ChatGPT

90 天全端開發學習計畫：打造 AI 智能個人部落格

第 1 週：Git 版本控制與 CLI 環境

圖注：ChatGPT 企業版的側欄中可以找到 OpenAI 的 Codex 編程代理功能，用戶可綁定 GitHub 專案並發送任務給 AI 自動完成程式編寫、除錯與提交 [1] [2] 。本週將以開發環境設置與版本控制為主軸，帶領學習者熟悉 Git 基本操作、命令列工具及 AI 開發助理的使用方式。透過每日練習，建立專案倉庫、進行版本提交與分支操作，為之後的開發打好協作基礎。

- **Day 1：**
  理論概念： 瞭解版本控制的重要性以及 Git 的工作流程，包括工作目錄、暫存區、版本庫等概念 [3] 。學習常見 CLI 操作（如檔案導覽、編輯）以熟悉命令列介面。
  學習資源： 推薦觀看 Git 入門教學影片（如 Bilibili 的「Git 初學者教程：1小時學會Git」系列） [4] ，快速掌握 Git 基本操作；參考 Will 保哥的「30 天精通 Git 版本控管」文章以獲取系統化知識 [5] 。
  AI 工具運用： 使用 Cursor IDE 開啟終端，透過內建的 AI 助理詢問「如何使用 Git 初始化倉庫？」等問題，由 AI 提供指令範例。若在操作 Git 過程中出現錯誤訊息，可將錯誤貼給 Codex 請求解釋與解決方案。
  實作任務： 安裝 Git 並進行全域配置（user name、email）。在本機建立一個新的專案資料夾，使用 `git init` 進行版本庫初始化，創建一個 README.md 檔案並撰寫專案簡介。將變更加入暫存區並完成第一次提交。嘗試使用 `git log` 查看提交記錄。
  驗收標準： 本地成功建立 Git 倉庫，執行 `git status` 顯示工作目錄乾淨無異常。README.md 能透過 `git log` 看到初次提交記錄。若將專案推送至 GitHub，遠端存儲庫能正確顯示提交內容。

- **Day 2：**
  理論概念： 瞭解遠端版本庫與本地版本庫的關係，學習 GitHub 平台的作用。掌握 Git 常用命令如 `git clone`、`git add`、`git commit`、`git push` 等的用法 [6] [7] 。認識 SSH Key 並了解其在身份驗證中的角色。
  學習資源： 參考廖雪峰的 Git 教學章節，逐步操作克隆遠端倉庫、提交更新等流程 [3] [8] 。觀看 YouTube「學習Git版本控管：新手上路篇」視頻（Will 保哥主講）瞭解實務操作技巧 [9] 。
  AI 工具運用： 利用 Codex 幫助設定 SSH Key：例如在 Cursor 中輸入「如何產生 SSH Key 並設定到 GitHub？」讓 AI 給出步驟說明。當執行 Git 操作遇到衝突或錯誤時，將錯誤訊息貼給 AI 要求排解。
  實作任務： 在 GitHub 上建立一個空的遠端倉庫，將 Day1 建立的本地倉庫綁定遠端（`git remote add origin ...`），並嘗試使用 `git push` 將本地提交推送上傳。接著，在 GitHub 網站上編輯 README.md 並提交變更，再使用 `git pull` 將遠端更新拉取回本地，體驗協作流程。
  驗收標準： 本地倉庫成功與遠端 GitHub 倉庫連線，GitHub 上可以看到本地提交的內容。能夠透過

`git pull` 拉取到遠端修改，並透過 `git log` 驗證合併結果無誤。SSH Key 設定正確的情況下，推送和拉取過程無需重複輸入帳號密碼。

- **Day 3：**
  理論概念： 深入學習 Git 分支（branch）概念，理解為何使用分支來開發新功能或修復 bug。掌握建立、切換與合併分支的流程，了解 Fast-Forward 與三方合併等合併方式的差異 [10] 。並初步認識 Pull Request（PR）的意義及流程，為後續團隊協作做準備。
  學習資源： 閱讀 Pro Git 第三章關於分支的章節，配合示意圖理解 Git 分支工作流程 [11] 。參考「Git Flow 工作流程」文章或圖表，以視覺化方式了解多人協作的典型流程（如開發分支、發佈分支等） [12] 。
  AI 工具運用： 使用 Cursor 的多檔案編輯功能，讓 AI 協助同時更新多處與分支相關的說明文件。例如，高亮 README 中的分支說明段落並要求 Codex 產生更清晰的解釋。如果對合併衝突的解決感到疑惑，可將衝突區塊貼給 AI 要求給出解決步驟。
  實作任務： 在本地倉庫建立一個新分支（例如 `dev`），在新分支中對 README.md 增加一行「測試分支功能」文字。切換回主分支，模擬同一行文字也被修改，然後嘗試將 `dev` 分支合併回主分支以觸發合併衝突。解決衝突後完成合併，最後將主分支推送至遠端並在 GitHub 上建立一個 Pull Request，觀察 PR 中的變更紀錄。
  驗收標準： 本地成功建立並切換分支，模擬的衝突已透過正確步驟手動合併解決 [13] 。Pull Request 順利建立，並在 GitHub 網站上顯示正確的差異內容。最終主分支包含來自 `dev` 分支的修改，版本歷史清晰無誤。

