳本第 5 次就得到 HTTP 429	取得 Token 的腳本最後也會送一次 <code>tools/list</code> ，已經用掉 1 次	這是正確行為： <code>/mcp</code> 的每個請求都計入，包含 <code>tools/list</code>
直接用 SQL 把 Grant 改成 Revoked， <code>tools/list</code> 仍列出那個工具（約 30~60 秒）	沒有經過 <code>AccessService.RevokeAsync</code> ，快取沒被清除，要等 60 秒過期	一律用後臺撤銷； <code>tools/call</code> 不用快取，呼叫本來就會被擋
並行測試回 <code>[UPSTREAM_UNREACHABLE]</code> 連不上上游：無法連線，因為目標電腦拒絕連線。 <code>(localhost:5080)</code> ，而且額度被扣掉了	示範 API 沒有啟動；額度在 Gateway 之前預留，上游失敗不退還	啟動 <code>McpPlatform.DemoApis</code> ；這正是「上游失敗也計次」的規則
Grant 管理頁的撤銷欄被擠出表格右邊	新增「今日已用」欄後表格變寬，撤銷表單的輸入框與按鈕並排	撤銷表單改成上下排（ <code>flex-column</code> ）、編號欄 <code>text-nowrap</code>

單元驗收（15 分鐘）

驗收標準：越權猜測工具名稱仍無法呼叫；撤銷後使用舊 Token 也被拒絕；兩個並行呼叫不能突破最後一筆額度。

驗收方式：

1. **越權猜測工具名稱仍無法呼叫：**以 `member-a` 呼叫 `ticket_create` → `-32602` 沒有名為 `ticket_create` 的工具。與 `no_such_tool` 的回應完全相同；`ApiCallLogs` 記下 `GRANT_REQUIRED`。
2. **撤銷後使用舊 Token 也被拒絕：**取得 Token → 管理者撤銷 Grant → 同一張 Token 的 `tools/call` 得到 `[GRANT_REVOKED]` 使用授權已被撤銷。；`tools/list` 立即是 0 個。
3. **兩個並行呼叫不能突破最後一筆額度：**把今天已用次數設為「額度 - 1」，`tools/mcp-call.ps1 -Parallel 2` → 一個成功、一個 `QUOTA_EXCEEDED`；加碼 `-Parallel 20` 搶 10 次 → 剛好 10 個成功。
4. **反向驗證：**停權（`IsActive = 0`）→ `MEMBER_SUSPENDED`；Token 沒有 `mcp:tools` → `SCOPE_MISSING`；Rate Limit 5 → 第 6 次 429。

本單元結束時，Grant #4（王小明的課程查詢）已經撤銷；Grant #5（林小華）仍有效。

回顧：Token 說明「你是誰」，能不能用工具每次都看資料庫；規則寫在一個共用服務裡，清單與呼叫一致；額度用一個原子的 UPDATE 預留；從未獲准與不存在回應相同；撤銷時清快取，呼叫本身從不信任快取。下一單元（Section 12）做會員與 API 的全生命週期管理：停權／復權、版本更新與下架對授權的影響，並留下可追查的異動歷程。

延伸閱讀

- [ASP.NET Core 的速率限制中介軟體](#)：Fixed Window、分區、中介軟體順序。
- [EF Core ExecuteUpdate](#)：不先載入實體，直接送出 UPDATE。
- [MCP 規範：Tools 的錯誤處理 \(2025-11-25\)](#)：協定錯誤與 `isError`。

營運治理與交付

S12~S15

在後臺管理會員與 API 的完整生命週期，以數位儀表板與稽核查詢掌握平台狀況，接著以越權、CSRF、SSRF 與併發情境做整合測試，最後部署到 IIS 並完成 Connector 展示與成果驗收。

第 12 章 會員與API全生命週期管理

會員查詢、停權／復權、角色異動；已上架與已授權 API 查詢；版本更新、下架與授權影響；後臺會員管理、審核歷史與停用原因記錄

第 13 章 數位儀表板與稽核查詢

有效會員、已發布 API、待審案件、呼叫量、成功率、平均／P95 延遲、Top API 與錯誤分布；日期篩選、KPI 卡片、每日趨勢圖、API 排名、呼叫日誌與審核軌跡查詢

第 14 章 權限安全與整合測試

角色越權、跨會員資料存取、CSRF、SSRF、憑證遮罩、重複審核、版本變更與額度併發；以測試帳號執行端到端案例，檢查 RWD、表單錯誤、鍵盤導覽與中文可讀性

第 15 章 部署安裝與成果驗收

IIS、HTTPS、環境設定、資料庫 Migration、Data Protection 金鑰持久化、日誌、備份與復原；發布到教室主機、專用 SQL 帳號、正式測試網址與 OAuth、Connector 展示與成果驗收

第 12 章

會員與API全生命週期管理

會員查詢、停權／復權、角色異動；已上架與已授權 API 查詢；版本更新、下架與授權影響；後臺會員管理、審核歷史與停用原因記錄



課程時數 2 小時（120 分鐘）

當堂成果

後臺管理及可追查異動歷程

範例專案

範例/12_會員與API全生命週期管理

學習目標

上完這個單元，你能夠：

- 做出後臺「會員管理」：以 Email 或名稱查詢，查看會員的角色、狀態、使用授權、登錄的 API 與審核異動歷程。
- 實作停權／復權與管理者角色異動：原因必填、不能對自己操作、寫進 `AuditLogs`，並用 SecurityStamp 讓舊的登入狀態在 1 分鐘內失效。
- 說明停權如何同時擋住三個入口：網站登入、既有的網站登入狀態、已發出的 MCP Access Token。
- 走完一次 API 版本更新：v2 必須重新審核、重新映射工具、會員重新申請授權，v1 的授權不會自動轉移。
- 在下架前看到影響評估，並驗證下架後歷史資料（呼叫紀錄、審核紀錄、Grant、工具映射）全部保留。

先備知識

- 完成 Section 11： `ToolAccessService` 在每次 `tools/call` 檢查會員 `IsActive`、Grant 與版本狀態；撤銷與核准會清除 `tools/List` 的快取。
- Section 04 的登入流程：密碼正確但 `IsActive = false` 時顯示「此帳號已停權，請聯絡管理者。」；角色 `Admin` 由 `IdentitySeeder.AdminRole` 定義。
- Section 05~07 的流程：「建立新版本」複製上一版為草稿 → 送審 → 管理者核准發布；會員對某個**版本**申請使用 → 管理者核准產生 Grant。
- Section 06 的下架確認頁（ `/Admin/Apis/Retire/{id}`，需填原因）。
- 目前資料：`teacher` 擁有課程查詢 v1（版本 Id 2）；`member-b`（林小華）有 v1 的有效 Grant #5。

觀念說明（20 分鐘）

維運要回答的三個問題

平台上線之後，管理者每天會遇到：「這個帳號外流了，馬上讓它不能用」、「這支 API 要改版，舊的怎麼收」、「上週是誰把這個人的授權撤掉的、為什麼」。本單元把前面各單元做出來的資料，整理成能回答這三個問題的管理功能：

問題	本單元的功能	資料來源
讓一個帳號立刻不能用	會員管理：停權／復權	<code>AspNetUsers.IsActive</code> 、 <code>SecurityStamp</code>
API 改版與下架	新版本重新審核與授權；下架前的影響評估	<code>ApiVersions</code> 、 <code>ApiGrants</code> 、 <code>ToolDefinitions</code>
誰、何時、為什麼	會員詳細頁的「審核與異動歷程」、下架頁的審核歷程	<code>AuditLogs</code> 、 <code>ApprovalRecords</code>

停權要擋住三個入口

停權不是只改一個欄位。會員可能同時有三種「已經在用」的狀態，每一種都要被擋下：

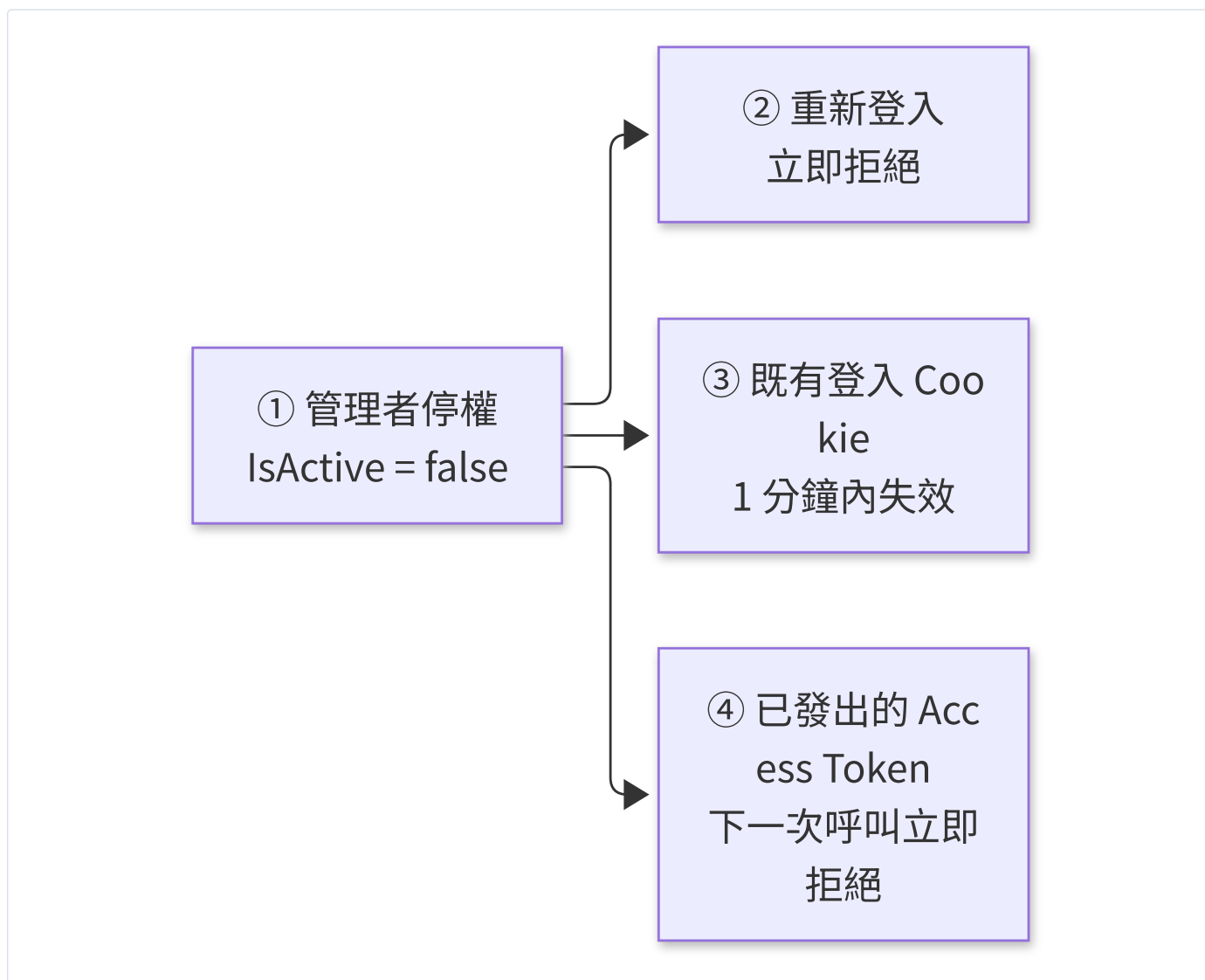


圖 12-1 停權對三個入口的影響

編號	誰在擋	本單元實測
①	<code>MemberAdminService.SuspendAsync</code> ：改 <code>IsActive</code> 、換 <code>SecurityStamp</code> 、寫 <code>AuditLogs</code> ，一次存檔	「已停權 b@demo.test：不能登入，網站的登入狀態 1 分鐘內失效，所有新的 MCP 呼叫立即拒絕。」
②	Section 04 的 <code>AccountController.Login</code> ：密碼正確後檢查 <code>IsActive</code>	「此帳號已停權，請聯絡管理者。」
③	Identity 的 <code>SecurityStamp</code> 驗證：登入 Cookie 裡記著當時的 Stamp，每隔 <code>ValidationInterval</code> 與資料庫比對一次，不同就登出	停權後立刻用舊 Cookie → 200；65 秒後 → 302 轉到登入頁
④	Section 11 的 <code>ToolAccessService</code> ：每次 <code>tools/call</code> 查 <code>IsActive</code>	停權前取得的 Token → <code>[MEMBER_SUSPENDE]</code> 會員已停權，不能呼叫工具。

③ 的間隔預設是 30 分鐘，本平台設成 1 分鐘：間隔越短，每分鐘多一次資料庫查詢；越長，停權後舊的網頁登入越久才失效。④ 不受這個間隔影響，因為 MCP 呼叫每次都查資料庫。

管理動作的共同規則

停權、復權、角色異動、撤銷授權、下架，都遵守同樣三條規則：

- 原因必填**：沒有原因的異動事後無法判斷對錯。原因寫進 `AuditLogs.Summary`，會員詳細頁看得到（大綱的「停用原因記錄」）。
- 不能對自己操作**：管理者不能停權自己、不能改自己的角色。避免誤操作把最後一位管理者鎖在門外，也避免有人替自己加權限。
- 狀態不對就拒絕**：已停權的再停權、已是管理者的再設為管理者，都回明確的訊息，不重複寫稽核。

角色異動同樣會換 `SecurityStamp`：被加入或移出管理者的人，舊的登入 Cookie 失效，重新登入後才拿到新角色。

版本更新：新版本是新的審核對象

Section 05 定下的原則是「URL、Method、Schema 任一項要改，就建立新版本並重新送審」。版本一變，後面每一層都是新的：



圖 12-2 課程查詢 v1 → v2 的升級流程

編號	為什麼不能沿用 v1	沒做這一步的結果（本單元實測）
① ②	v2 的端點或參數可能不同，要重新看過	v2 維持草稿，不會出現在目錄
③	工具名稱唯一， <code>courses_search</code> 仍屬於 v1；AI 可能已記住舊名稱與舊參數	v2 沒有工具，AI 叫不到
④	Grant 綁版本：會員同意的是 v1 的用途與資料範圍	林小華的 <code>tools/list</code> 只有 <code>courses_search</code> ；呼叫 <code>courses_search_v2</code> 得到 <code>-32602</code> 沒有名為 <code>courses_search_v2</code> 的工具。
⑤	兩版並存一段時間讓會員轉移，再下架舊版	下架後 v1 的 Grant 失去作用，但資料保留

下架不刪資料

下架只把版本狀態改成 `Retired`。之後：

資料	下架後	為什麼保留
<code>ApiCallLogs</code> （本例 52 筆）	保留，仍可查詢與統計	帳務、稽核、儀表板（Section 13）都要用
<code>ApprovalRecords</code> （本例 3 筆）	保留	當初為什麼核准，事後要查得到
<code>ApiGrants</code> （Grant #5）	狀態仍是 Active，但呼叫一律被擋	Grant 記的是「當時核准了什麼」，不是現在能不能用
<code>ToolDefinitions</code> （ <code>courses_search</code> ）	保留，但不再出現在 <code>tools/list</code>	舊紀錄的 <code>ToolName</code> 仍對得上

能不能呼叫由版本狀態決定（Section 09 的 `FindActiveToolAsync`、Section 08 的 Gateway 第一道檢查），所以不必去改 Grant。

限制與注意事項

- SecurityStamp 每 1 分鐘驗證一次，停權後網頁最慢 1 分鐘才被登出；需要立即失效就把間隔設成 0（每個請求都查資料庫）。
- 停權不撤銷 OAuth 已發出的 Token：Token 仍有效到期滿，只是每次呼叫都被 `MEMBER_SUSPENDED` 擋下。
- 會員詳細頁的歷程最多顯示 30 筆；完整查詢在 Section 13。
- 本單元沒有做「移轉授權到新版本」；新版本一律重新申請，是刻意的設計。