- **Day 4：**
  理論概念： 學習使用 CLI 提高工作效率的技巧，例如利用 `grep` 搜尋代碼、`ssh` 遠端連線、`curl` 發送 HTTP 請求等常用指令。了解 Package Manager (如 `npm`、`pip`) 在 CLI 下的使用，為後續裝置相依套件做準備。並熟悉 VS Code/Cursor 終端整合操作，加速在 IDE 中執行命令的流程。
  學習資源： 參閱 Linux 常用指令中文教程，重點關注檔案操作和網路相關指令。觀看 Bilibili 上的「CLI 生產力工具推薦」視頻，瞭解一些提高效率的 CLI 工具（如 `fzf` 模糊搜索等）。閱讀 npm 官方文件的 CLI 章節，掌握在命令列下初始化與管理 Node 專案的方法。
  AI 工具運用： 善用 Cursor IDE 內建終端的代碼補全功能。例如，在終端輸入部分指令後按下 `Tab`，觀察 AI 的補全建議。遇到不熟悉的 CLI 工具時，直接詢問 Codex 該工具的用途和簡單範例（如「curl 指令如何發送 GET 請求？」AI 將給出格式與說明）。
  實作任務： 練習透過 CLI 建立前端專案目錄結構：例如使用 `mkdir` 建立 `frontend` 資料夾，裡面再建立 `index.html`、`styles` 資料夾等。利用 `npm init -y` 在專案內初始化 Node 模組（雖然本週未正式用到，但可提前熟悉）。另外，使用 `pip` 在全域安裝 `fastapi` 及 `uvicorn`，確保 Python 執行環境準備就緒。
  驗收標準： 能夠在不依賴圖形介面的情況下完成基本檔案操作，所有步驟皆透過 CLI 成功執行。`npm` 或 `pip` 等套件管理工具運行正常（例如 `node -v`、`python -m pip --version` 顯示正確版本）。透過 AI 輔助下達的命令無誤，結果與預期一致。

- **Day 5：**
  理論概念： 綜合本週所學，整理出一套使用 Git 與 CLI 的日常工作流程。例如每日開新分支開發新功能、定期將進度推送到遠端備份、使用 CLI 腳本自動化重複任務等。理解良好提交訊息的重要性，以及如何撰寫語意清晰的 commit message。
  學習資源： 閱讀《Git 團隊協作指南》中的工作流程章節，了解在團隊中使用 Pull Request 開發的模式。參考 Angular Commit Message 規範或 Conventional Commits 規範，學習撰寫結構化的提交訊息以便於日後變更記錄追蹤。
  AI 工具運用： 嘗試讓 Codex 自動生成 commit message：在 Cursor 中選取代碼更動區塊，然後輸入指令例如「產生描述上述變更的提交訊息」，AI 將給出清晰的訊息內容。未來每日工作開始時，可請 AI 列出當天待辦任務清單，作為開發指南。
  實作任務： 將本週練習的專案推送至 GitHub 並建立一個README，內容包含對專案的簡介和目錄結構

說明。練習使用 GitHub Issue 記錄一項待辦功能（例如「撰寫初始首頁 HTML」），並在提交訊息中加入關聯的 Issue 編號。最後，模擬一天的工作流程：從 Issue 開新分支進行修改、提交並PR，然後合併回主分支。

驗收標準： 提交訊息符合規範，例如包含動詞開頭的簡潔描述。GitHub 上的 Issue 狀態隨對應的 Pull Request 合併而自動關閉（若 commit message 中使用了關鍵字關聯）。整個流程順暢，說明學習者已能在日常開發中正確運用版本控制與CLI技能。

第 2 週：HTML/CSS 網頁設計入門

本週聚焦前端基礎，學習使用 HTML 建立網頁結構，透過 CSS 美化呈現，為部落格網站奠定視覺和佈局基礎。每一天循序漸進強化排版技能，並引導運用簡單的 AI 工具來加速切版和除錯。目標是打造出靜態的個人部落格頁面雛形，包括文章列表和文章內頁的版型。

- **Day 6：**
  理論概念： 認識 HTML 文件的基本結構，包括 `<!DOCTYPE>` 宣告、`<html>`、`<head>`、`<body>` 等標籤的作用。了解常用的區塊級元素與行內元素，例如段落 `<p>`、標題 `<h1>`~`<h6>`、超連結 `<a>`、圖片 `<img>` 等，理解它們在頁面中的語意意義與預設排列方式。
  學習資源： 推薦參考 MDN Web Docs 的 HTML 入門教學，快速瞭解元素標籤的用法與示例。可觀看「3 小時前端入門教程」的前半部分，涵蓋 HTML 基礎知識與實例演示 [14] [15]（該系列對零基礎者非常友好 [16]）。另可查閱菜鳥教程關於 HTML 的章節以獲取簡潔的語法說明。
  AI 工具運用： 在 Cursor 中新建一個 `index.html`，利用 AI 自動補全功能快速生成 HTML 標準文件結構（輸入 `html:5` 等縮寫讓 IDE 協助展開模板）。編寫過程中，如不確定某個標籤的用法，可直接詢問 Codex，例如：「`<section>` 和 `<div>` 有何差異？」AI 將回覆其語意上的差別與使用場合。
  實作任務： 建立網站首頁的 HTML 框架：包含標題（網站名稱）、一個導航欄（簡單的超連結列表，例如 Home, About），以及一個文章列表區塊。文章列表中先硬編兩篇文章項目，每項包括標題（使用 `<h2>`）、發布日期和摘要段落。確保 HTML 語意正確，例如使用 `<article>` 包裹每篇文章資訊。
  驗收標準： 在瀏覽器中打開 `index.html`，能看到正確的結構：頁首標題清晰可見，導航連結可點擊且水平方向排列，下面列出兩篇文章的標題和摘要。使用瀏覽器內建的開發者工具檢查元素，確認結構層次清晰無誤且無遺漏閉合標籤等語法錯誤。