名詞對照

名詞	白話說明	在本課程中是
停權／復權	暫停／恢復一個帳號的所有使用	<code>AspNetUsers.IsActive</code>
SecurityStamp	帳號安全資訊的版本號，換了就讓舊的登入失效	<code>userManager.UpdateSecurityStampAsync</code>
ValidationInterval	多久比對一次 Cookie 裡的 SecurityStamp	1 分鐘
升版	建立新版本、重新審核與授權	課程查詢 v2
下架	版本不再提供使用，資料保留	<code>ApiVersionStatus.Retired</code>
影響評估	動手前先列出會受影響的會員、工具與紀錄	<code>RetireImpactDto</code>

教師示範（25 分鐘）

示範情境：林小華的帳號疑似外流，管理者停權後確認三個入口都被擋，事後復權；接著課程查詢從 v1 升級到 v2，林小華重新申請，最後下架 v1 並確認歷史資料都在。

示範步驟

1. 以管理者登入，側欄「會員管理」：



圖 12-3 會員管理清單

畫面重點：

- ①查詢框：輸入 Email 或顯示名稱的一部分（例如 **demo**），查詢以參數化 SQL 執行。
- ②角色與狀態：「系統管理員」是管理者；林小華已停權（這張圖在步驟 2 之後擷取）。
- ③有效授權：只算狀態 Active 且未到期的 Grant。
4. 點林小華進詳細頁，依序送出下列操作。本單元實測訊息：

停權，原因空白

在自己的詳細頁按停權

停權林小華（原因：課堂示範：帳號疑似外流，暫停使用）

再停權一次

✗ 停權必須填寫原因。

✗ 不能停權自己的帳號。

✓ 已停權 b@demo.test：不能登入，網站的登入狀態 1 分鐘內失效，所有新的 MCP 呼叫立即拒絕。

✗ 這位會員目前已是「停權」。



圖 12-4 會員詳細頁：狀態、使用授權、登錄的 API、審核與異動歷程

畫面重點：

1. ①狀態「停權」與「復權原因」表單：停權後按鈕自動變成復權，一樣要填原因。
2. ②使用授權：Grant #5 課程查詢 v1，版本狀態「已發布」、授權「有效」；停權不改 Grant，但呼叫會被擋。
3. ③登錄的 API：林小華自己登錄的包裹追蹤 v1（已退回）、v2（審核中）。
4. ④審核與異動歷程：最上面一列 **SuspendMember**，執行者「系統管理員」，內容含停權原因；下面是他的申請被核准、駁回的紀錄。
5. 驗證三個入口（停權前林小華已登入網站、並已取得 Access Token）：

MCP 呼叫（停權前取得的 Token）	[MEMBER_SUSPENDED] 會員已停權，不能呼叫工具。
重新登入	此帳號已停權，請聯絡管理者。
停權前的登入 Cookie（立刻）	GET /Member/Access → 200
停權前的登入 Cookie（65 秒後）	GET /Member/Access → 302 /Account/Login?ReturnUrl=%2FMember%2FAccess

4. 角色異動與復權：

在自己的詳細頁移除管理者	✗ 不能變更自己的角色，請由另一位管理者處理。
教師示範帳號 設為管理者	✓ 已設為管理者：teacher@demo.test（對方重新登入後生效）。
教師示範帳號 移除管理者	✓ 已移除管理者角色：teacher@demo.test（對方重新登入後生效）。
林小華 復權（原因：課堂示範結束，恢復使用）	✓ 已復權 b@demo.test。

5. 升版。以 **teacher** 登入，在「我的 API」對課程查詢按「建立新版本」，再送審；管理者核准並在「MCP 工具映射」把 v2 映射為 **courses_search_v2**。實測：

NewVersion → /Member/Apis/Edit/19	✓ 已建立新版本草稿，內容複製自上一版。
Submit	✓ 已送審 課程查詢 v2，等待管理者審核。
管理者核准案件 #12	✓ 已核准並發布 課程查詢 v2。
映射工具	✓ 已將 課程查詢 v2 映射為工具 courses_search_v2。

6. 林小華用手上的 Token 試 v2，再申請、等核准：

tools/list	工具 (1 個) : courses_search
呼叫 courses_search_v2	協定錯誤 -32602: 沒有名為 courses_search_v2 的工具。
林小華申請 v2	✓ 已送出 課程查詢 v2 的使用申請。
管理者核准申請 #15	✓ 已核准, 並建立 60 天、每日 300 次的使用授權。
tools/list (同一張 Token)	工具 (2 個) : courses_search, courses_search_v2
呼叫 courses_search_v2	SUCCESS CorrelationId: b4dca366f90c4718

7. 下架 v1。管理者在「API 管理」點課程查詢 v1 的下架：

MCP Server API 平台

API 目錄

會員中心

管理後臺

系統管理員

登出

管理後臺

總覽

API 管理

資料庫概況

上架審核

使用申請審核

Grant 管理

API 測試台

MCP 工具映射

Connector 登錄

會員管理

下架 課程查詢 v1

目前狀態：

✓ 已發布

下架後，這個版本不再出現在公開目錄，新的使用申請與呼叫一律拒絕；申請、授權、呼叫紀錄全部保留。

下架影響

- 有效授權 1 筆會失去作用：林小華 (Grant #5)
- MCP 工具：courses_search 會從 tools/list 消失，呼叫一律拒絕
- 保留的歷史：呼叫紀錄 52 筆、審核紀錄 3 筆 (不刪除)
- 已發布的新版本 v2：舊授權不會自動轉移，會員要對新版本重新申請。

下架原因 (必填)

確認下架

審核歷程

2026-09-20 16:00 送審 (教師示範帳號)

案件 #2

2026-09-20 18:00 核准發布 (系統管理員)

案件 #2：規格完整，端點為課堂允許清單內的主機。

© 2026 MCP Server API 平台 | 環境檢查

圖 12-5 下架前的影響評估

畫面重點：

- 1. ①有效授權 1 筆會失去作用：林小華 (Grant #5)：王小明的 Grant #4 在 Section 11 已撤銷，不算在內。
- 2. ②MCP 工具 courses_search 會從 tools/list 消失。
- 3. ③保留的歷史：呼叫紀錄 52 筆、審核紀錄 3 筆：3 筆是 v1 的上架核准與兩件使用申請的核准。
- 4. ④已發布的新版本 v2：提醒管理者舊授權不會自動轉移。

填原因「已由 v2 取代」確認下架。實測：

下架前：v1 呼叫紀錄 52 筆；Grant #5 = Active	
下架	✓ 已下架 課程查詢 v1。
tools/list	工具 (1 個) : courses_search_v2
呼叫 courses_search (v1)	協定錯誤 -32602: 沒有名為 courses_search 的工具。
呼叫 courses_search_v2	SUCCESS
下架後：v1 呼叫紀錄 52 筆；Grant #5 = Active；ToolDefinitions 中 v1 的工具列仍在	

學員實作（60 分鐘）

範例專案：範例/12_會員與API全生命週期管理/McpPlatform/（從 Section 11 複製）。本單元新增 Core 的 `Members/MemberAdminService.cs`、Infrastructure 的 `Members/IdentityMemberAdminStore.cs`、Web 的後臺 `MembersController` 與兩個頁面；修改：`PublicationContracts.cs`（`RetireImpactDto`）、`PublicationService`、`EfPublicationRepository`、下架確認頁、後臺側欄、`Program.cs`。本單元沒有新的 Migration。

步驟一 會員管理服務（20 分鐘）

目的：停權／復權的規則寫在 Core，資料存取交給 Infrastructure。

```
// 範例/12_會員與API全生命週期管理/McpPlatform/src/McpPlatform.Core/Members/MemberAdminService.cs (ChangeActiveAsync)
private async Task<MemberResult> ChangeActiveAsync(string adminId, string userId, bool active, string? reason, CancellationToken ct)
{
    var verb = active ? "復權" : "停權";
    if (string.IsNullOrEmpty(reason)) return MemberResult.Refused($"{{verb}}必須填寫原因。"); // ① 原因必填
    if (adminId == userId) return MemberResult.Refused($"不能{{verb}}自己的帳號。"); // ② 不能鎖住自己
    var details = await store.GetDetailsAsync(userId, clock.UtcNow(), ct);
    if (details is null) return MemberResult.Missing();
    if (details.Member.IsActive == active) return MemberResult.Refused($"這位會員目前已是「{{active ? "啟用" : "停權"}}」。");

    await store.SetActiveAsync(userId, active, Audit(adminId, active ? "ReactivateMember" : "SuspendMember", userId,
        $"{{verb}} {{details.Member.Email}}:{{reason.Trim()}}", ct); // ③ 狀態、SecurityStamp、稽核一起存
    toolAccessCache.InvalidateMember(userId); // ④ tools/List 立即反映
    return MemberResult.Ok(active
        ? $"已復權 {{details.Member.Email}}。"
        : $"已停權 {{details.Member.Email}}：不能登入，網站的登入狀態 1 分鐘內失效，所有新的 MCP 呼叫立即拒絕。");
}
```

實際寫入在 Infrastructure，一律經過 `UserManager`：

```
// 範例/12_會員與API全生命週期管理/McpPlatform/src/McpPlatform.Infrastructure/Members/IdentityMemberAdminStore.cs (SetActiveAsync)
public async Task SetActiveAsync(string userId, bool active, AuditLog audit, CancellationToken ct)
{
    var user = await users.FindByIdAsync(userId) ?? throw new InvalidOperationException("找不到會員。");
    user.IsActive = active;
    db.AuditLogs.Add(audit); // 同一個 DbContext：下一行存檔時一起寫入
    var result = await users.UpdateSecurityStampAsync(user); // 換 SecurityStamp 並存檔：舊的登入 Cookie 失效
    if (!result.Succeeded) throw new InvalidOperationException(string.Join(" ", result.Errors.Select(e => e.Description)));
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	原因空白就拒絕	原因是日後判斷這次停權是否正確的唯一依據
②	比對操作者與對象	管理者停權自己會立刻被登出，而且沒有人能替他復權
<code>IsActive == active</code>	狀態沒有改變就拒絕	不寫一筆沒有意義的稽核
③ <code>SetActiveAsync</code>	改 <code>IsActive</code> 、加一筆 <code>AuditLogs</code> 、呼叫 <code>UpdateSecurityStampAsync</code>	<code>userManager</code> 與 <code>db</code> 是同一個 <code>AppDbContext</code> ， <code>UpdateSecurityStampAsync</code> 存檔時把三項一起寫入，不會只成功一半
④	清掉這位會員的 Grant 快取	停權後 <code>tools/List</code> 本來就會回空清單（Section 11 查 <code>IsActive</code> ），清快取讓復權後也立即恢復
<code>Audit(...)</code>	<code>ResourceType = "Member"</code> 、 <code>ResourceId = 會員 Id</code>	會員詳細頁以這兩個欄位找出「別人對他做的事」

角色異動 `SetAdminAsync` 的規則相同（原因必填、不能改自己、狀態沒變就拒絕），寫入時用 `AddToRoleAsync`／`RemoveFromRoleAsync`，之後同樣換 `SecurityStamp`。

預期結果：示範步驟 2、4 的訊息；`AuditLogs` 多出 `SuspendMember`、`ReactivateMember`、`GrantAdminRole`、`RevokeAdminRole` 各一筆。

步驟二 SecurityStamp 驗證間隔（5 分鐘）

目的：讓停權後舊的網頁登入 1 分鐘內失效。

```
// 範例/12_會員與API全生命週期管理/McpPlatform/src/McpPlatform.Web/Program.cs (Section 12)
// Section 12: 會員管理。SecurityStamp 每 1 分鐘檢查一次，停權或角色異動後舊的登入狀態最慢 1 分鐘失效。
builder.Services.AddScoped<MemberAdminService>();
builder.Services.AddScoped<IMemberAdminStore, IdentityMemberAdminStore>();
builder.Services.Configure<SecurityStampValidatorOptions>(options => options.ValidationInterval = TimeSpan.FromMinutes(1));
```

設定	預設	本平台	取捨
<code>ValidationInterval</code>	30 分鐘	1 分鐘	越短，停權越快生效；每位登入中的使用者每個間隔多一次資料庫查詢

預期結果：示範步驟 3：停權當下舊 Cookie 仍是 200，65 秒後被轉到登入頁。

步驟三 會員清單與詳細頁（15 分鐘）

目的：查詢會員，並在一頁內看到他的授權、API 與歷程。

```
// 範例/12_會員與API全生命週期管理/McpPlatform/src/McpPlatform.Infrastructure/Members/IdentityMemberAdminStore.cs (SearchAsync)
public async Task<IReadOnlyList<MemberRowDto>> SearchAsync(string? keyword, DateTimeOffset now, CancellationToken ct)
{
    var query = db.Users.AsNoTracking();
    if (keyword is not null)
        query = query.Where(u => u.Email!.Contains(keyword) || u.DisplayName.Contains(keyword)); // 參數化查詢，不會 SQL Injection
    return await query
        .OrderBy(u => u.Email)
        .Select(u => new MemberRowDto(u.Id, u.Email!, u.DisplayName, u.IsActive, AdminIds.Contains(u.Id),
            db.ApiGrants.Count(g => g.UserId == u.Id && g.Status == GrantStatus.Active && g.ExpiresAtUtc > now),
            u.CreatedAtUtc))
        .ToListAsync(ct);
}
```

詳細頁的歷程查詢要同時找出「他做的事」與「別人對他做的事」：

```
// 範例/12_會員與API全生命週期管理/McpPlatform/src/McpPlatform.Infrastructure/Members/IdentityMemberAdminStore.cs (GetDetailsAsync 節錄)
var history = await (from a in db.AuditLogs.AsNoTracking()
    join u in db.Users on a.ActorId equals u.Id
    where a.ActorId == userId
        || (a.ResourceType == "Member" && a.ResourceId == userId)
        || (a.ResourceType == "ApiAccessRequest" && requestIds.Contains(a.ResourceId))
        || (a.ResourceType == "ApiGrant" && grantIds.Contains(a.ResourceId))
    orderby a.OccurredAtUtc descending
    select new MemberHistoryDto(a.OccurredAtUtc, u.DisplayName, a.Action, a.Summary))
    .Take(30).ToListAsync(ct);
```

程式說明：

片段	做了什麼	為什麼這樣寫
<code>Contains(keyword)</code>	EF Core 翻譯成 <code>LIKE @p</code> ，參數化	使用者輸入不會被當成 SQL
<code>AdminIds.Contains(u.Id)</code>	<code>AdminIds</code> 是 <code>UserRoles</code> join <code>Roles</code> 取出管理者 Id 的子查詢	清單一次查完，不必對每位會員各呼叫一次 <code>IsInRoleAsync</code>
有效授權 <code>Count(...)</code>	只算 Active 且未到期	與 Section 11 判斷「能不能用」的條件一致
歷程四個條件	他自己做的事、對他這個人的操作、對他的申請的審核、對他的授權的撤銷	<code>AuditLogs.ResourceId</code> 是字串，所以先把申請與 Grant 的 Id 轉成字串清單
<code>Take(30)</code>	最多 30 筆	詳細頁只看最近的；完整查詢在 Section 13

Controller 的四個 POST（`Suspend`、`Reactivate`、`SetAdmin`）都轉回詳細頁，成功放 `TempData["Notice"]`、拒絕放 `TempData["Error"]`（Section 09 的紅色 X）。`MembersController` 在 `Admin` Area，只有管理者進得來。

預期結果：圖 12-3、圖 12-4。

步驟四 下架影響評估（20 分鐘）

目的：下架前列出會受影響的授權、工具與會保留的紀錄。


```
// 範例/12_會員與API全生命週期管理/McpPlatform/src/McpPlatform.Infrastructure/Publishing/EfPublicationRepository.cs (GetRetireImpactAsync)
public async Task<RetireImpactDto> GetRetireImpactAsync(int versionId, DateTimeOffset now, CancellationToken ct)
{
    var holders = await (from g in db.ApiGrants.AsNoTracking()
                        join u in db.Users on g.UserId equals u.Id
                        where g.ApiVersionId == versionId && g.Status == GrantStatus.Active && g.ExpiresAtUtc > now
                        orderby u.DisplayName
                        select $"{u.DisplayName} (Grant #{g.Id}) ").ToListAsync(ct);
    var toolName = await db.ToolDefinitions.Where(t => t.ApiVersionId == versionId).Select(t => t.ToolName).SingleOrDefaultAsync(ct);
    var calls = await db.ApiCallLogs.CountAsync(l => l.ApiVersionId == versionId, ct);
    var approvals = await db.ApprovalRecords.CountAsync(a =>
        (a.PublicationRequest != null && a.PublicationRequest.ApiVersionId == versionId)
        || (a.AccessRequest != null && a.AccessRequest.ApiVersionId == versionId), ct);
    var newer = await (from v in db.ApiVersions
                    join self in db.ApiVersions on v.ApiDefinitionId equals self.ApiDefinitionId
                    where self.Id == versionId && v.Id != versionId && v.Status == ApiVersionStatus.Published
                    orderby v.Id descending
                    select v.Version).FirstOrDefaultAsync(ct);
    return new RetireImpactDto(holders, toolName, calls, approvals, newer);
}
```

程式說明：

片段	做了什麼	為什麼這樣寫
<code>holders</code>	這個版本上 Active 且未到期的 Grant，連同會員名稱	下架後這些會員就不能呼叫了，管理者應該先通知他們
<code>toolName</code>	這個版本映射的工具名稱	AI 應用會發現工具消失
<code>calls</code> 、 <code>approvals</code>	呼叫紀錄與審核紀錄筆數	明確告訴管理者「這些不會被刪除」
<code>newer</code>	同一個 API 其他已發布版本中 Id 最大的一個	有新版本就提醒「舊授權不會自動轉移」；沒有就提醒「會員將無版本可用」
<code>select \$"..."</code>	組成「林小華 (Grant #5)」這樣的顯示字串	頁面直接列出，不必再查一次會員名稱

`Admin/ApiController` 的下架確認頁 GET 多一行 `ViewData["Impact"] = await publication.GetRetireImpactAsync(id, ct);`，`Retire.cshtml` 把它列成「下架影響」四點（圖 12-5）。

預期結果：圖 12-5 的四點；下架後 `ApiCallLogs`、`ApprovalRecords`、`ApiGrants`、`ToolDefinitions` 的筆數都不變。

檢查點

- ☐ 停權：原因空白、對自己、重複停權都被拒；成功後 MCP 呼叫 `MEMBER_SUSPENDED`、重新登入被拒、舊 Cookie 1 分鐘內失效
- ☐ 復權後可以重新登入、MCP 呼叫恢復
- ☐ 角色異動寫入 `AuditLogs`，不能改自己的角色
- ☐ v2 需重新審核、重新映射工具、會員重新申請後才叫得到
- ☐ 下架前看到影響評估；下架後歷史資料筆數不變

常見錯誤

以下都是撰寫本單元時實際遇到的情況。

症狀	原因	處理
停權後，會員原本開著的網頁仍然可以操作（ <code>GET /Member/Access</code> → 200）	登入 Cookie 只在每個 <code>ValidationInterval</code> 比對一次 SecurityStamp；預設 30 分鐘	本平台設為 1 分鐘（實測 65 秒後 302 到登入頁）；MCP 呼叫不受影響，每次都查 <code>IsActive</code>
角色異動後，對方畫面上的選單沒有變	角色是登入當時寫進 Cookie 的	異動時換 SecurityStamp，對方 1 分鐘內被登出、重新登入後套用
升版後會員說「新版本的工具叫不到」： <code>-32602 沒有名為 courses_search_v2 的工具。</code>	Grant 綁版本，v1 的授權不涵蓋 v2	會員對 v2 送出使用申請，管理者核准；核准後同一張 Token 立即看得到
想把 v2 也映射成 <code>courses_search</code> ，被拒：「工具名稱 <code>courses_search</code> 已被使用，或這個版本已經映射過。」（Section 09）	工具名稱唯一，舊名稱仍屬於 v1	新版本用新名稱（本例 <code>courses_search_v2</code> ）；舊工具隨 v1 下架從清單消失
測試腳本 <code>Invoke-WebRequest -MaximumRedirection 0</code> 印出「已超過重新導向次數上限」	PowerShell 7 對 302 回應丟出錯誤訊息，但請求其實已經送出並完成	加 <code>-ErrorAction SilentlyContinue</code> 隱藏這個訊息，再讀轉址後的頁面確認結果

單元驗收（15 分鐘）

驗收標準：會員停權阻擋新呼叫；API 下架不刪除歷史資料；新版本需重新審核及授權。

驗收方式：

1. **會員停權阻擋新呼叫：**林小華先取得 Token → 管理者停權（填原因）→ 同一張 Token 呼叫得到 `[MEMBER_SUSPENDED]` 會員已停權，不能呼叫工具。；重新登入顯示「此帳號已停權，請聯絡管理者。」；會員詳細頁的歷程有 `SuspendMember` 與原因。
2. **API 下架不刪除歷史資料：**下架課程查詢 v1 前後比對：`ApiCallLogs` 52 筆 → 52 筆、Grant #5 仍在、`ToolDefinitions` 中 v1 的工具列仍在；`courses_search` 從清單消失，呼叫得到 `-32602`。
3. **新版本需重新審核及授權：**v2 建立後是草稿，需送審核准才發布；發布後林小華的舊 Token 只看到 v1 的工具，對 v2 申請並獲准後才出現 `courses_search_v2`。
4. **反向驗證：**管理者停權自己、改自己的角色、原因空白，都被拒絕。

本單元結束時：課程查詢 v1 已下架、v2 已發布（工具 `courses_search_v2`）；林小華有 v2 的有效授權並已復權。

回顧：停權一次擋住登入、既有網頁、既有 Token 三個入口；所有管理動作原因必填、寫稽核、不能對自己操作；版本是審核與授權的單位，升版就重新走一次；下架只改狀態，資料全部保留。下一單元（Section 13）用這些累積下來的紀錄做儀表板與稽核查詢。

延伸閱讀

- [ASP.NET Core Identity : SecurityStampValidator](#)：`ValidationInterval` 與登出行為。
- [UserManager.UpdateSecurityStampAsync](#)：何時需要換 SecurityStamp。

第 13 章

數位儀表板與稽核查詢

有效會員、已發布 API、待審案件、呼叫量、成功率、平均／P95 延遲、Top API 與錯誤分布；日期篩選、KPI 卡片、每日趨勢圖、API 排名、呼叫日誌與審核軌跡查詢



課程時數 2 小時（120 分鐘） 當堂成果 真實資料 KPI、圖表與日誌查詢 範例專案 範例/13_數位儀表板與稽核查詢

學習目標

上完這個單元，你能夠：

- 寫出指標的**精確定義**：什麼算「工具執行」、什麼算「授權拒絕」、成功率的分子分母、P95 用哪一種算法，並用 SQL 驗證儀表板的每個數字。
- 用「臺灣日期」處理日期篩選：資料存 UTC，查詢邊界是臺灣時間 0 點，零筆資料顯示「無資料」而不是 0%。
- 做出後臺儀表板：平台現況與區間呼叫的 KPI 卡片、Chart.js 每日趨勢圖（附表格替代）、API 排名、錯誤分布。
- 做出呼叫紀錄查詢，能用 CorrelationId 追查單筆失敗並看到錯誤代碼的中文說明；做出稽核軌跡查詢。

先備知識

- Section 08 起每次呼叫寫一筆 `ApiCallLogs` (`CalledAtUtc` 、 `Outcome` 、 `DurationMs` 、 `ErrorCode` 、 `CorrelationId`) ; Section 11 起授權拒絕也寫一筆 (`GRANT_*` 、 `SCOPE_MISSING` 、 `MEMBER_SUSPENDED` 、 `QUOTA_EXCEEDED`) 。
- 各單元的管理動作都寫 `AuditLogs` (`ActorId` 、 `Action` 、 `ResourceType` 、 `ResourceId` 、 `Summary`) 。
- Section 12 的 `tools/mcp-call.ps1` 與 `tools/oauth-flow.ps1` 。
- 知道 `GROUP BY` 、 `COUNT` 、 `AVG` ; 知道 JSON 與 `<canvas>` 是什麼。

觀念說明（20 分鐘）

先定義，再畫圖

儀表板最常見的錯誤不是圖畫錯，而是**同一個名詞，每個人算法不同**。「今天呼叫了幾次？」要不要算被額度擋下的？「成功率」的分母包不包括沒授權的人亂猜？大綱把定義寫死了，本平台照做：

指標	定義	為什麼
授權拒絕	<code>ErrorCode</code> 是 <code>GRANT_REQUIRED</code> 、 <code>GRANT_REVOKED</code> 、 <code>GRANT_EXPIRED</code> 、 <code>GRANT_NOT_YET_VALID</code> 、 <code>SCOPE_MISSING</code> 、 <code>MEMBER_SUSPENDED</code> 、 <code>QUOTA_EXCEEDED</code> 其中之一	Section 11 在送進 Gateway 之前就擋下，工具根本沒有執行
工具執行	其餘所有紀錄（成功、參數錯誤、上游失敗、Gateway 拒絕）	Gateway 接受了這次呼叫，開始檢查或送出
成功率	成功的工具執行 ÷ 工具執行總數	授權拒絕是「不該執行」，不是「執行失敗」，混在一起會讓 API 看起來很不穩
平均／P95 延遲	工具執行的 <code>DurationMs</code>	授權拒絕耗時接近 0，會把平均拉低
零筆	顯示「無資料」	0 筆的成功率不是 0%，是沒有意義

P95 是「95% 的呼叫都在這個時間內完成」。本平台用**最近排名法**：由小到大排序，取第 $[0.95 \times n]$ 筆。它等於 SQL Server 的 `PERCENTILE_DISC(0.95)`；另一個函式 `PERCENTILE_CONT` 會在兩筆之間內插，本單元實測同一批資料：`PERCENTILE_DISC` = 4081、`PERCENTILE_CONT` = 4067.5。定義不寫清楚，兩個人對同一批資料會報出不同的 P95。

時區：存 UTC，看臺灣日期

`CalledAtUtc` 存的是 UTC。管理者說「9 月 26 日」指的是**臺灣時間** 9/26 00:00~23:59，也就是 UTC 9/25 16:00~9/26 16:00：

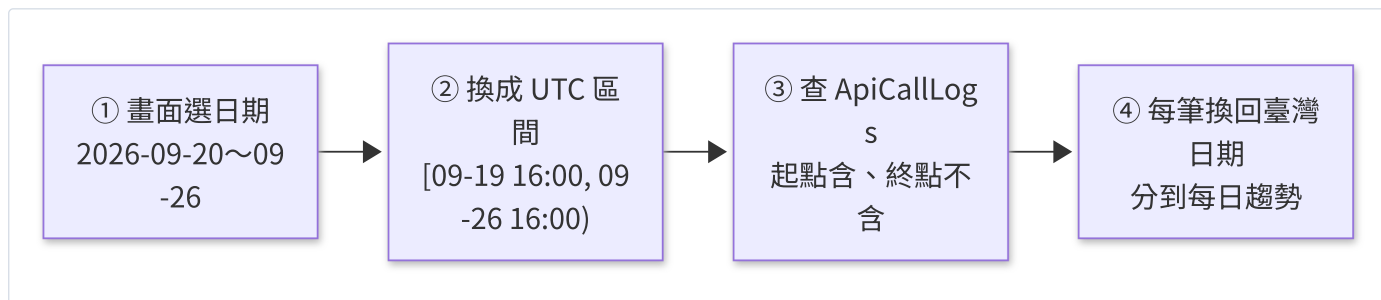


圖 13-1 日期篩選的時區換算

編號	做了什麼	對應程式碼
①	沒選日期時預設「今天往前 7 天」；起訖顛倒自動對調；最長 92 天	<code>DashboardService.NormalizeRange</code>
②	起日臺灣 0 點、迄日隔天臺灣 0 點，換成 UTC	<code>TaipeiCalendar.StartUtc</code>
③	<code>≥ 起點 AND < 終點</code> ：半開區間，相鄰兩天的查詢不會重複算到剛好 0 點的那一筆	<code>EfDashboardReader.ListCallsAsync</code>
④	每筆紀錄換成臺灣日期，才分得出「哪一天」	<code>TaipeiCalendar.DateOf</code>

如果直接用 UTC 日期分組，臺灣早上 8 點以前的呼叫都會被算到「前一天」。自己用 SQL 驗算每日數字時，要用 `CAST(SWITCHOFFSET(CalledAtUtc, '+08:00') AS date)` 分組。

儀表板上有什麼

區塊	內容	是否受日期篩選影響
平台現況	有效會員、已發布 API 版本、待審上架、待審使用申請	否（現在的狀態）
區間內的呼叫	工具執行、成功率（附「成功／總數」）、平均延遲、P95 延遲、授權拒絕	是
每日趨勢	成功、失敗、授權拒絕的每日堆疊長條圖；附表格	是
API 排名	工具執行次數前 5 名與各自的成功率	是
錯誤分布	每個錯誤代碼的次數、類別；點代碼進入呼叫紀錄查詢	是

追查單筆：CorrelationId

會員回報「剛剛那次呼叫失敗了」時，他手上有的是 MCP 回應裡的 `CorrelationId: 2cdbf7696a4a4a03`。管理者在「呼叫紀錄」輸入這串，就看到：時間、會員、Connector、API 與工具、用了哪一筆 Grant、結果、HTTP 狀態、耗時、錯誤代碼與中文說明（例如 `INVALID_ARGUMENTS`：「參數不符合版本的 JSON Schema，沒有送出請求。請 Client 修正參數。」）。平台不存請求與回應內容（Section 08 的原則），但這些欄位已經足以判斷問題出在會員、Client、平台設定還是上游。

限制與注意事項

- 儀表板把區間內的呼叫紀錄（只取 5 個欄位）讀進記憶體再計算，最長 92 天。課堂與小型平台沒有問題；每天數十萬筆以上時，應改成 SQL 彙總或每日預先統計的表格。
- 「有效會員」是 `IsActive = 1` 的帳號數，包含管理者。
- 呼叫紀錄與稽核軌跡一次最多顯示 200 筆，最新的在前。
- 圖表只是呈現；數字以 KPI 卡片與表格為準，Chart.js 沒載入時頁面仍然可用。

名詞對照

名詞	白話說明	在本課程中是
KPI	關鍵指標，一眼就要看到的數字	儀表板上的卡片
工具執行	Gateway 接受並處理的一次呼叫	非授權拒絕的 <code>ApiCallLogs</code>
授權拒絕	送進 Gateway 前就被擋下	<code>CallCategories.AuthorizationDenials</code>
P95	95% 的呼叫在此時間內完成	最近排名法，等於 <code>PERCENTILE_DISC</code>
半開區間	含起點、不含終點	<code>>= 起點 AND < 終點</code>
稽核軌跡	誰、何時、對什麼、做了什麼	<code>AuditLogs</code> 查詢頁

教師示範（25 分鐘）

示範情境：打開儀表板看近 7 天的狀況，用 SQL 驗證每一個數字；做一次成功與一次失敗的呼叫，看數字變化；用失敗那次的 CorrelationId 追查；最後查稽核軌跡。

示範步驟

1. 以管理者登入，側欄「儀表板」（預設 2026-09-20~2026-09-26）：



圖 13-2 儀表板：KPI、每日趨勢、API 排名、錯誤分布

畫面重點：

1. ①日期篩選：兩個日期欄位與說明「臺灣時間 2026-09-20 00:00 ~ 2026-09-26 23:59；最長 92 天」。
2. ②平台現況：有效會員 4、已發布 API 版本 8、待審上架 1、待審使用申請 1。這四個數字不受日期影響。
3. ③區間內的呼叫：工具執行 59、成功率 54.2% (32 / 59)、平均 578 ms、P95 4081 ms、授權拒絕 29 (「不計入成功率」)。
4. ④每日趨勢：9/26 的長條最高，三種顏色分別是成功、失敗、授權拒絕；下方「以表格檢視每日數字」可展開看確切數字。
5. ⑤API 排名與錯誤分布：課程查詢 v1 執行最多；錯誤分布第一名是 `QUOTA_EXCEEDED` 20 次 (Section 11 的並行測試)，類別是「授權拒絕」。
6. 用 SQL 驗證同一個區間 (在 SSMS 執行)：