- **Day 7：**
  理論概念： 學習 CSS 的基本語法與選擇器機制，了解如何將 CSS 樣式加入 HTML（行內樣式、`<style>` 區塊或外部 CSS 檔）。認識常用選擇器如元素選擇器、類別選擇器（`.class`）、ID 選擇器（`#id`）等，以及層級選擇器的作用。理解 CSS 權重的概念，知道如何處理樣式衝突。
  學習資源： 繼續觀看「3 小時前端入門教程」中關於 CSS 的章節，從選擇器到常用樣式屬性逐步講解 [17] [18]。參考 w3schools 或 MDN 上的 CSS 基礎教學文件，重點閱讀選擇器與盒模型章節。亦可使用 Khan Academy 中文站的互動課程練習簡單 CSS。
  AI 工具運用： 在 Cursor 中打開昨天的 `index.html`，新建一個 `style.css` 檔案並連結進 HTML。嘗試利用 Cursor 的片段補全：例如輸入 `h1 {` 後按下 Enter，看 AI 是否自動補上一些常見樣式屬性。當需要取得某種效果的 CSS 寫法時，可以直接問 Codex，例如：「如何讓圖片圓角？」，AI 將建議使用 `border-radius`.
  實作任務： 為首頁增加基本樣式：將頁首標題文字置中並加入背景色或底線區分；導航連結排成水平菜單（可使用 `display: inline-block` 或 `float` 方法）；文章列表部分，文章項目之間增加留白（例如 `margin-bottom`），標題文字調整顏色與大小，摘要文字使用較淺字體顏色以凸顯層次。練習使用各種類型的選擇器，例如用類別選擇器為導航鏈結設定樣式，用後代選擇器為文章段落設定不同字體大小。
  驗收標準： 頁面樣式美觀且各元素佈局合理。檢查 CSS，確認樣式外聯成功且瀏覽器已正確載入（可透過開發者工具的 Network 或 Elements 面板驗證）。例如，頁首應置中且突出，導航鏈結具備適當的間距與互動樣式（如 hover 變色），文章列表的每項內容分明易讀。

- **Day 8：**
  理論概念： 深入理解 CSS 盒模型[18]，包括內容區域、內邊距 (padding)、外邊距 (margin)、邊框 (border) 的概念及作用。學習區分塊級元素與行內元素在盒模型計算上的差異。探討常見的佈局方式，如使用 `width` 設定元素寬度、`margin: 0 auto` 讓區塊水平置中等技巧。
  學習資源： 查閱菜鳥教程中關於 CSS 盒模型的解說，配合示意圖理解各部分對元素大小和間距的影響[19]。觀看 YouTube 上帶有中文字幕的CSS Box Model講解影片，透過實際演示強化概念。閱讀知乎或部落格文章「CSS 盒模型完全指南」，掌握標準盒模型與IE舊版盒模型的差異，避免陷阱。
  AI 工具運用： 在開發者工具中使用 AI 協助調試盒模型：Cursor 內建瀏覽器預覽功能，當選取元素後，可詢問 Codex「為何此元素的寬度沒有如預期？」AI 將根據盒模型推論可能的原因（例如 padding 計入總寬度等）。也可以提供 CSS 片段請 AI 優化，例如：「這段 CSS 可以更簡潔嗎？」AI 或許會建議合併一些 margin 或使用簡寫屬性。
  實作任務： 將首頁版型進一步調整：限制主內容欄寬度（例如設定整體容器寬度為 800px 且水平置中），以免在大螢幕上過寬不易閱讀。調整文章項目的盒模型：例如為每個 `<article>` 容器增加 padding 讓內文不貼邊，設定 margin 底距讓相鄰文章間隔明顯，並加上邊框或背景底色區隔。確保對不同元素（標題、段落、影像等）設定適當的 margin 以維持整體美觀。
  驗收標準： 頁面在桌面瀏覽器下中央對齊且內容寬度適中。不同比例螢幕下版面無明顯顛倒或內容溢出。利用瀏覽器的檢查工具查看元素，確認 margin 和 padding 的應用符合預期（例如文字未貼邊，元素間距合理）。所有樣式修改均透過 CSS 控制，HTML 結構無需額外調整即可達成版面優化。