```
-- 在 MCPServerManag 執行；只讀。區間：臺灣時間 2026-09-20 00:00 ~ 2026-09-27 00:00 (不含)
DECLARE @from datetimeoffset = TODATETIMEOFFSET(CAST('2026-09-20' AS datetime2), '+08:00');
DECLARE @to datetimeoffset = TODATETIMEOFFSET(CAST('2026-09-27' AS datetime2), '+08:00');
SELECT COUNT(*) AS 工具執行, SUM(CASE WHEN Outcome = 'Success' THEN 1 ELSE 0 END) AS 成功,
       ROUND(AVG(CAST(DurationMs AS float)), 0) AS 平均ms
FROM ApiCallLogs
WHERE CalledAtUtc >= @from AND CalledAtUtc < @to
      AND (ErrorCode IS NULL OR ErrorCode NOT IN ('GRANT_REQUIRED', 'GRANT_REVOKED', 'GRANT_EXPIRED',
        'GRANT_NOT_YET_VALID', 'SCOPE_MISSING', 'MEMBER_SUSPENDED', 'QUOTA_EXCEEDED'));
```

呼叫前實測兩邊都是：工具執行 57、成功 31 (54.4%)、平均 597 ms；`PERCENTILE_DISC(0.95)` 為 4081；授權拒絕 29。

3. 以林小華做兩次呼叫（`tools/mcp-call.ps1`）：`courses_search_v2 {"level":"進階"}` 成功、`{"level":"高手"}` 得到 `INVALID_ARGUMENTS`。重新整理儀表板：

	呼叫前	呼叫後
工具執行	57	59 (+2)
成功	31 (54.4%)	32 (54.2%)
平均延遲	597 ms	578 ms (新增的兩筆耗時很短，平均下降)
授權拒絕	29	29

再用 SQL 查一次，數字相同。

4. 點錯誤分布的 `INVALID_ARGUMENTS`，進入呼叫紀錄；或直接在 CorrelationId 欄輸入剛才那次失敗的代碼：

MCP Server API 平台

API 目錄 會員中心 管理後臺

系統管理員 登出

管理後臺

總覽

儀表板

API 管理

資料庫概況

上架審核

使用申請審核

Grant 管理

API 測試台

MCP 工具映射

Connector 登錄

會員管理

呼叫紀錄

稽核軌跡

呼叫紀錄

起日

迄日

會員

工具名稱

錯誤代碼

CorrelationId

2026/09/20

2026/09/26

2cdbf7696a4a

查詢

有 CorrelationId 時只依它查詢，忽略其他條件；最多顯示 200 筆。

追查 2cdbf7696a4a03

時間 (臺灣)

2026-09-26 21:20

會員 / Client

林小華 / classroom-mcp-client

API / 工具

課程查詢 v2 / courses_search_v2

使用的 Grant

#7

結果

InvalidInput · HTTP — · 0 ms

錯誤代碼

INVALID_ARGUMENTS：參數不符合版本的 JSON Schema，沒有送出請求。請 Client 修正參數。

時間	會員	API／工具	結果	耗時	CorrelationId
2026-09-26 21:20	林小華 classroom-mcp-client	課程查詢 v2 courses_search_v2	InvalidInput INVALID_ARGUMENTS	0 ms	2cdbf7696a4a03

© 2026 MCP Server API 平台 | 環境檢查

圖 13-3 呼叫紀錄：追查 2cdbf7696a4a03

畫面重點：

- 1. ①查詢條件：有 CorrelationId 時只依它查詢，日期等其他條件都忽略（說明文字寫在查詢鈕旁）。
- 2. ②追查卡片：林小華／classroom-mcp-client、課程查詢 v2／courses_search_v2、使用的 Grant #7、`InvalidInput`、HTTP —、0 ms。
- 3. ③錯誤代碼說明：`INVALID_ARGUMENTS`：參數不符合版本的 JSON Schema，沒有送出請求。請 Client 修正參數。HTTP 「—」與 0 ms 也印證了「沒有送出請求」。
- 4. 側欄「稽核軌跡」，關鍵字輸入 `b@demo.test`：



圖 13-4 稽核軌跡：以關鍵字找出林小華的停權與復權

畫面重點：

1. ①動作下拉選單：選項來自 `AuditLogs` 裡實際出現過的動作（`SuspendMember`、`RevokeGrant`、`RegisterConnector` …）。
2. ②兩筆結果：21:05 `SuspendMember`、21:06 `ReactivateMember`，執行者都是系統管理員，資源 `Member #member-b`，內容含原因。
3. 反向示範：把日期設為 2026-09-21（那天沒有任何呼叫），五張呼叫 KPI 顯示「0、無資料、無資料、無資料、0」，API 排名與錯誤分布顯示「無資料」。把起訖顛倒輸入（9/26～9/01），頁面自動換成 9/01～9/26。

學員實作（60 分鐘）

範例專案：範例/13_數位儀表板與稽核查詢/McpPlatform/（從 Section 12 複製）。本單元新增 Core 的 Dashboard/（DashboardService.cs、LogSearchService.cs）、Infrastructure 的 Dashboard/EfDashboardReader.cs、Web 的 DashboardController、LogsController 與三個頁面、wwwroot/js/dashboard.js、wwwroot/Lib/chart.js/（Chart.js 4.5.1，含授權檔）；修改後臺側欄與 Program.cs。本單元沒有新的 Migration。

步驟一 指標定義與日期邊界（15 分鐘）

目的：把圖 13-1 與指標定義寫成程式。

```
// 範例/13_數位儀表板與稽核查詢/McpPlatform/src/McpPlatform.Core/Dashboard/DashboardService.cs (TaipeiCalendar)
public static class TaipeiCalendar
{
    private static readonly TimeZoneInfo Taipei = TimeZoneInfo.FindSystemTimeZoneById("Asia/Taipei");

    public static DateOnly DateOf(DateTimeOffset utc) => DateOnly.FromDateTime(TimeZoneInfo.ConvertTime(utc, Taipei).DateTime);

    /// <summary>臺灣時間 date 00:00 對應的 UTC 時刻，例如 2026-09-26 → 2026-09-25 16:00 UTC。</summary>
    public static DateTimeOffset StartUtc(DateOnly date) =>
        new(TimeZoneInfo.ConvertTimeToUtc(date.ToDateTime(TimeOnly.MinValue), Taipei), TimeSpan.Zero);
}
```

```
// 範例/13_數位儀表板與稽核查詢/McpPlatform/src/McpPlatform.Core/Dashboard/DashboardService.cs (CallCategories)
public static class CallCategories
{
    public static readonly IReadOnlySet<string> AuthorizationDenials = new HashSet<string>
    {
        AccessErrors.GrantRequired, AccessErrors.GrantRevoked, AccessErrors.GrantExpired, AccessErrors.GrantNotYetValid,
        AccessErrors.ScopeMissing, AccessErrors.MemberSuspended, AccessErrors.QuotaExceeded
    };

    public static bool IsAuthorizationDenial(string? errorCode) => errorCode is not null && AuthorizationDenials.Contains(errorCode);
}
```

程式說明：

片段	做了什麼	為什麼這樣寫
FindSystemTimeZoneById("Asia/Taipei")	取得臺灣時區	.NET 在 Windows 與 Linux 都認得 IANA 名稱；部署到 Linux（Section 15）也不必改
DateOf	UTC 時刻 → 臺灣日期	每日趨勢分組用
StartUtc	臺灣日期 0 點 → UTC	查詢邊界用；區間終點是「迄日隔天」的 StartUtc
AuthorizationDenials	直接引用 Section 11 的 AccessErrors 常數	新增授權錯誤代碼時只改一處，定義不會兩邊不一致

預期結果：TaipeiCalendar.StartUtc(new DateOnly(2026, 9, 26)) 是 2026-09-25 16:00:00 +00:00。

步驟二 計算儀表板（15 分鐘）

目的：讀取區間內的呼叫事實，算出 KPI、每日趨勢、排名與錯誤分布。

```
// 範例/13_數位儀表板與稽核查詢/McpPlatform/src/McpPlatform.Core/Dashboard/DashboardService.cs (GetAsync 前半)
public async Task<DashboardDto> GetAsync(DateOnly? from, DateOnly? to, CancellationToken ct)
{
    var (start, end) = NormalizeRange(from, to);
    var facts = await reader.ListCallsAsync(TaipeiCalendar.StartUtc(start), TaipeiCalendar.StartUtc(end.AddDays(1)), ct); // ① [起日 0:0
0, 迄日隔天 0:00)
    var executions = facts.Where(f => !CallCategories.IsAuthorizationDenial(f.ErrorCode)).ToList(); // ② 工具執行
    var succeeded = executions.Count(f => f.Outcome == CallOutcome.Success);
    var durations = executions.Select(f => f.DurationMs).Order().ToList();

    var daily = Enumerable.Range(0, end.DayNumber - start.DayNumber + 1)
        .Select(start.AddDays)
        .Select(day =>
        {
            var ofDay = facts.Where(f => TaipeiCalendar.DateOf(f.CalledAtUtc) == day).ToList(); // ③ 依臺灣日期分組
            var denied = ofDay.Count(f => CallCategories.IsAuthorizationDenial(f.ErrorCode));
            var ok = ofDay.Count(f => f.Outcome == CallOutcome.Success);
            return new DailyPoint(day, ok, ofDay.Count - ok - denied, denied);
        }).ToList();
}
```

```
// 範例/13_數位儀表板與稽核查詢/McpPlatform/src/McpPlatform.Core/Dashboard/DashboardService.cs (GetAsync 後半與 Percentile)
return new DashboardDto(start, end, await reader.GetCountsAsync(ct),
    executions.Count, succeeded,
    executions.Count == 0 ? null : succeeded / (double)executions.Count, // ④ 零筆：無資
料，不是 0%
    durations.Count == 0 ? null : (int)Math.Round(durations.Average()),
    Percentile(durations, 0.95),
    facts.Count - executions.Count, daily, top, errors);
}

/// <summary>最近排名法 (nearest-rank)：由小到大批序後取第 [p × n] 筆，與 SQL Server 的 PERCENTILE_DISC 相同。</summary>
public static int? Percentile(IReadOnlyList<int> sorted, double p) =>
    sorted.Count == 0 ? null : sorted[(int)Math.Ceiling(p * sorted.Count) - 1];
```

程式說明：

標號	做了什麼	為什麼這樣寫
①	查詢區間是半開的 [起日 0:00, 迄日隔天 0:00)	「含迄日整天」又不會跟下一個區間重疊
②	排除授權拒絕，剩下的才是工具執行	成功率、延遲都只看工具執行
	<code>Enumerable.Range(...)</code>	區間內 每一天 都產生一筆，即使當天 0 筆
③	<code>failed = 當天總數 - 成功 - 授權拒絕</code>	三者相加等於當天所有紀錄
④	分母 0 時回 <code>null</code>	畫面把 <code>null</code> 顯示成「無資料」
<code>Percentile</code>	排序後取第 [0.95n] 筆	本例 n = 57，[54.15] = 55，取第 55 筆 = 4081 ms，與 <code>PERCENTILE_DISC</code> 相同
<code>top</code> 、 <code>errors</code> (完整檔案中)	<code>GroupBy</code> 版本名稱取前 5 名； <code>GroupBy</code> 錯誤代碼並標記類別	排名依執行次數，同次數再依名稱排序，結果穩定

資料來自 `EfdashboardReader.ListCallsAsync`：`ApiCallLogs` 以半開區間篩選，left join `ApiVersions` 取「API 名稱 版本」，版本不存在時顯示「(未知版本)」；只選 5 個欄位並 `AsNoTracking()`。

預期結果：示範步驟 1、2 的數字，與 SQL 一致。

步驟三 儀表板頁面與圖表 (15 分鐘)

目的：KPI 卡片、Chart.js 趨勢圖與表格替代。

資料以 JSON 放在頁面裡，圖表腳本讀它來畫圖，不另外呼叫 API：

```
<script type="application/json" id="daily-data">@Json.Serialize(Model.Daily.Select(d => new { date = d.Date.ToString("MM-dd"), d.Succeeded, d.Failed, d.Denied })))</script>
```

```
// 範例/13_數位儀表板與稽核查詢/McpPlatform/src/McpPlatform.Web/wwwroot/js/dashboard.js (節錄)
const points = JSON.parse(source.textContent);
new Chart(canvas, {
  type: 'bar',
  data: {
    labels: points.map(p => p.date),
    datasets: [
      { label: '成功', data: points.map(p => p.succeeded), backgroundColor: '#1f4d3a' },
      { label: '失敗', data: points.map(p => p.failed), backgroundColor: '#b3541e' },
      { label: '授權拒絕', data: points.map(p => p.denied), backgroundColor: '#a88450' }
    ]
  }
},
```

程式說明：

片段	做了什麼	為什麼這樣寫
<code><script type="application/json"></code>	瀏覽器不會執行這段，只是資料	伺服器算好的數字直接帶給腳本，不必再開一支 API、再做一次授權
<code>@Json.Serialize(...)</code>	ASP.NET Core 的 JSON 序列化，屬性名稱轉成小寫開頭（ <code>succeeded</code> ）	會把 <code><</code> 等字元跳脫，資料裡的字串不會提早結束 <code><script></code>
<code>JSON.parse(source.textContent)</code>	讀出資料	—
三個 <code>datasets</code> ， <code>stacked: true</code> （完整檔案）	成功、失敗、授權拒絕疊成一根長條	一眼看到每天總量與組成
顏色	墨綠、焦橙、古銅金，取自 <code>site.css</code> 的配色	與網站古典風格一致
<code><canvas role="img" aria-label=...> + <details></code> 表格	螢幕閱讀器讀得到說明，也能展開表格看數字	圖表不是唯一的資訊來源

Chart.js 放在 `wwwroot/lib/chart.js/dist/chart.umd.min.js`（本機檔案，不用 CDN），只在儀表板頁的 `@section Scripts` 載入。

預期結果：圖 13-2。

步驟四 呼叫紀錄與稽核軌跡查詢（15 分鐘）

目的：依條件查詢，並能用 CorrelationId 追查單筆。

```
// 範例/13_數位儀表板與稽核查詢/McpPlatform/src/McpPlatform.Infrastructure/Dashboard/EfDashboardReader.cs (SearchCallsAsync 前半)
var logs = db.ApiCallLogs.AsNoTracking();
if (q.CorrelationId is not null)
    logs = logs.Where(l => l.CorrelationId == q.CorrelationId); // 追查單筆：只看 CorrelationId
else
{
    logs = logs.Where(l => l.CalledAtUtc >= fromUtc && l.CalledAtUtc < toUtc);
    if (q.Tool is not null) logs = logs.Where(l => l.ToolName == q.Tool);
    if (q.Outcome is not null) logs = logs.Where(l => l.Outcome == q.Outcome);
    if (q.ErrorCode is not null) logs = logs.Where(l => l.ErrorCode == q.ErrorCode);
}
```

稽核軌跡的 Controller：

```
// 範例/13_數位儀表板與稽核查詢/McpPlatform/src/McpPlatform.Web/Areas/Admin/Controllers/LogsController.cs (Audit)
/// <summary>參數不能叫 action：那是 MVC 保留的路由值，會繫結成目前的 Action 名稱「Audit」。</summary>
public async Task<IActionResult> Audit(DateOnly? from, DateOnly? to, string? auditAction, string? keyword, CancellationToken ct)
{
    var (query, rows) = await logs.SearchAuditAsync(new AuditQuery(from, to, auditAction, keyword), ct);
    ViewData["Query"] = query;
    ViewData["Actions"] = await logs.ListAuditActionsAsync(ct);
    return View(rows);
}
```

程式說明：

片段	做了什麼	為什麼這樣寫
有 CorrelationId 時忽略其他條件	追查單筆時不受日期限制	會員可能隔幾天才回報，不必先猜日期
其餘條件逐一 Where	沒給的條件就不加	EF Core 組出剛好需要的 SQL，全部參數化
DateOnly? from	ASP.NET Core 直接把 ?from=2026-09-20 繫結成 DateOnly	沒給日期時是 null，服務層套用預設區間
auditAction	下拉選單的欄位名稱也叫 auditAction	取名 action 會被路由值蓋掉（見〈常見錯誤〉）
ErrorCatalog.Describe(code)	追查卡片把代碼翻成中文說明	管理者不必背代碼；代碼本身仍保留，方便搜尋

預期結果：圖 13-3、圖 13-4。

檢查點

- ☐ 儀表板的工具執行、成功數、平均、P95、授權拒絕與 SQL 查詢完全相同
- ☐ 做一次呼叫後，工具執行 +1；做一次被額度擋下的呼叫，授權拒絕 +1、成功率不變
- ☐ 沒有資料的日期顯示「無資料」；起訖顛倒自動對調
- ☐ 用 CorrelationId 查得到單筆，並看到錯誤代碼說明
- ☐ 稽核軌跡可以依動作與關鍵字查詢

常見錯誤

以下都是撰寫本單元時實際遇到的情況。

症狀	原因	處理
稽核軌跡選了動作 <code>SuspendMember</code> ，結果「無資料」，下拉選單又跳回「全部」	Action 參數取名 <code>action</code> ，它是 MVC 保留的路由值，繫結到的是目前的 Action 名稱 <code>Audit</code> ，於是查詢條件變成 <code>Action = 'Audit'</code>	參數與表單欄位改名為 <code>auditAction</code>
P95 卡片的「4081 ms」被擠成兩行	五張卡片並排時，大字的數字加單位超過卡片寬度	單位移到標題（「P95 延遲 (ms)」），卡片只放數字
測試腳本印出 <code>tools/List → 200 工具：</code> （空白），但呼叫 <code>courses_search_v2</code> 其實成功	腳本用 <code>"name": "([a-z_]+)"</code> 擷取工具名稱，名稱含數字就對不上	改成 <code>[a-z0-9_]+</code> （Section 10~13 的 <code>oauth-flow.ps1</code> 都已修正）

單元驗收（15 分鐘）

驗收標準：完成一次呼叫後統計有對應變化；成功率與資料表一致；可透過 CorrelationId 追查單筆失敗。

驗收方式：

1. **完成一次呼叫後統計有對應變化：**記下儀表板的工具執行與成功數 → 用 `tools/mcp-call.ps1` 成功呼叫一次 → 重新整理，工具執行與成功各 +1（實測 57/31 → 做一成一敗後 59/32）。
2. **成功率與資料表一致：**用示範步驟 2 的 SQL 查同一個臺灣日期區間，工具執行、成功數、成功率、平均與儀表板完全相同；`PERCENTILE_DISC(0.95)` 等於儀表板的 P95。
3. **可透過 CorrelationId 追查單筆失敗：**把失敗回應中的 CorrelationId 貼到「呼叫紀錄」，看到會員、Client、API、Grant、`InvalidInput`、`INVALID_ARGUMENTS` 與中文說明。
4. **反向驗證：**沒有資料的日期顯示「無資料」而不是 0%；授權拒絕（例如 `QUOTA_EXCEEDED`）只增加「授權拒絕」，不影響成功率。

回顧：先把指標定義寫死（授權拒絕不是執行失敗、P95 用哪種算法），再用 SQL 逐一驗證；資料存 UTC、邊界用臺灣日期、區間半開；零筆顯示無資料；每一筆失敗都能用 CorrelationId 追到。下一單元（Section 14）把權限、安全與整合測試系統化，並驗證三種視窗寬度的 RWD。

延伸閱讀

- [Chart.js 文件：Bar Chart 與 Stacked](#)。
- [PERCENTILE_DISC \(Transact-SQL\)](#)、[PERCENTILE_CONT](#)。
- [.NET 的時區轉換 \(TimeZoneInfo\)](#)。

第 14 章

權限安全與整合測試

角色越權、跨會員資料存取、CSRF、SSRF、憑證遮罩、重複審核、版本變更與額度併發；以測試帳號執行端到端案例，檢查 RWD、表單錯誤、鍵盤導覽與中文可讀性



課程時數 2 小時 (120 分鐘)

當堂成果

測試報告、問題修正與 RWD 驗證

範例專案

範例/14_權限安全與整合測試

學習目標

上完這個單元，你能夠：

- 建立 xUnit 測試專案，在中央套件管理下正確加入測試套件，用 `dotnet test` 一次跑完單元測試與整合測試。
- 用 `WebApplicationFactory<Program>` 以真的 SQL Server 啟動整個網站，對獨立的測試資料庫執行端到端的安全案例。
- 把大綱列出的八種風險（角色越權、跨會員資料、CSRF、SSRF、憑證遮罩、重複審核、版本變更、額度併發）各寫成至少一個「應該被拒絕」的測試。
- 依「先寫會失敗的測試 → 修正 → 測試通過」的順序，修掉本單元實際找到的跨會員資料外洩缺陷。
- 用 Playwright 量測三種寬度的水平溢出與最小字級，檢查鍵盤導覽與表單錯誤，整理成測試報告。

先備知識

- 完成 Section 13：網站有前臺、會員中心、後臺、`/mcp`、儀表板與紀錄查詢；本單元新增會員自己的「呼叫紀錄」頁（`/Member/Calls`）。
- 開發資料庫 `MCPServerManag` 與網站專案的 User Secrets（`ConnectionStrings:DefaultConnection`、`Seed:DemoPassword`、`Seed:AdminPassword`）。
- 知道 HTTP 狀態碼 302、400、401、403、404 的差別；知道 `[Fact]`、`Assert.Equal` 的意思即可。

觀念說明（20 分鐘）

測什麼：成功之外，更要測「應該失敗」

前面 13 個單元的教師示範，大多證明「功能做得到」。安全問題恰好相反：出事的地方是**不該做到卻做到了**。所以本單元每個測試的形式都是「用某種身分、做某件不該做的事，系統必須拒絕，而且資料沒有改變」：

風險	攻擊者的想法	本平台的防線（單元）	測試怎麼證明
角色越權	一般會員直接打後臺網址、呼叫後臺 API；註冊時偷塞角色	Area 授權慣例（04）、 <code>Authorize(Policy)</code> 、註冊不綁定角色	302 到拒絕存取／403；資料庫查不到管理者角色
跨會員資料	改網址上的 Id 看別人的草稿、紀錄	擁有者檢查（05）、查詢固定帶本人（14）	404／「無資料」
CSRF	誘使管理者的瀏覽器送出表單	全站 <code>AutoValidateAntiForgeryToken</code> （04）	400，且對象沒有被停權
SSRF	讓平台去打內網或雲端中繼資料	允許清單＋連線時檢查 IP（08）	私有位址全部判定為擋下
憑證遮罩	從任何頁面、API 看到上游金鑰	Data Protection 加密，頁面不回填（05）	資料庫只有密文、五個頁面都沒有明文
重複審核	同一件申請核准兩次，多拿一筆授權	狀態檢查＋rowversion（06、07）	第二次被拒；Grant 只有 1 筆
版本變更	用 v1 的授權呼叫 v2	Grant 綁版本（11、12）	當作沒有這個工具
額度併發	同時送出大量呼叫突破額度	原子 <code>UPDATE</code> （11）	20 個並行搶 10 次，剛好 10 個成功

單元測試與整合測試

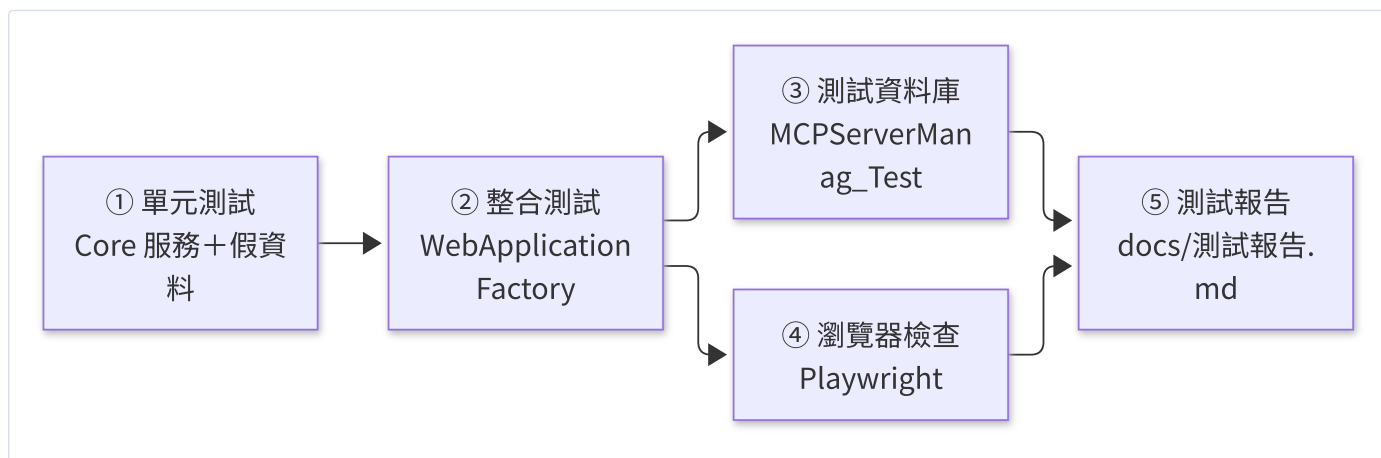


圖 14-1 本單元的三層檢查與報告

編號	測什麼	速度與範圍
①	<code>ToolAccessService</code> 、 <code>ConnectorService</code> 、 <code>MemberAdminService</code> 、 <code>SsrfGuard</code> 、儀表板計算：規則本身對不對	毫秒級；不需要資料庫，用自己寫的假 Repository
②	經過路由、驗證、授權、防偽、Controller、Service、EF Core 的完整請求	整個網站在測試程序裡啟動（TestServer），不佔用埠號
③	獨立資料庫，每次測試前刪除重建、套用同一套 Migration、匯入示範資料	不會弄亂開發資料庫；每次結果一樣
④	版面、字級、鍵盤、表單訊息：只有真的瀏覽器才看得出來	三種寬度 × 六個頁面
⑤	把通過與拒絕的證據、發現的缺陷與修正整理成文件	大綱要求的驗收產出

先寫會失敗的測試，再修

本單元用探測腳本發現：林小華打開 `/Member/Access/Apply/6`（王小明**尚未公開**的草稿），頁面回 200，顯示「匯率換算 v1（提供者：王小明）」；同一個版本在公開目錄是 404。這不會讓人申請成功（送出時服務層會拒絕），但洩漏了別人還沒公開的資料。

修正的順序是：

1. 寫一個測試描述正確行為（申請頁應該 404）。
2. 執行，確認它**失敗**（實測：`Expected: NotFound`、`Actual: OK`）。這證明測試真的測得到這個缺陷。
3. 修正程式。
4. 再執行，確認它通過，而且其他測試仍然通過。

沒有經過第 2 步的測試，可能因為寫錯而永遠通過，給人錯誤的安全感。

限制與注意事項

- 整合測試需要本機的 SQL Server 與網站專案的 User Secrets；在沒有資料庫的電腦上只能跑單元測試。
- 測試資料庫每次都會被刪除重建。程式裡有保險：資料庫名稱不是 `MCPServerManag_Test` 就拒絕刪除。
- 整合測試以 HTTP 表單操作網站，不執行 JavaScript；OAuth 同意頁的瀏覽器流程在 Section 10 以 Playwright 驗證過。
- 字級量測只能找出「小於 12px 的文字」，無法判斷用語是否通順；中文可讀性還是要人看。

名詞對照

名詞	白話說明	在本課程中是
單元測試	只測一個類別的規則，其他相依換成假的	<code>tests/McpPlatform.Tests/Unit</code>
整合測試	把多層串起來一起測	<code>tests/McpPlatform.Tests/Integration</code>
WebApplicationFactory	在測試程序裡啟動整個 ASP.NET Core 網站	<code>PlatformFactory</code>
測試固件（Fixture）	多個測試共用、只建立一次的準備工作	刪除重建測試資料庫
回歸測試	修正後保留的測試，防止同一個缺陷再出現	<code>別人的草稿_不能開申請頁看到名稱</code>

教師示範（25 分鐘）

示範情境：先用探測腳本對執行中的網站打一輪「不該成功」的請求，找出缺陷；再執行 `dotnet test` 看修正前後的差別；最後量測 RWD 與鍵盤。

示範步驟

- 探測（網站在 `http://localhost:5199` 執行，以王小明、林小華、管理者三個身分）。本單元實測節錄：

```
PASS 會員 GET /Admin 302 → /Account/AccessDenied
PASS 會員 POST /Admin/Members/Suspend/member-b 302 → /Account/AccessDenied
PASS a 改 b 的草稿 POST /Member/Apis/Edit/14 404
PASS 管理者 POST /Admin/Members/Suspend/member-b 不帶權杖 400
PASS 註冊時附加 Role=Admin、IsAdmin=true → 新帳號 GET /Admin 302 → /Account/AccessDenied
PASS 管理者頁面 /api/admin/apis 不含金鑰明文 clean
FAIL a 申請草稿版本 6 (GET Apply/6) 200
FAIL a 申請已下架版本 2 (GET Apply/2) 200
```

兩個 FAIL 追下去：送出申請會被服務層拒絕（沒有產生任何申請紀錄），但**林小華**打開 `Apply/6` 看得到「匯率換算 v1（提供者：王小明） 這個版本目前不是「已發布」，不能申請。」——別人的草稿名稱外洩了。

- 在範例目錄執行測試（修正前）：

```
dotnet test tests/McpPlatform.Tests
```

```
失敗 McpPlatform.Tests.Integration.SecurityTests.別人的草稿_不能開申請頁看到名稱 [96 ms]
錯誤訊息:
    Assert.Equal() Failure: Values differ
Expected: NotFound
Actual:    OK
失敗! - 失敗:      2, 通過:      43, 略過:      0, 總計:      45
```

另一個失敗是測試本身的假設錯了（見〈常見錯誤〉），修正測試後只剩這一個真的缺陷。

- 修正 `AccessService.GetApplyContextAsync`（學員實作步驟四），再執行：

```
已通過! - 失敗:      0, 通過:      45, 略過:      0, 總計:      45, 持續時間: 4 s - McpPlatform.Tests.dll (net10.0)
```

- RWD 與鍵盤。以 Playwright 在 375、768、1280 三種寬度開啟六個頁面，量測整頁水平溢出與最小字級。第一次量測發現手機上 `GET` 標籤 11.25px、表格內的錯誤代碼 11.48px，修正 CSS 後全部 $\geq 12\text{px}$ （完整表格在 `docs/測試報告.md`）。



圖 14-2 375px：展開的選單、登入表單錯誤、儀表板、會員呼叫紀錄

畫面重點：

- 1. ①**展開的選單**：右上角選單按鈕有焦點外框，選單項目「API 目錄、會員中心、登入、註冊」上下排列，按鈕的 `aria-expanded` 變成 `true`。
- 2. ②**空白送出登入表單**：紅框欄位下方「請輸入Email。」「請輸入密碼。」，訊息緊鄰欄位、是中文。
- 3. ③**儀表板**：KPI 卡片兩張一列，日期欄位與「套用」按鈕換行排列，沒有被切掉。
- 4. ④**會員呼叫紀錄**：表格欄位多，只在表格外框內左右捲動；整頁寬度仍是 375px（量測的水平溢出為 0）。

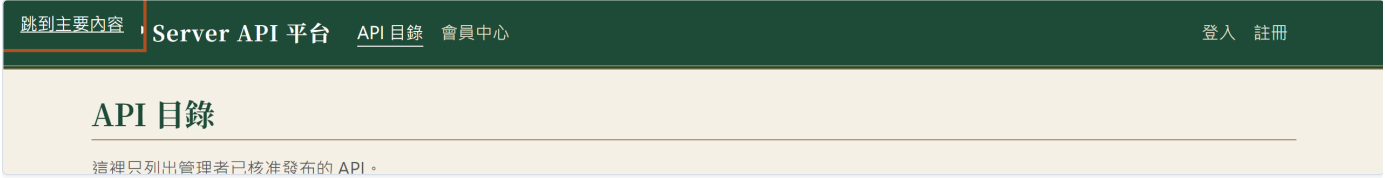


圖 14-3 第一次按 Tab：「跳到主要內容」出現並帶焦點外框

按 Enter 後網址變成 `/Catalog#main`，鍵盤使用者不必每頁都先經過整排導覽連結。

學員實作（60 分鐘）

範例專案：[範例/14_權限安全與整合測試/McpPlatform/](#)（從 Section 13 複製）。本單元新增 `tests/McpPlatform.Tests`（`Unit/`、`Integration/` 共 45 項測試）、`docs/測試報告.md`、會員呼叫紀錄頁（`Member/CallsController`、查詢固定帶本人）；修正 `AccessService.GetApplyContextAsync` 與 `site.css`；`Program.cs` 結尾加 `public partial class Program;`；`Directory.Packages.props` 加入四個測試套件。本單元沒有新的 Migration。

步驟一 建立測試專案（10 分鐘）

目的：在中央套件管理下加入 xUnit 專案。