- **Day 9：**
  理論概念： 學習 CSS 布局的進階技巧：認識浮動 (float) 及其對文檔流的影響，以及清除浮動的方法[19]。了解行內元素轉換為區塊元素的方式（display 屬性），初探 CSS Flexbox 布局模型的概念，理解主軸、交叉軸和常用屬性如 `justify-content`、`align-items` 等，為未來的響應式布局奠基礎。
  學習資源： 參閱 MDN 上的「浮動和清除」教學文章，學習如何使用clearfix技巧清除浮動造成的佈局問題。觀看 Bilibili 上關於 Flexbox 的入門動畫教學（有中文字幕），直觀理解彈性盒布局如何運作。閱讀《CSS 世界》相關章節（如有中文翻譯）以深化對 CSS 佈局的理解。
  AI 工具運用： 如果在調整佈局時遇到疑難，可以將相關 HTML/CSS 片段交給 Codex，請教「為何這兩欄沒有並排顯示？」AI 將根據代碼推斷可能的原因，例如忘記設定父容器寬度或子元素浮動。當引入新的 CSS 概念（如 Flexbox）時，可請 AI 提供簡短示例來展示基本用法。
  實作任務： 優化網站導航列的實作：改用浮動或 Flexbox 重新排版導覽選單，使其水平置頂排列且兩端對齊（例如Logo或標題在左，選單項在右）。嘗試建立一個雙欄佈局的文章頁範本：左側主要內容（文章標題及內文），右側側邊欄（例如文章分類清單或關於我的簡介）。可先使用浮動實現，讓側邊欄靠右並具有固定寬度，主內容區自適應剩餘空間。必要時調整結構以適應此布局（例如將主要內容和側欄各包在一個容器中）。
  驗收標準： 導覽列在頁面頂部橫向排列整齊，無論視窗寬度變化，選單在一定範圍內都保持佈局（超出範圍時不會掉行）。文章頁雙欄範本有效：側邊欄內容獨立一列且寬度固定，主內容區能隨頁面拉伸而擴展。使用開發者工具檢查，兩欄沒有互相重疊，且清除了浮動影響（若採用浮動方案，頁腳不會被側欄浮動影響而上移）。

- **Day 10：**
  理論概念： 初步了解響應式設計的概念，明白網站需適應不同裝置螢幕（電腦、平板、手機）的需求。認識媒體查詢（media query）的語法與作用，例如如何針對最大寬度 `max-width` 條件調整版面。了解相對單位（%、em、rem、vw 等）和流式布局的概念，使版面具有一定彈性。
  學習資源： 閱讀 Bootstrap 官方文件對響應式設計的介紹部分，雖然尚未使用框架，但其中對響應式斷點和格線系統的說明值得參考。參考 MDN 關於 CSS 媒體查詢的指南，學習幾個簡單範例。觀看 YouTube 上「響應式網頁設計入門」的中文字幕影片，看看在不使用框架時如何自行實作簡單的 RWD。
  AI 工具運用： 請教 Codex 如何讓圖片在小螢幕上縮放：提供 HTML `<img>` 給 AI，詢問「如何讓圖片在行動裝置上自適應螢幕寬度？」AI 可能建議使用 CSS 的 `max-width: 100%; height: auto;`。也可以讓 AI 協助生成媒體查詢範例，例如「請提供一段CSS，在螢幕寬度小於600px時隱藏側邊欄」。

實作任務： 為之前製作的頁面新增一些響應式考量：例如撰寫一個媒體查詢，當瀏覽器寬度小於 600px 時，自動將文章雙欄佈局轉為單欄（側邊欄改為顯示在主內容下方或隱藏側欄）。調整文字大小使用相對單位，如將先前固定像素的字體大小改用 `em` 或 `rem` 表示，以便在不同裝置上有良好閱讀體驗。確保圖片和影片等媒體可以在小螢幕上縮放不超出螢幕範圍。
驗收標準： 在桌機與手機模擬模式下檢視頁面，佈局均能合理顯示：縮小瀏覽器時，側欄自動調整位置或消失而不影響閱讀，文字大小和行高在小螢幕上仍保持可讀性。使用 Chrome 開發者工具的裝置模擬（如 iPhone 尺寸）檢查，各項內容無溢出邊界或過小難以點擊的問題。CSS 媒體查詢能正確作用，對應條件下的樣式覆蓋生效。

- **Day 11：**
  理論概念： 小結一週所學，強調結構與樣式分離的重要性，以及良好前端程式碼風格（例如使用 BEM 方法論為 class 命名以增加可讀性）。理解在實務開發中，往往會借助前端框架或樣板，但掌握原生 HTML/CSS 能夠幫助定位問題與進階自定義。
  學習資源： 閱讀《深入淺出 Web 設計》相關章節，了解從設計圖切版到瀏覽器呈現的完整流程。參考一些開源個人博客的前端原始碼（例如檢視他人 GitHub Pages 的 HTML/CSS），對比自己的實作找出可以改進之處。
  AI 工具運用： 使用 Cursor 的 Compare 模式將自己寫的 HTML/CSS 和某個最佳實踐範例進行比較，請 AI 找出不同並建議改進。例如要求 Codex 審視 CSS 是否重複或不夠簡潔。也可以讓 AI 檢查 HTML 結構的可存取性（Accessibility），如圖片有無加上 `alt` 屬性等。
  實作任務： 整理本週產出的所有 HTML、CSS 檔案，確保命名與結構清晰。在 GitHub 建立一個新分支 `frontend-design`，將這些檔案提交上去，作為前端初版版型的里程碑。撰寫一份簡短的說明文件（可以在 README 增補）解釋目前實作了哪些頁面與樣式，以及未來待完善之處（例如尚未加入互動、假資料需要替換成動態內容等）。
  驗收標準： Pull Request 成功建立並通過審查合併，代表前端靜態頁面初版完成。所有 HTML 檔在 W3C Validator 上檢查無重大錯誤，CSS 檔透過 stylelint 或 VSCode 提示無重大語法問題。說明文件清楚列出目前成果與下一步計劃，使後續開發人員（即使只有自己）能快速瞭解專案現狀。

第 3 週：JavaScript 語言基礎與前端互動

本週將從靜態頁面進一步走向動態互動，學習原生 JavaScript 語言的基礎知識。透過操作 DOM、處理事件，讓部落格頁面具備簡單的互動功能（如搜尋篩選）。每天的安排漸進式涵蓋 JS 的語法、DOM API、異步基礎等，並持續示範如何運用 AI 助理協助撰寫與除錯腳本。