```
dotnet new xunit -o tests/McpPlatform.Tests -n McpPlatform.Tests --framework net10.0
dotnet sln McpPlatform.slnx add tests/McpPlatform.Tests
dotnet add tests/McpPlatform.Tests reference src/McpPlatform.Web
```

範本產生的 csproj 在 `PackageReference` 上寫了版本號，建置時實測得到 `NU1008`（見〈常見錯誤〉）。版本改寫到 `Directory.Packages.props`：
`Microsoft.AspNetCore.Mvc.Testing` 10.0.12、`Microsoft.NET.Test.Sdk` 17.14.1、`xunit` 2.9.3、`xunit.runner.visualstudio` 3.1.4；csproj 只留套件名稱。

網站的 `Program.cs` 使用頂層陳述式，最後加一行讓測試專案看得到它：

```
// 範例/14_權限安全與整合測試/McpPlatform/src/McpPlatform.Web/Program.cs（結尾）
/// <summary>讓整合測試（tests/McpPlatform.Tests）能以 WebApplicationFactory<Program> 啟動整個網站。</summary>
public partial class Program;
```

預期結果：`dotnet build` 整個方案 0 個警告、0 個錯誤。

步驟二 單元測試：規則本身（10 分鐘）

目的：不用資料庫，用假的 Repository 驗證 Section 11 的授權規則。

```
// 範例/14_權限安全與整合測試/McpPlatform/tests/McpPlatform.Tests/Unit/ToolAccessServiceTests.cs（節錄）
[Fact]
public async Task 舊版本的授權不涵蓋新版本()
{
    var (service, _) = Create(Grant(versionId: 2)); // 只有 v1（版本 2）的 Grant
    var decision = await service.AuthorizeCallAsync("member-b", McpScope, 19, default); // 呼叫 v2（版本 19）
    Assert.True(decision.Hidden);
}

[Theory]
[InlineData(GrantStatus.Revoked, 30, AccessErrors.GrantRevoked)]
[InlineData(GrantStatus.Active, -1, AccessErrors.GrantExpired)] // 狀態仍是 Active，但時間已過
[InlineData(GrantStatus.Expired, 30, AccessErrors.GrantExpired)]
public async Task 授權不能用_回傳對應代碼(GrantStatus status, int days, string expected)
{
    var (service, repo) = Create(Grant(1, status, days));
    var decision = await service.AuthorizeCallAsync("member-a", McpScope, 1, default);
    Assert.False(decision.Allowed);
    Assert.Equal(expected, decision.ErrorCode);
    Assert.Equal(0, repo.Reservations);
}
```

程式說明：

片段	做了什麼	為什麼這樣寫
<code>Create(...)</code>	建立 <code>ToolAccessService</code> ，傳入假的 Repository 與固定時間的 <code>FixedClock</code>	結果不受執行日期影響；不必準備資料庫
方法名稱用中文	測試失敗時，名稱本身就說明「哪一條規則壞了」	報告與輸出一看就懂
<code>[Theory]</code> + <code>[InlineData]</code>	同一個測試跑三組資料	撤銷、時間到期、狀態到期三種情況共用一段程式
<code>Assert.Equal(0, repo.Reservations)</code>	被拒絕時沒有扣額度	Section 11 的計次規則：被拒絕不計次

同一個資料夾的 `SecurityRuleTests` 測 SSRF 位址（8 組）、Redirect URI（6 組）、管理者不能對自己操作、P95 與臺灣日期邊界。

預期結果：單元測試 29 項通過，不需要 SQL Server。

步驟三 整合測試的基礎：測試資料庫與網站（15 分鐘）

目的：以真的 SQL Server 啟動整個網站，但絕不碰到開發資料庫。

```
// 範例/14_權限安全與整合測試/McpPlatform/tests/McpPlatform.Tests/Integration/PlatformFactory.cs (建構與初始化)
public PlatformFactory()
{
    var main = secrets.GetConnectionString("DefaultConnection")
        ?? throw new InvalidOperationException("網站專案的 User Secrets 缺少 ConnectionStrings:DefaultConnection。");
    ConnectionString = new SqlConnectionStringBuilder(main) { InitialCatalog = TestDatabase }.ConnectionString; // ① 只換資料庫名稱
    Environment.SetEnvironmentVariable("ConnectionStrings__DefaultConnection", ConnectionString); // ② 環境變數優先於 User S
    secrets
    }

    public async Task InitializeAsync()
    {
        if (new SqlConnectionStringBuilder(ConnectionString).InitialCatalog != TestDatabase)
            throw new InvalidOperationException("只允許刪除重建測試資料庫。"); // ③ 保險：絕不動到開發資料庫
        await using var db = NewDbContext();
        await db.Database.EnsureDeletedAsync();
        await db.Database.MigrateAsync(); // ④ 與正式環境同一套 Mig
    }
}
```

程式說明：

標號	做了什麼	為什麼這樣寫
<code>secrets</code>	<code>new ConfigurationBuilder().AddUserSecrets<Program>()</code> 讀網站專案的 User Secrets	測試程式碼裡沒有任何密碼
①	同一台 SQL Server、同一組帳密，資料庫名稱換成 <code>MCPServerManag_Test</code>	不必另外設定一組連線
②	用環境變數覆蓋連線字串	網站啟動時的設定來源順序是 appsettings → User Secrets → 環境變數，後面的蓋前面；若用較早的來源，User Secrets 會把它蓋回開發資料庫
③	刪除前再確認一次名稱	設定錯一個字就刪掉開發資料的風險，用一行程式擋掉
④	刪除重建並套用 Migration	每次從同一個起點開始；網站啟動時 <code>DemoDataSeeder</code> 匯入與開發環境相同的示範資料
<code>ConfigureWebHost</code> → <code>UseEnvironment("Development")</code> （完整檔案）	以開發環境啟動	示範資料只在開發環境匯入
<code>ICollectionFixture<PlatformFactory></code>	所有整合測試共用同一個網站與資料庫	只重建一次；同一個 Collection 的測試依序執行，不會互相干擾

`PlatformFactory.LoginAsync(email)` 先 GET 登入頁取防偽權杖，再 POST 帳密，並斷言回應是 302；`TestBrowser` 不自動跟隨轉址，測試才看得到 302 轉去哪裡。

預期結果：第一次執行 `dotnet test` 時，SQL Server 上出現 `MCPServerManag_Test`，裡面有 4 位示範會員與示範 API。

步驟四 安全案例與缺陷修正（15 分鐘）

目的：寫出跨會員資料、額度併發等案例，並修正本單元找到的缺陷。

先寫測試（修正前實測失敗）：

```
// 範例/14_權限安全與整合測試/McpPlatform/tests/McpPlatform.Tests/Integration/SecurityTests.cs (跨會員草稿)
[Fact]
public async Task 別人的草稿_不能開申請頁看到名稱()
{
    var draftId = await VersionIdAsync("匯率換算", "v1"); // 王小明的草稿，公開目錄看不到
    var other = await platform.LoginAsync("b@demo.test");
    Assert.Equal(HttpStatusCode.NotFound, (await other.GetAsync($"Catalog/Details/{draftId}")).StatusCode);
    Assert.Equal(HttpStatusCode.NotFound, (await other.GetAsync($"Member/Access/Apply/{draftId}")).StatusCode);
}
```

再修正：

```
// 範例/14_權限安全與整合測試/McpPlatform/src/McpPlatform.Core/Access/AccessService.cs (GetApplyContextAsync)
/// <summary>
/// 申請頁的資料。只有「已發布」的版本才回傳；草稿、審核中、被退回、已下架一律當作不存在（404）。
/// Section 14 修正：原本會回傳任何版本，別的會員可以從申請頁看到尚未公開的草稿名稱與提供者。
/// </summary>
public async Task<ApplyContextDto?> GetApplyContextAsync(string userId, int versionId, CancellationToken ct)
{
    var context = await repository.GetApplyContextAsync(userId, versionId, clock.UtcNow(), ct);
    return context is { IsPublished: true } ? context : null;
}
```

額度併發直接對 SQL Server 測：

```
// 範例/14_權限安全與整合測試/McpPlatform/tests/McpPlatform.Tests/Integration/QuotaConcurrencyTests.cs (節錄)
var results = await Task.WhenAll(Enumerable.Range(0, 20).Select(async _ =>
{
    await using var scope = platform.Services.CreateAsyncScope(); // 每個請求自己的 DbContext，就像真實的並行請
求
    var repository = scope.ServiceProvider.GetRequiredService<IToolAccessRepository>();
    return await repository.TryReserveAsync(grantId, period, quota: 10, CancellationToken.None);
}));

Assert.Equal(10, results.Count(ok => ok));
```

程式說明：

片段	做了什麼	為什麼這樣寫
<code>VersionIdAsync("匯率換算", "v1")</code>	用名稱查出 Id	全新資料庫的 Id 與開發資料庫不同，不能寫死 6
同時斷言公開目錄也是 404	證明「這是不該公開的東西」	申請頁的行為要與公開目錄一致
修正放在 <code>AccessService</code> ，而不是 Controller	GET 與 POST 都經過這個方法	一處修正，兩個入口都生效； <code>ApplyAsync</code> 原本的「只能申請已發布版本」檢查保留，多一層保險
<code>platform.Services.CreateAsyncScope()</code>	每個並行任務有自己的 DbContext	共用同一個 DbContext 不是真的並行，而且 EF Core 不允許同時使用
<code>period = 2099-01-01</code> （完整檔案）	專用的計量日	不影響其他測試與示範資料

其他整合案例：匿名／會員開後臺（6 項）、後臺 API 401／403、註冊偷塞角色、CSRF、別人的編輯頁與憑證頁、別人的 CorrelationId、上游金鑰不外洩、重複核准只產生一筆 Grant。

預期結果：修正前 `別人的草稿_不能開申請頁看到名稱` 失敗；修正後 45 項全部通過。

步驟五 RWD、鍵盤、表單與可讀性（10 分鐘）

目的：用瀏覽器量測，並修正小於 12px 的文字。

量測的核心是在頁面裡執行：

```
// 取自教師的 Playwright 量測腳本：整頁是否出現水平捲軸
const overflow = document.documentElement.scrollWidth - window.innerWidth;
```

以及逐一計算有文字的元素的 `getComputedStyle(el).fontSize` 取最小值。沒有 Playwright 時，用瀏覽器開發人員工具的裝置模式切換 375／768／1280px，在 Console 執行上面那一行也能得到同樣的數字。手機版 `html` 字級是 15px（Section 02），`.75rem` 就只有 11.25px。修正：

```
/* 範例/14_權限安全與整合測試/McpPlatform/src/McpPlatform.Web/wwwroot/css/site.css (節錄) */
.method { display: inline-block; font-size: max(.75rem, 12px); font-weight: 700; padding: 0 .4rem; border-radius: 3px; background: var(--brown); color: #fff; }