- **Day 12：**
  理論概念： 認識 JavaScript 在前端中的地位，了解其與 HTML/CSS 的關係（結構、樣式、行為三者）。學習 JS 基本語法：變數宣告（使用 `let`、`const` [20]）、基本資料型別（Number、String、Boolean、Array、Object 等）以及運算子用法。撰寫簡單的運算表達式並輸出結果至 Console，以熟悉語法。
  學習資源： 建議參考 MDN 的 JavaScript 指南入門章節，逐步演練變數與型別的概念。Bilibili 上「3 小時前端入門教程」後半部分已涵蓋 JS 入門（變數、資料型別等） [20] [21]，可反覆觀看加深印象。亦可閱讀阮一峰的《JavaScript 教程（ES5版）》第一章內容，了解JS語言基本特性。
  AI 工具運用： 在瀏覽器 Console 中輸入簡單 JS 片段並利用 Cursor 的即時執行功能檢驗結果。如對某個語法點有疑問，詢問 Codex：「JavaScript 有哪些假值 (falsy)？」AI 將列出 `0`、`""`、`null`、`undefined` 等等並加以說明。嘗試讓 AI 出題測試自己，例如「請問以下程式輸出什麼？」透過解題來驗證自己對語法的理解。
  實作任務： 編寫一個獨立的 `hello.js`，內容是宣告數個變數存放不同型別的資料，然後透過 `console.log` 印出。在首頁 HTML 中，用 `<script>` 標籤引入此 JS 檔並開啟瀏覽器開發者工具，確認 Console 是否如預期輸出結果（例如字串拼接、算術運算結果）。額外地，練習在 JS 中操作先前 HTML 的元素：使用 `document.getElementById` 抓取頁首標題，將其文字內容透過 JS 更改（如加上「！Hello」字樣）。

驗收標準： 瀏覽器 Console 顯示 JS 檔案輸出的結果正確無誤，變數型別操作符合預期（例如相加運算得到正確數值）。透過 JS 修改 HTML 元素內容成功，刷新頁面後確實看到頁首標題文本發生改變。所有程式無語法錯誤，開發者工具Console中沒有出現 runtime error。

- **Day 13：**
  理論概念： 探索 JavaScript 操作網頁的核心：DOM（文件物件模型）。理解網頁載入後瀏覽器如何將 HTML 結構轉換成 DOM 樹，JavaScript 可以如何透過 `document` 物件存取和修改 DOM 節點。學習常用 DOM API：如 `document.querySelector` / `querySelectorAll` 選取元素，節點的 `innerText` / `innerHTML` 屬性修改內容，`createElement` 和 `appendChild` 動態新增節點等。
  學習資源： 參閱 MDN DOM 入門指南，了解 DOM 結構和常見的屬性、方法。觀看 YouTube 上的「JavaScript DOM 基礎教程」中文解說版，透過實例演示如改變樣式、新增列表項目等操作。閱讀《JavaScript DOM 編程藝術》相關章節，以較平易近人的方式認識DOM操作最佳實踐。
  AI 工具運用： 利用 Codex 快速查找 API 用法，例如詢問「如何使用 JS 選取所有帶有 class='post' 的元素？」AI 將回答使用 `document.querySelectorAll('.post')` [22]。當嘗試動態新增元素卻沒有效果時，可以讓 AI 幫忙除錯：提供相關程式碼請它分析可能問題（如忘記將新元素附加到 DOM）。
  實作任務： 為首頁新增一個「文章搜尋」輸入框和按鈕（可先直接在 HTML 加入簡單的 `<input>` 與 `<button>`）。編寫 `main.js` 腳本：在 DOMContentLoaded 事件後，選取該按鈕並綁定一個點擊事件監聽器。在事件處理函式中，取得輸入框的關鍵字值，遍歷目前頁面上的文章清單，如果文章標題不包含該關鍵字，就將對應的 `<article>` 元素隱藏（可操作元素的 `style.display` 或 `classList`）。嘗試運用 DOM 方法來達成篩選效果。
  驗收標準： 進入頁面後可在 Console 輸入 `document.readyState` 確認為 "complete"，主動觸發點擊搜尋按鈕，文章列表依輸入關鍵字正確篩選顯示/隱藏（例如輸入不存在的詞則全部文章隱藏）。實作的搜尋功能無頁面刷新，完全由 JS 即時更新 DOM。檢查 Console 確認無錯誤拋出，所有 DOM 操作的方法調用有效。

- **Day 14：**
  理論概念： 瞭解瀏覽器事件機制，學習如何監聽和處理用戶事件。掌握 `addEventListener` 的使用模式，理解事件對象 `event` 中常見屬性（例如 `event.target` 代表觸發事件的元素）。認識不同事件類型：如鍵盤事件、表單事件、頁面加載事件等，特別是表單提交事件的預設行為與如何透過 `event.preventDefault()` 來阻止。
  學習資源： 參考 MDN 事件基礎教程，閱讀關於事件捕獲與冒泡的說明以及範例。觀看 Bilibili 上的「深入理解 JavaScript 事件機制」講解，以動畫方式展示事件冒泡流程。查閱《JavaScript 高級程式設計》第 6 章關於 DOM 事件的內容，瞭解不同版本 IE/現代瀏覽器事件處理差異（雖然現代專案較少遇到，但可增進對歷史的理解）。
  AI 工具運用： 當不確定事件名稱或觸發條件時，詢問 AI 例如：「有哪些事件可以用在 `<input>` 元素上？」AI 將列出如 `input`、`change`、`focus` 等並解釋何時觸發。編寫事件處理函式遇到問題時，可以將代碼與錯誤訊息提供給 Codex，請它協助找出邏輯問題或遺漏的步驟。
  實作任務： 為文章搜尋功能補強使用者體驗：在輸入框上監聽鍵盤事件（`keyup` 或 `input`），使使用者不按按鈕也能即時篩選文章列表。此外，在點擊搜尋按鈕時，透過 `preventDefault()` 阻止可能的表單提交默認動作（若用了 `<form>` 包裹）。增加一個「清除搜尋」的按鈕，點擊時清空輸入框並顯示所有文章。透過這些練習，強化對事件處理的掌握。
  驗收標準： 不論使用者是按下 Enter 鍵、實時輸入，或點擊搜尋按鈕，文章列表都會正確響應並即時過濾。清除按鈕運作正常，點擊後輸入框變空且所有文章重新可見。開發者工具中無報錯，並能在 Sources 面板觀察到綁定的事件監聽器列表，驗證確實有註冊相應事件。