/* Section 14：手機上（html 15px）小字的程式碼、錯誤代碼不低於 12px */
code { font-size: max(.875em, 12px); }
```

片段	做了什麼	為什麼這樣寫
<code>max(.75rem, 12px)</code>	取兩者較大的	桌機（16px 基準）維持原比例 12px，手機不會低於 12px
<code>code {875em ... }</code>	程式碼字級相對於外層	放在 <code>.small</code> 表格裡時原本會乘兩次 0.875

鍵盤與表單的檢查項目：第一次 Tab 的焦點、Enter 之後的網址、手機選單的 `aria-expanded`、空白送出登入表單的訊息。結果寫進 `docs/測試報告.md`。

預期結果：六個頁面 × 三種寬度，水平溢出都是 0、最小字級都 ≥ 12px；圖 14-2、圖 14-3。

檢查點

- ☐ `dotnet test` 45 項全部通過，SQL Server 上有 `MCPServerManag_Test`，開發資料庫沒有被改動
- ☐ 把 `GetApplyContextAsync` 暫時改回原本的寫法，`別人的草稿_不能開申請頁看到名稱` 會失敗；改回修正版又通過
- ☐ 375、768、1280px 三種寬度沒有整頁水平捲軸，文字不小於 12px
- ☐ 第一次 Tab 是「跳到主要內容」；空白登入顯示中文錯誤訊息
- ☐ `docs/測試報告.md` 記錄了通過與拒絕的證據、缺陷與修正

常見錯誤

以下都是撰寫本單元時實際遇到的原文。

症狀（錯誤訊息原文）	原因	處理
<code>error NU1008: 下列 PackageReference 項目無法定義 Version: coverlet.collector, Microsoft.NET.Test.Sdk, xunit, xunit.runner.visualstudio 的值。使用中央套件管理的專案必須在 PackageVersion 項目上定義 Version 值。</code>	<code>dotnet new xunit</code> 的範本不知道方案用了中央套件管理	版本移到 <code>Directory.Packages.props</code> ， <code>csproj</code> 只寫 <code>Include</code> ；用不到的 <code>coverlet.collector</code> 直接移除
測試 上游金鑰_只存密文_任何頁面都看不到明文 失敗：Expected: Found、Actual: OK	測試的假設錯了：示範資料的「匯率換算 v1」認證方式是「不需認證」，設定憑證被正確拒絕（頁面顯示「此版本的認證方式不是「API Key 標頭」，不需要設定憑證。」）	測試先把該草稿改成 <code>ApiKeyHeader</code> 再設定憑證；不是程式缺陷
手機上 <code>GET</code> 標籤只有 11.25px	手機的 <code>html</code> 字級是 15px， <code>.75rem</code> = 11.25px	<code>font-size: max(.75rem, 12px)</code>

單元驗收（15 分鐘）

驗收標準：提交成功與拒絕案例證據；修復能略過審核、授權或後臺角色限制的缺陷。

驗收方式：

1. **提交成功與拒絕案例證據：** `dotnet test` 輸出「已通過! - 失敗: 0，通過: 45」； `docs/測試報告.md` 依八種風險列出每個拒絕案例與結果，並附 RWD 量測表。
2. **修復缺陷：**報告第二節的缺陷 1（草稿名稱外洩）有「修正前測試失敗 → 修正 → 測試通過」的完整證據；把修正暫時還原，測試會再次失敗。
3. **反向驗證**（抽查三項）：一般會員 POST 停權他人 → 拒絕存取；管理者不帶防偽權杖 → 400；同一件申請核准兩次 → 只有 1 筆 Grant。

回顧：安全測試測的是「不該成功的事確實失敗、資料沒有改變」；單元測試驗規則、整合測試驗整條路、瀏覽器量測驗版面；缺陷一律先寫會失敗的測試再修。下一單元（Section 15）部署到教室主機，用真實 Connector 展示整條流程。

延伸閱讀

- [ASP.NET Core 整合測試 \(WebApplicationFactory\)](#)。
- [OWASP Top 10](#)：存取控制失效、注入、SSRF 等常見風險。
- [WCAG 2.2 快速參考](#)：鍵盤操作、焦點可見、重排（Reflow）。

第 15 章

部署安裝與成果驗收

IIS、HTTPS、環境設定、資料庫 Migration、Data Protection 金鑰持久化、日誌、備份與復原；發布到教室主機、專用 SQL 帳號、正式測試網址與 OAuth、Connector 展示與成果驗收



課程時數 2 小時（120 分鐘） 當堂成果 可操作網站、Connector 展示及操作文件 範例專案 範例/15_部署安裝與成果驗收

學習目標

上完這個單元，你能夠：

- 說明開發環境與正式環境的差別，把連線字串、憑證密碼、OAuth 網址全部改用環境變數，並讓正式環境只接受 https。
- 用昇等 Migration 腳本建立全新的正式資料庫，建立只有讀寫權限的網站專用 SQL 帳號，並證明它刪不了資料表。
- 設定 Data Protection 金鑰持久化，實測「有持久化時重新啟動仍登入、沒有時被登出」。
- 以環境變數建立第一位管理者，在全新的正式網站上跑完「新會員註冊 → API 上架核准 → 使用申請核准 → Connector 呼叫 → 儀表板查詢 → 撤銷後拒絕」。
- 完成備份與還原演練，並交付安裝說明、角色操作手冊、API／MCP 文件與測試報告。

先備知識

- 完成 Section 14：45 項自動化測試通過，`docs/測試報告.md` 已有安全與 RWD 的證據。
- Section 01 的 User Secrets 與 Section 10 的 OAuth 設定（`OAuth:Issuer`、`OAuth:McpResource`）；Section 05 的上游金鑰以 Data Protection 加密。
- 會用 `sqlcmd` 以系統管理員（sa）連線；知道 PFX 是「憑證加私鑰」的檔案。

觀念說明（20 分鐘）

開發環境與正式環境差在哪

項目	開發（Development）	正式（Production）	為什麼
設定的秘密	User Secrets	環境變數（或主機的祕密管理）	User Secrets 只存在開發者的電腦
示範資料	啟動時匯入	不匯入	正式資料庫不能出現測試帳號
第一位管理者	示範資料裡的 <code>admin</code>	<code>Bootstrap:AdminEmail</code> 、 <code>Bootstrap:AdminPassword</code> 只設一次	沒有示範資料時要有辦法建立第一位管理者
資料庫帳號	<code>sa</code>	<code>mcp_app</code> （只能讀寫資料）	網站被攻破時，攻擊者拿到的權限越小越好
結構變更	<code>dotnet ef database update</code>	系統管理員執行冪等 SQL 腳本	網站帳號沒有改結構的權限，變更要有人審閱
HTTP	允許 (<code>DisableTransportSecurityRequirement</code>)	只允許 https	OAuth 授權碼與 Token 不能在網路上明文傳送
OAuth 憑證	OpenIddict 開發憑證	從 PFX 檔載入	重新啟動、換主機後，已簽發的 Token 仍能驗證
Data Protection 金鑰	使用者設定檔	指定資料夾並以 DPAPI 加密	登入 Cookie、防偽權杖、上游金鑰都靠它
套件版本	依 <code>Directory.Packages.props</code>	<code>packages.lock.json</code> + <code>--locked-mode</code>	部署時裝到的套件與測試時完全相同

部署的順序



圖 15-1 部署與驗收的五個步驟

編號	做了什麼	做錯的後果（本單元實測或推得的）
①	<code>dotnet restore --locked-mode</code> 、 <code>dotnet publish -c Release</code>	沒有鎖定檔時，部署當天的套件版本可能與測試時不同
②	sa 建立資料庫、執行冪等腳本、建立 <code>mcp_app</code>	腳本在第一次實測時失敗（錯誤 207，見〈常見錯誤〉），資料庫停在「做一半」
③	HTTPS 憑證、Token 簽章憑證、 <code>keys</code> 資料夾	沒有 <code>keys</code> ：重新啟動後所有人被登出（實測）
④	以環境變數啟動 Production；第一次帶 Bootstrap	忘記移除 Bootstrap：不會重複建立（程式會檢查），但密碼留在主機環境變數裡
⑤	用全新帳號跑完驗收流程	只有頁面能開不算數：大綱規定至少一個真實 MCP Client 完成端到端呼叫

Data Protection 金鑰為什麼要持久化

ASP.NET Core 用 Data Protection 加密三樣東西：**登入 Cookie**、**防偽權杖**、本平台的**上游 API 金鑰**（Section 05）。加解密用的金鑰（Key Ring）預設存在執行帳號的使用者設定檔裡；換一個帳號、換一台主機、或在 IIS 沒有載入使用者設定檔的情況下，會變成另一組金鑰。結果：

- 已登入的人全部被登出（本單元實測：舊 Cookie 被轉到登入頁，日誌**沒有任何警告**）。
- 資料庫裡已加密的上游金鑰**永遠解不開**，Gateway 呼叫需要金鑰的 API 全部失敗。

所以正式環境要把金鑰放在固定資料夾、固定應用程式名稱，並把 `keys` 資料夾與資料庫一起備份。

限制與注意事項

- 本單元的「教室主機」以同一台 Windows 11 模擬：網站以 Kestrel 直接提供 https（`https://localhost:7443`）。實測機器有 IIS，但沒有安裝 ASP.NET Core Hosting Bundle（需系統管理員權限），IIS 的步驟寫在 `docs/安裝說明.md` 第八節，**未實際執行**。

- HTTPS 憑證使用匯出的 .NET 開發憑證（CN=localhost），只適合本機；正式主機要用主機網域的憑證。
- 雲端 AI 服務的 Connector 需要外部連得到的 https 網址；本單元的 Connector 是官方 MCP C# SDK 的測試用戶端。
- UseHsts() 對 localhost 不送出 Strict-Transport-Security 標頭（實測回應沒有這個標頭），正式網域才會送。

名詞對照

名詞	白話說明	在本課程中是
發行 (Publish)	編譯成可以直接執行的檔案集合	dotnet publish -c Release
冪等腳本	執行幾次結果都一樣的 SQL 腳本	sql/deploy/01_migrate_idempotent.sql
最小權限	只給完成工作所需的權限	mcp_app : db_datareader 、 db_datawriter
金鑰持久化	把 Data Protection 的金鑰存到固定位置	DataProtection:KeysPath
Bootstrap	系統第一次啟動時的初始化	建立第一位管理者
鎖定檔	記錄每個套件的確切版本與雜湊	packages.lock.json

教師示範 (25 分鐘)

示範情境：把 Section 15 的網站部署成正式環境（全新資料庫、專用帳號、https），用兩個全新的會員跑完成果驗收流程，再示範金鑰持久化與備份還原。所有密碼都是隨機產生、存在部署資料夾的 `secrets`，不出現在程式碼與講義裡。

示範步驟

1. 資料庫 (sa)：建立 `MCPServerManag_Prod`、執行幕等腳本、建立 `mcp_app`。實測：

```
已套用的 Migration: InitialCreate、AddVersionSpecFields、AddPublicationReview、AddAccessReview、AddOpenIdDict  
帳號      角色  
mcp_app   db_datareader  
mcp_app   db_datawriter
```

以 `mcp_app` 登入測試權限：

```
SELECT COUNT(*) FROM AuditLogs;    → 0 (可以讀)  
DROP TABLE AuditLogs;             → 訊息 3701:無法 卸除 資料表 'AuditLogs'，原因是它不存在或您沒有權限。  
ALTER TABLE AuditLogs ADD Hack...; → 訊息 1088:找不到物件 "AuditLogs"，原因是物件不存在，或您沒有權限。
```

再執行一次幕等腳本：沒有任何錯誤，`EFMigrationsHistory` 仍是 5 筆。

2. 發行並以 Production 啟動（第一次帶 Bootstrap）。日誌：

```
2026-09-26 21:52:21 warn: Program[0] 已建立第一位管理者 admin@classroom.test。請立即移除 Bootstrap 環境變數。  
2026-09-26 21:52:21 info: Microsoft.Hosting.Lifetime[14] Now listening on: https://localhost:7443  
2026-09-26 21:52:21 info: Microsoft.Hosting.Lifetime[0] Hosting environment: Production
```

`keys` 資料夾出現 `key-e13f8b1a-...xml`。不帶 Token 呼叫 `https://localhost:7443/mcp` → 401，`WWW-Authenticate: Bearer resource_metadata="https://localhost:7443/.well-known/oauth-protected-resource/mcp"`：Section 10 的探索資訊已自動變成正式網址。

3. 成果驗收（管理者先登錄 Connector）。實測：

```
== 1. 新會員註冊      provider@classroom.test、learner@classroom.test → /Member  
== 2. API 上架核准    建立草稿 → 送審 → ✓ 已核准並發布 課程查詢 v1 → ✓ 映射為工具 courses_search  
                    (核准使用前，學員的 Connector 登入成功但 tools/list 0 個工具)  
== 3. 使用申請核准    ✓ 已核准，並建立 30 天、每日 100 次的使用授權。  
== 4. Connector 呼叫   tools/list: 共 1 個工具  
                    courses_search {"level": "進階"} → isError = False, CorrelationId: d25c7bd98daa4697  
                    courses_search {"level": "高手"} → [INVALID_ARGUMENTS]  
== 5. 儀表板查詢      工具執行=2、成功率=50.0%、平均延遲 (ms) =26、P95 延遲 (ms) =53  
== 6. 撤銷後拒絕      ✓ 已撤銷。→ 同一位學員重新連線: tools/list 0 個; courses_search → [GRANT_REVOKED] 使用授權已被撤銷。
```

MCP Server API 平台API 目錄會員中心管理後臺系統管理員登出

管理後臺

總覽

儀表板

API 管理

資料庫概況

上架審核

使用申請審核

Grant 管理

API 測試台

MCP 工具映射

Connector 登錄

會員管理

呼叫紀錄

稽核軌跡

呼叫紀錄

起日2026/09/20迄日2026/09/26會員工具名稱錯誤代碼CorrelationId

查詢有 CorrelationId 時只依它查詢，忽略其他條件；最多顯示 200 筆。

時間	會員	API／工具	結果	耗時	CorrelationId
2026-09-26 22:01	課程學員 classroom-mcp-client	課程查詢 v1 courses_search	Denied GRANT_REVOKED	0 ms	43f3389541ac4e6b
2026-09-26 22:01	課程學員 classroom-mcp-client	課程查詢 v1 courses_search	Denied GRANT_REVOKED	0 ms	190bee93b44b4665
2026-09-26 22:01	課程學員 classroom-mcp-client	課程查詢 v1 courses_search	InvalidInput INVALID_ARGUMENTS	0 ms	54a684fabeab41f9
2026-09-26 22:01	課程學員 classroom-mcp-client	課程查詢 v1 courses_search	Success	53 ms	d25c7bd98daa4697
2026-09-26 21:59	課程學員 classroom-mcp-client	課程查詢 v1 courses_search	Denied GRANT_REQUIRED	0 ms	f7ff230e4e12463c
2026-09-26 21:59	課程學員 classroom-mcp-client	課程查詢 v1 courses_search	Denied GRANT_REQUIRED	0 ms	b7011d3b844a4e59
2026-09-26 21:59	課程學員 classroom-mcp-client	課程查詢 v1 courses_search	Denied GRANT_REQUIRED	0 ms	1b2b03f86af141a8
2026-09-26 21:59	課程學員 classroom-mcp-client	課程查詢 v1 courses_search	Denied GRANT_REQUIRED	0 ms	4b951ef6fbe54f6c
2026-09-26 21:55	課程學員 classroom-mcp-client	課程查詢 v1 courses_search	Denied GRANT_REQUIRED	0 ms	b6e02ca40b574c99

圖 15-2 正式網站 (https://localhost:7443) 的呼叫紀錄：整條驗收流程的足跡

畫面重點（由下往上是時間順序）：

- 1. ①下方數列 **Denied**／**GRANT_REQUIRED**：21:59 之前，學員還沒獲准，Connector 的呼叫被當成「沒有這個工具」，但紀錄照寫。
- 2. ② **Success**，**53 ms**，**CorrelationId d25c7bd98daa4697**：核准後第一次成功呼叫上游。
- 3. ③ **InvalidInput**／**INVALID_ARGUMENTS**：參數「高手」不在列舉內，0 ms，沒有送出請求。
- 4. ④最上方兩列 **Denied**／**GRANT_REVOKED**：撤銷後同一位學員的呼叫被拒。
- 5. ⑤每一列的會員都是「課程學員」、Client 都是 **classroom-mcp-client**：Section 10 的 Issuer + Subject 對應在正式環境一樣運作。
- 6. 金鑰持久化。以管理者登入並存下 Cookie，然後重新啟動網站：

重新啟動前	用舊 Cookie 開 /Admin/Dashboard → 200
重新啟動（有 DataProtection:KeysPath）	用舊 Cookie 開 /Admin/Dashboard → 200
重新啟動（沒有設定 KeysPath）	用舊 Cookie 開 /Admin/Dashboard → 302 https://localhost:7443/Account/Login

第三種情況的網站日誌沒有任何警告。

- 5. 備份與還原演練：

BACKUP DATABASE 已於 0.017 秒內成功處理了 1026 頁 (471.277 MB/sec)。
檔案 1 上的備份組有效。
RESTORE DATABASE 已於 0.013 秒內成功處理了 1026 頁 (616.286 MB/sec)。
← RESTORE VERIFYONLY
← 還原成 MCPServerManag_RestoreTest

會員 | 版本 | 呼叫紀錄 | 稽核紀錄

正式3 | 1 | 12 | 6

還原副本3 | 1 | 12 | 6

演練後刪除還原副本。

MCP Server 治理平台實戰手冊（30HRS）

— 226 —

學員實作（60 分鐘）

範例專案：`範例/15_部署安裝與成果驗收/McpPlatform/`（從 Section 14 複製）。本單元新增 `Directory.Build.props`（鎖定檔）、各專案的 `packages.lock.json`、`appsettings.Production.json`、`sql/deploy/`（00~04 五支腳本）、`deploy/start-production.sample.ps1`、`docs/`（安裝說明、角色操作手冊、API 與 MCP 文件，並更新測試報告）；修改 `Program.cs`（金鑰持久化、正式環境憑證、只允許 https、Bootstrap）、`IdentitySeeder`（`EnsureFirstAdminAsync`）、`MigrationAddPublicationReview`（`EXEC` 包裝）。

步驟一 套件鎖定與部署 Migration 腳本（10 分鐘）

目的：部署時裝到的套件與資料庫結構，都和測試時一模一樣。

```
<!-- 範例/15_部署安裝與成果驗收/McpPlatform/Directory.Build.props -->
<Project>
  <!-- 全方案共用的建置設定（Section 15）。套件版本在 Directory.Packages.props，這裡放建置行為。 -->
  <PropertyGroup>
    <!-- 每個專案產生 packages.lock.json：部署時以鎖定模式還原（dotnet restore 加 locked-mode 參數），裝到完全相同的套件 -->
    <RestorePackagesWithLockFile>true</RestorePackagesWithLockFile>
  </PropertyGroup>
</Project>
```

```
dotnet build # 6 個專案各產生 packages.lock.json
dotnet restore --locked-mode # 部署時：鎖定檔與專案不一致就失敗
dotnet ef migrations script --idempotent -o sql/deploy/01_migrate_idempotent.sql `
  --project src/McpPlatform.Infrastructure --startup-project src/McpPlatform.Web
```

第一次在全新資料庫執行這份腳本時失敗（錯誤 207），原因是 Section 06 的 Migration 裡有一段自訂 SQL。修正方式是用 `EXEC` 包起來：

```
// 範例/15_部署安裝與成果驗收/McpPlatform/src/McpPlatform.Infrastructure/Data/Migrations/20260926112151_AddPublicationReview.cs（節錄）
// Section 15：用 EXEC 包起來。部署腳本會把整段放進 IF NOT EXISTS 的批次裡，
// SQL Server 編譯該批次時 SnapshotJson 欄位還不存在，直接寫 UPDATE 會得到錯誤 207。
migrationBuilder.Sql("""
    EXEC(N'
        UPDATE r SET SnapshotJson =
```

程式說明：

片段	做了什麼	為什麼這樣寫
<code>Directory.Build.props</code>	MSBuild 最早匯入的檔案，對方案內每個專案生效	放在 <code>Directory.Packages.props</code> 也能寫，但本單元實測那裡寫錯時的錯誤訊息很難看懂（見〈常見錯誤〉）
<code>--idempotent</code>	每個 Migration 包在 <code>IF NOT EXISTS (... __EFMigrationsHistory ...)</code> 裡	同一份腳本可以對任何版本的資料庫執行，只補上缺的部分
<code>EXEC(N'...')</code>	把 <code>UPDATE</code> 變成執行時才編譯的動態 SQL	批次編譯時欄位還不存在；包成字串後，執行到這一行時欄位已經加上了
修改已套用過的 Migration	只改寫法，不改效果	開發資料庫已套用過， <code>__EFMigrationsHistory</code> 有紀錄，不會重跑；新資料庫用新寫法

預期結果：全新資料庫執行後 5 個 Migration 都在；再執行一次沒有錯誤。

步驟二 專用 SQL 帳號（10 分鐘）

目的：網站用的帳號只能讀寫資料。