- **Day 15：**
  理論概念： 認識 JavaScript 中的函式和作用域。學習宣告函式的語法（函式宣告與函式表達式），理解參數與返回值的運作。介紹變數作用域（全域與區域）以及 `var` / `let` / `const` 差異所導致的作用域行為。了解閉包 (closure) 的概念及其簡單應用場景。

學習資源： 閱讀 MDN JavaScript Guide 中關於 Functions 的章節，掌握函式的基本用法和箭頭函式（ES6）的語法。參考阮一峰的 ES6 教程中關於 `let`、`const` 及區塊作用域的段落，理解為何應優先使用 `let`/`const` 而非傳統 `var` [20]。觀看 YouTube 上的「JavaScript Closure Explained」中文講解（如有），幫助理解閉包產生的原因。
AI 工具運用： 請 Codex 解釋一段函式相關的代碼。例如，給 AI 這段程式：

```
for(var i=1;i<=3;i++){ setTimeout(()=> console.log(i), 1000) }
```

詢問為何輸出不是 1,2,3 而是 4 次 4，讓 AI 說明作用域和閉包的影響（此例實際輸出會是四次數字4）。透過 AI 的講解，加深對作用域與閉包的理解 [23] [24]。
實作任務： 將前幾天的搜尋過濾邏輯重構包裝成一個函式 `filterPosts(keyword)`，以函式方式組織程式碼，並讓搜尋按鈕和鍵盤事件都呼叫此函式執行過濾。新增一個簡單的點擊計數功能作為練習：在每篇文章項目中加入一個「Like」按鈕，點擊後對應文章的按鈕文字會顯示被點擊的次數（可利用閉包儲存計數狀態或將計數儲存在 DOM 節點屬性上）。
驗收標準： 重構後的搜尋功能依然正常運作，代碼可讀性提升（透過函式封裝避免重複）。每個文章的 Like 按鈕點擊計數準確累加，並只影響各自文章（例如點了第一篇的 Like，不會影響第二篇）。檢查 Console 沒有未捕獲的 ReferenceError 或 TypeError，表示作用域使用正確；透過閱讀程式碼，可以清楚理解函式參數與返回值的流動。

- **Day 16：**
  理論概念： 介紹 JavaScript 的非同步機制，理解為何瀏覽器需要非同步處理（避免阻塞 UI）。學習使用 `setTimeout`、`setInterval` 進行簡單延遲與週期任務。初步認識 Promise 的概念和狀態（pending/fulfilled/rejected），以及 `async/await` 語法糖，為日後與伺服器通訊做準備。
  學習資源： 閱讀 MDN 關於 Event Loop (事件循環) 的文章，了解 JS 單執行緒透過任務佇列實現非同步的運作原理。參考 YouTube 上「JavaScript Promises/Future」的中文翻譯視頻或動畫，輔助理解 Promise 的執行流程。查閱《JavaScript 高級程式設計》中有關計時器與非同步的章節，全面掌握基礎。
  AI 工具運用： 請 Codex 幫忙舉例 Promise 的用法，例如：「能否提供一個 Promise 版本的 setTimeout 範例？」AI 可能會寫出一個回傳 Promise 的函式，在內部使用 `setTimeout` 後 resolve。亦可詢問 AI 關於 async/await 的錯誤處理方式，它應會提及 try/catch 方案。
  實作任務： 在部落格首頁加入一個模擬通知的功能：點擊「檢查新文章」按鈕後，2 秒後（使用 `setTimeout`）彈出一個 alert（或在頁面上顯示訊息）告知「目前沒有新文章」。此過程需非同步完成，不阻塞主執行緒。進一步，將這段非同步流程改寫成 Promise，例如創建一個函式 `checkNewPosts()` 回傳一個在2秒後resolve的 Promise，再在按鈕事件中透過 `.then` 或 `await` 呼叫它並展示結果。
  驗收標準： 點擊檢查新文章按鈕後，網頁並不會卡死，2 秒後才出現通知消息，期間仍可進行其他操作。改寫 Promise 後功能仍正常，並且 Console 無 UnhandledPromiseRejection 等錯誤。對比使用 callback（`setTimeout`）和 Promise/await 的代碼，功能等價且可讀性提升，說明學習者已掌握基本非同步處理的方法。

- **Day 17：**
  理論概念： 綜合本週 JS 學習內容，強調前端「資料—邏輯—顯示」的互動關係。理解如何在不刷新頁面的情況下更新UI，以及這為何是單頁應用 (SPA) 的基礎。初步了解將來可能用到的前端框架（如 React）的思想：組件化、狀態管理等，但目前重點仍放在原生JS的理解上。
  學習資源： 閱讀博客文章「在沒有框架時用原生 JS 打造 SPA」，看看純 JS 如何管理視圖和狀態。參考少量關於 React 的簡介資料，了解框架如何替我們處理底層 DOM 操作，從而體認掌握原生的重要性。回顧 freeCodeCamp 的前端課程地圖，確保自己已覆蓋其中 JS 基礎部分的要點。
  AI 工具運用： 讓 AI 協助總結：詢問 Codex「列出開發博客網站前端尚需改善的事項」。AI 可能會指出目前功能上的不足，例如缺少實際資料拉取、缺乏模組化結構等。我們可以根據 AI 的總結來計劃下週學

習（如引入 TypeScript、模組化開發等）。
實作任務： 進行代碼重構與整理。將本週撰寫的零散 JS 函式收納到一個模組化的檔案中，如果檔案過大可拆成多個（例如 `search.js`、`like.js`）。確保透過 `<script>` 正確引用，多個檔案之間的載入順序符合依賴關係。新增一些註解為關鍵程式碼添加說明，方便日後回顧。最後，將更新後的前端互動程式碼提交至 Git 分支並推送到遠端。
驗收標準： 重構後的程式組織架構清晰，在瀏覽器 Console 中無錯誤，所有互動功能（搜尋篩選、點擊喜歡、延遲通知等）維持正常。Git 倉庫中對應的 PR 獲得通過並合併，並在描述中清楚提及重構內容（可讓 Codex 幫忙產生 PR 描述）。這意味著學習者已能運用原生 JS 獨立實現一定程度的前端互動效果，為下一步學習更高級主題做好準備。

第 4 週：TypeScript 與前端模組化開發

本週引入靜態型別的 TypeScript (TS) 語言，提升程式碼的可維護性和可靠性。同時學習模組化開發方法，將前端代碼拆分為更易管理的模組。透過在專案中逐步加入 TS，我們將體驗型別檢查帶來的助益，並為日後整合前後端做準備。 Node.js 環境也會在此週稍作觸及，以利用 TS 在開發工具上的優勢。

- **Day 18：**
  理論概念： 認識 TypeScript 的定位：它是 JavaScript 的超集合，擁有靜態型別系統 [25]。了解為何型別有助於開發大型專案（如避免常見型別錯誤）。學習基本的 TS 語法：如何在變數、函式上標註型別，常見的基本型別（string, number, boolean, array<T>, any 等） [26]。
  學習資源： 閱讀 TypeScript 官方中文文件入門教程 [27]，從「基礎類型」章節開始練習型別註記。參考阮一峰老師的 TypeScript 教程 [28]，其中對 TS 強大型別系統有完整且易讀的介紹。觀看 Bilibili 上的「TypeScript 十分鐘入門」視頻，快速瀏覽 TS 語法重點。
  AI 工具運用： 在 Cursor 中將昨天的部分 JS 檔案改名為 `.ts` 檔，觀察 IDE 給出的型別錯誤提示。若有，不妨讓 AI 協助修正。例如當 TS 報錯需要某變數型別時，可詢問 Codex 該如何宣告。也可以請 AI 將一段 JS 代碼自動轉換為 TS（它可能會猜測所需的型別定義 [26]）。
  實作任務： 初始化一個 TypeScript 專案環境：在專案根目錄執行 `npm init -y` 初始化 Node 專案，接著 `npm install typescript --save-dev` 安裝 TS 編譯器。使用 `npx tsc --init` 生成一份 `tsconfig.json` 並確認設定（例如 `target` 設為 `ES6`，`module` 設為 `ES6` 以便瀏覽器支援）。然後將先前的 `main.js` 重構為 `main.ts`：為主要函式和變數添加適當的型別註記（如搜尋關鍵字為 string，文章列表為 HTMLElement 陣列等）。嘗試編譯（`npx tsc`）並修正任何出現的型別錯誤。
  驗收標準： 成功建立 TS 編譯環境並生成對應的 JavaScript 檔案（預設輸出在同目錄或 `dist` 目錄，看 tsconfig 配置）。`tsc` 編譯時無報錯，或能根據錯誤提示迅速修正型別問題後重新編譯通過。產生的 JS 在瀏覽器中運行正常，功能與轉換前一致。TypeScript 型別檢查已經開始發揮作用，例如明確標註的型別可以在 VSCode/Cursor 中得到自動提示和錯誤預警 [25]。

- **Day 19：**
  理論概念： 學習進階的 TypeScript 型別系統特性：接口（Interface）和類別（Class）的用法。了解如何用 Interface 定義物件的型別結構，方便對資料物件建模。簡介 TS 中的類別語法（與 ES6 Class 類似但有型別限制），以及存取修飾詞（public/private/protected）對成員可見性的控制。
  學習資源： 參閱 TypeScript 官方文件中「Interfaces」和「Classes」章節，學習定義和實作接口 [29]。閱讀博客文章「用 TypeScript 重構 JavaScript 專案的經驗分享」，看看在 JS 轉 TS 過程中如何運用接口來替代冗長的物件型別註記。觀看 YouTube 上的「TypeScript Classes vs Interfaces」中文講解，理解何時應用接口、何時使用類別。
  AI 工具運用： 讓 AI 幫忙將前端代碼中的某些物件定義提取為 interface。例如，網站的文章物件可能包含 `title`, `summary`, `likes` 等屬性，請 Codex 協助定義一個 TypeScript interface 來描述這種結構。或者提供一個 class 草稿，請 AI 協助填充建構式及方法的型別。
  實作任務： 定義介面來描述部落格的資料結構：例如建立 `Post` 介面定義文章物件型別（包含標題、內容、作者、喜歡數等），建立 `User` 介面定義使用者物件型別（包含使用者名稱、註冊日期等）。