```
-- 範例/15_部署安裝與成果驗收/McpPlatform/sql/deploy/02_create_app_login.sql（節錄）
USE master;
IF NOT EXISTS (SELECT 1 FROM sys.server_principals WHERE name = N'$(AppLogin)')
    CREATE LOGIN [$(AppLogin)] WITH PASSWORD = N'$(AppPassword)', CHECK_POLICY = ON, DEFAULT_DATABASE = [$(DbName)];
GO
USE [$(DbName)];
IF NOT EXISTS (SELECT 1 FROM sys.database_principals WHERE name = N'$(AppLogin)')
    CREATE USER [$(AppLogin)] FOR LOGIN [$(AppLogin)];
ALTER ROLE db_datareader ADD MEMBER [$(AppLogin)]; -- 讀取
ALTER ROLE db_datawriter ADD MEMBER [$(AppLogin)]; -- 新增、修改、刪除資料列
```

片段	做了什麼	為什麼這樣寫
<code>\$(AppPassword)</code>	sqlcmd 變數，執行時以 <code>-v AppPassword="..."</code> 傳入	密碼不寫進任何檔案
<code>CHECK_POLICY = ON</code>	套用 Windows 密碼原則	大綱：不修改 SQL Server 的密碼原則，弱密碼會被拒
<code>IF NOT EXISTS</code>	可重複執行	重新部署時不會因為帳號已存在而失敗
只有 <code>db_datareader</code> 、 <code>db_datawriter</code>	沒有 <code>db_ddladmin</code> 、 <code>db_owner</code>	實測 <code>DROP TABLE</code> 、 <code>ALTER TABLE</code> 都被拒；網站也不在啟動時自動套用 Migration

預期結果：示範步驟 1 的權限測試結果。

步驟三 正式環境的程式設定（20 分鐘）

目的：金鑰持久化、Token 憑證、只允許 https。

```
// 範例/15_部署安裝與成果驗收/McpPlatform/src/McpPlatform.Web/Program.cs (Data Protection)
var keysPath = builder.Configuration["DataProtection:KeysPath"];
if (!string.IsNullOrEmpty(keysPath))
{
    var dataProtection = builder.Services.AddDataProtection()
        .SetApplicationName("McpPlatform") // 固定名稱：部署路徑改變也用同一組金鑰
        .PersistKeysToFileSystem(new DirectoryInfo(keysPath));
    if (OperatingSystem.IsWindows())
        dataProtection.ProtectKeysWithDpapi(); // 金鑰檔本身再用 Windows DPAPI 加密
}

// 範例/15_部署安裝與成果驗收/McpPlatform/src/McpPlatform.Web/Program.cs (OpenIddict 憑證與 https)
if (builder.Environment.IsDevelopment())
    options.AddDevelopmentEncryptionCertificate()
        .AddDevelopmentSigningCertificate(); // 開發用憑證
else
{
    var tokenCertificate = X509CertificateLoader.LoadPkcs12FromFile( // Section 15：正式環境從 PFX 載入，重啟後 Token
        仍可驗證
        oAuth.CertificatePath, builder.Configuration["OAuth:CertificatePassword"]);
    options.AddSigningCertificate(tokenCertificate)
        .AddEncryptionCertificate(tokenCertificate);
}
options.DisableAccessTokenEncryption(); // Token 只簽章不加密，課堂上可以解開來看
var aspNetCore = options.UseAspNetCore()
    .EnableAuthorizationEndpointPassthrough(); // ◎ 授權端點交給 AuthorizationController
if (builder.Environment.IsDevelopment())
    aspNetCore.DisableTransportSecurityRequirement(); // 只有開發環境允許 http；正式環境一律 https
```

程式說明：

片段	做了什麼	為什麼這樣寫
沒設 <code>KeysPath</code> 就不設定	開發環境維持預設	開發資料庫裡已加密的金鑰是用預設金鑰加密的，改了反而解不開
<code>SetApplicationName("McpPlatform")</code>	固定應用程式名稱	名稱預設與安裝路徑有關，換資料夾部署就會變成另一組金鑰
<code>ProtectKeysWithDpapi()</code>	金鑰檔以 Windows DPAPI 加密	拿到 <code>keys</code> 資料夾的人不能直接讀出金鑰
<code>X509CertificateLoader.LoadPkcs12FromFile</code>	.NET 10 載入 PFX 的新 API	舊的 <code>new X509Certificate2(path, password)</code> 在 .NET 9 起標記為過時
同一張憑證簽章與加密	憑證建立時同時給 <code>DigitalSignature</code> 與 <code>KeyEncipherment</code> 用途	課堂只需一個 PFX；正式環境也可以分成兩張
<code>DisableTransportSecurityRequirement</code> 只在開發環境	正式環境 OpenIddict 拒絕 http 請求	授權碼與 Token 不能明文傳送

`appsettings.Production.json` 只放不含祕密的設定：記錄層級（`Microsoft.AspNetCore`、`EntityFrameworkCore`、`OpenIddict`、`ModelContextProtocol` 設為 `Warning`）與單行、帶時間的主控台格式；所有祕密與網址由環境變數提供（`deploy/start-production.sample.ps1`）。

預期結果：以 Production 啟動成功，只聽 https；示範步驟 4 的三種結果。

步驟四 第一位管理者（10 分鐘）

目的：正式資料庫沒有示範資料時，建立第一位管理者。

```
// 範例/15_部署安裝與成果驗收/McpPlatform/src/McpPlatform.Infrastructure/Identity/IdentitySeeder.cs (EnsureFirstAdminAsync 節錄)
if (!await roles.RoleExistsAsync(AdminRole))
    await roles.CreateAsync(new IdentityRole(AdminRole));
if ((await users.GetUsersInRoleAsync(AdminRole)).Count > 0)
{
    logger.LogInformation("已有管理者，略過第一位管理者的建立；請移除 Bootstrap 環境變數。");
    return;
}
```

片段	做了什麼	為什麼這樣寫
已有任何管理者就結束	Bootstrap 只在「一個管理者都沒有」時生效	忘記移除環境變數也不會多建帳號；之後的管理者一律由管理者在「會員管理」指派
<code>users.CreateAsync(admin, password)</code> （完整檔案）	走 Identity 的密碼規則與雜湊	弱密碼一樣會被拒，不會因為是 Bootstrap 就放寬
<code>Program.cs</code> 在所有環境呼叫	只要設定了 <code>Bootstrap:AdminEmail</code>	開發環境有示範資料的 <code>admin</code> ，所以不會觸發

預期結果：示範步驟 2 的日誌；第二次啟動（不帶 Bootstrap）不再建立。

步驟五 成果驗收與交付（10 分鐘）

目的：用全新帳號跑完大綱的驗收流程，並整理交付文件。

依示範步驟 3 的順序操作（可用瀏覽器，也可用教師的腳本）。交付清單：

交付項目	位置
原始碼	範例/15_部署安裝與成果驗收/McpPlatform/
套件鎖定資訊	各專案的 <code>packages.lock.json</code> 、 <code>global.json</code> （SDK 10.0.401）
Migration	<code>src/McpPlatform.Infrastructure/Data/Migrations/</code> 、 <code>sql/deploy/01_migrate_idempotent.sql</code>
設定範例	<code>appsettings.Production.json</code> 、 <code>deploy/start-production.sample.ps1</code>
安裝說明	<code>docs/安裝說明.md</code>
角色操作手冊	<code>docs/角色操作手冊.md</code>
API／MCP 文件	<code>docs/API與MCP文件.md</code>
測試報告	<code>docs/測試報告.md</code> （Section 14 的 45 項測試＋本單元的部署驗收）

預期結果：圖 15-2；`docs/測試報告.md` 第四節記錄每一步的結果。

檢查點

- ☐ 全新資料庫以冪等腳本建立，執行兩次都沒有錯誤
- ☐ 網站以 `mcp_app` 連線；`mcp_app` 無法刪表、改表
- ☐ Production 只聽 https；不帶 Token 呼叫 `/mcp` 得到指向 https 的 `resource_metadata`
- ☐ 重新啟動後管理者仍登入；`keys` 資料夾與資料庫都納入備份
- ☐ 驗收流程六個步驟都有結果；撤銷後同一位使用者被拒絕

常見錯誤

以下都是撰寫本單元時實際遇到的原文。

症狀 (錯誤訊息原文)	原因	處理
寫等腳本： 訊息 207 ... 無效的資料行名稱 'SnapshotJson'。 , __EFMigrationsHistory 少了 AddPublicationReview	Migration 裡的自訂 UPDATE 與新增欄位在同一個 IF NOT EXISTS 批次，編譯時欄位還不存在	自訂 SQL 用 EXEC(N'...') 包起來；已經做一半的全新 資料庫刪除重建
error NU1015: 下列 PackageReference 項目未指定版本: ...	在 Directory.Packages.props 加的 XML 註解裡寫了 --locked-mode，XML 註解不能含 --，整個檔案失效	註解改寫，不含連續兩個連字號
error MSB4024: 無法載入已匯入的專案檔 "... Directory.Build.props"。An XML comment cannot contain '--'	同一個原因，發生在 Directory.Build.props 時訊息比較直接	同上
Sqlcmd: 'BackupFile=C:\Program Files\...' TargetDb=...': 引數無效。	-v 的值含空白，沒有再加一層引號	-v BackupFile="C:\Program Files\... \xxx.bak" (PowerShell 中寫成 "BackupFile='"\$bak`"'")
重新啟動網站後所有人被登出，日誌沒有任何錯誤	沒有設定 DataProtection:KeysPath，金鑰與上次不同	設定金鑰資料夾並一起備份
回應沒有 Strict-Transport-Security 標頭	UseHsts() 預設排除 localhost	正常現象；正式網域會送出

單元驗收（15 分鐘）

驗收標準：新會員註冊 → API 上架核准 → 使用申請核准 → Connector 呼叫 → 儀表板查詢 → 撤銷後拒絕，整條流程可重現。

驗收方式：

1. **整條流程可重現：**在全新的正式資料庫上，以兩個新註冊的會員依序完成六個步驟（示範步驟 3）；每一步的畫面訊息與圖 15-2 的紀錄一致。
2. **真實 MCP Client：**Connector 是官方 MCP C# SDK 的用戶端，經 OAuth 授權碼 + PKCE（https）取得 Token 後呼叫 `/mcp`，不是用一般 REST 請求模擬。
3. **撤銷後拒絕：**撤銷後同一位學員重新連線，`tools/list` 0 個、`tools/call` 得到 `GRANT_REVOKED`。
4. **大綱的最低通過條件：**一般會員不能進入後臺、未核准 API 不能執行、未授權會員不能呼叫（Section 14 測試）；撤銷、停權、下架可阻擋新呼叫（Section 11、12、本單元）；至少一個真實 MCP Client 完成端到端呼叫（本單元）。
5. **交付：**`docs/` 四份文件、部署腳本與設定範例齊全，別人可以照 `docs/安裝說明.md` 重現部署。

回顧：正式環境的祕密放環境變數、資料庫用最小權限帳號、結構變更用幕等腳本、Token 憑證與 Data Protection 金鑰都要持久化並備份；驗收要用全新帳號、真實 MCP Client 跑完整條流程。十五個單元下來，平台從一個 MVC 版型長成一個有審核、授權、OAuth、Gateway、額度、儀表板、測試與部署文件的 MCP Server API 治理平台。

延伸閱讀

- [在 Windows 上使用 IIS 裝載 ASP.NET Core](#)。
- [ASP.NET Core Data Protection：金鑰儲存提供者與 DPAPI](#)。
- [EF Core：套用 Migration（SQL 腳本、幕等腳本、Bundle）](#)。
- [NuGet 鎖定檔（packages.lock.json）](#)。

索引

依英文字母與中文詞彙排列；頁碼為該詞彙討論最集中的章節起始頁，可直接點選跳頁。

A	
Access Token	5, 102, 132, 146, 147, 167
Admin Area	18, 49, 50, 69, 85, 123
Anti-forgery	50, 210
API Gateway	70, 118, 123, 129, 167, 196
ApiGrant	5, 36, 99, 164, 182, 185
ApiInvocationService	115, 130, 131, 132, 139, 172
ASP.NET Core MVC	34, 65
Audience	5, 146, 147
C	
Chart.js	201
Connector	5, 150, 153, 220
Controller	18, 53, 56, 76
Cookie	50, 139, 147, 182, 223
CorrelationId	118, 136, 139, 172, 198, 206
CSRF	28, 34, 65, 207
D	
Data Protection	70, 220
DbContext	39, 42, 44, 156
DTO	18, 44, 70, 109
E	
EF Core	36, 86
H	
HttpClientFactory	115
HTTPS	5, 16, 70, 220
I	
Identity	17, 36, 50, 76, 149, 182
IIS	220
inputSchema	5, 70, 129
Issuer + Subject	147, 166, 222, 225
J	
JSON Schema	44, 65, 67, 116, 131, 132
K	
KPI	193, 212
M	
MCP	2, 70, 132, 136, 139, 231
MCP Server	5, 231
MCPServerManag	11, 44, 49, 214
Migration	36, 76, 94, 156, 220
O	
OAuth	5, 156, 220
OpenIddict	147, 172, 223, 227
OwnerId	19, 21, 24, 28, 39, 76
P	
P95 延遲	193, 214
PKCE	147, 231
Policy	50, 210

R	
Rate Limit	164
Razor Layout	18, 139
Redirect URI	147, 214
Repository	21, 28, 34, 44, 76, 139
REST	2, 231
rowversion	36, 86, 94, 109, 112
RWD	17, 18, 206, 207, 222
S	
Scope	5, 99, 118, 132, 147, 164
Service	18, 76
SQL Server	2, 38, 39, 44, 214
SSMS	4, 38, 42, 44, 48, 64
SSRF	76, 86, 113, 115, 132, 207
Streamable HTTP	5, 129
T	
Tag Helper	18, 64, 156
ToolDefinition	39, 42, 44, 129, 182, 188
Tools 映射	39, 70, 116, 128, 130, 136
tools/call	129, 167, 172
tools/list	129, 150, 172, 176
U	
User Secrets	11, 214
UTC 時間	36, 196
V	
ViewModel	18, 60

中文詞彙	
下架	5, 83, 109, 132, 136, 179
上架	2, 24, 39, 42, 83, 179
不可變版本快照	5, 39, 83
允許清單	115, 132, 210
反向驗證	17, 34, 49, 65, 82, 98
外鍵與索引	36
本人資料範圍	50, 86
示範 API	2, 28, 72, 117, 118, 123
示範資料	36, 50, 214, 223
交易	44, 83, 100, 102, 104, 112
有效期間	5, 99, 150, 172
角色	2, 50, 179, 210, 227
使用申請	5, 39, 42, 99
併發衝突	39, 44, 49, 83, 101, 207
呼叫日誌	39, 81, 112, 118, 193, 220
版本管理	67
狀態轉換	83
核准發布	5, 39, 83, 99, 115, 185
送審	5, 67, 83
退回修訂	3, 5, 21, 39, 70, 83
停權／復權	53, 60, 170, 172, 177, 179
密碼雜湊	50
帳號鎖定	2, 50, 227
授權快取	5, 164, 181, 188
部署	150, 220
備份與復原	70, 220
越權	51, 76, 170, 177, 207
逾時	11, 115
撤銷	5, 99, 164, 231
端到端測試	207, 223, 231
儀表板	39, 167, 182, 193, 212, 231
審核軌跡	90, 179, 193
稽核	86, 90, 102, 182, 192, 193
幂等	115, 223
憑證加密	67, 223, 227
憑證遮罩	207
環境變數	223
額度	5, 99, 164, 207