在現有前端程式中，對應的資料結構套用這些介面進行型別註記。若有時間，可嘗試將 Like 計數功能改寫為一個 `LikeButton` 類別，內含一個 `count` 屬性和一個 `increment()` 方法，每次呼叫將計數加一，並更新按鈕文字。將頁面上的每個按鈕實例化為這個類別，並在點擊時調用其方法。
驗收標準： TypeScript 編譯通過，新增的介面和類別定義無語法錯誤，且被正確運用在對應程式碼處。介面的應用使得函式的參數或物件更具意義（例如函式簽名從 `function renderPost(post: any)` 改為 `function renderPost(post: Post)`）。LikeButton 類別實例能正常工作，點擊按鈕後計數器遞增如前。藉由 `tsc` 的型別檢查，可防止對文章或使用者物件調用不存在的屬性，提升了代碼穩定性 [25]。

- **Day 20：**
理論概念： 介紹前端模組化的概念與好處：避免全域命名空間污染、按需載入等。學習 ES6 模組語法：`export` 和 `import` 的用法，如何將程式碼拆分成多個模組檔案。了解在瀏覽器環境使用 ES6 模組需要在 script 標籤添加 `type="module"`，以及可能需要的打包工具（如 Webpack、Vite，但此處簡要提及即可）。
學習資源： 閱讀 MDN 關於 JavaScript Modules 的教學，瞭解靜態模組與動態匯入(`import()`)的基礎。參考 TypeScript 手冊中關於模組解析與匯入策略的說明，以確保 TS 環境下模組能正確編譯。觀看 Bilibili 上的「前端模組化演進史」簡報或動畫，理解從過去無模組到 AMD/CJS，再到現今 ES Modules 的變遷脈絡，增加知識廣度。
AI 工具運用： 如果對模組語法不熟悉，可以讓 AI 提供範例，例如：「如何將一個函式導出並在另一個檔案匯入？」AI 會給出 `export function foo() {...}` 以及對應的 `import { foo } from './file.js';` 範例 [30]。若打包配置遇到困難，也可請教 AI 簡單的 Vite/Webpack 配置方法，但在本計畫重點是理解原理，不深陷工具設定。
實作任務： 將前端代碼進行模組劃分：例如新建 `utils.ts` 存放共用的輔助函式，`dom.ts` 專門處理 DOM 操作相關的函式。使用 `export` 將這些函式導出，在 `main.ts` 中以 `import` 匯入並使用。調整 HTML 引用 JS 的方式：使用 `<script type="module" src="main.js"></script>` 確保瀏覽器將之視為模組執行。編譯 TS，並在瀏覽器中測試功能是否依舊正常。
驗收標準： 模組拆分後網站功能無回歸錯誤，各按鈕與搜尋互動仍正常。檢查編譯出的 JS，驗證各個模組檔案都被正確引用（或透過打包工具匯總，如果使用打包工具的話）。在瀏覽器開發者工具的 Network 頁籤，可見多個 JS 模組檔案被載入（若未經打包）。程式架構更為清晰，學習者能夠說明每個模組的職責範圍，展現模組化開發的思維。

- **Day 21：**
理論概念： 熟悉 Node.js 環境下執行 TS/JS 的方式，為將來後端開發做鋪墊。學習使用 Node.js 執行簡單腳本，了解 `npm scripts` 的配置，可利用其來執行編譯、啟動開發伺服器等。認識 .npm 套件管理的意義，了解前端構建工具如 Vite 的作用，但本日重點在於 Node 環境基本操作。
學習資源： 參考 Node.js 官方文件的「Command Line Options」部分，學習如何直接執行 TS（例如需要先安裝 ts-node）或編譯後執行。閱讀一篇介紹 Vite 的中文博客文章，了解我們很快將使用它來整合前後端開發（next-fastapi-prisma 模板中可能用到） [31] [32]。瀏覽 GitHub 上 nextjs-fastapi 範本的 README [33] [34]，感受未來我們專案的最終形態。
AI 工具運用： 在 VSCode/Cursor 的終端中使用 AI 協助執行命令。例如可以嘗試對 AI 說「執行 npm run build 並說明結果」，AI 會先產生執行的預測，再解釋每步意義。如果出現錯誤日誌難以理解，也可直接貼給 Codex 要求分析原因。
實作任務： 在 `package.json` 中新增兩個腳本：`"build": "tsc"`（編譯 TypeScript）和 `"start": "vite"`（若已安裝 Vite；如果未使用打包工具，則此 start 可以暫留或運行輔助腳本）。透過 `npm run build` 測試編譯指令是否可順利執行並輸出結果。若時間允許，可嘗試安裝 Vite 並進行簡單配置，使之能夠提供靜態檔案與 Hot Reload 功能。在終端執行 `npm run start`，確認能啟動本地開發伺服器並透過瀏覽器訪問我們的前端頁面。
驗收標準： `npm run build` 成功執行並產生對應的 JS 檔，所有 TypeScript 檢查通過。`npm run start` 能啟動本地伺服器（假如使用 Vite，預設應在 http://localhost:5173），並且可以正常訪問網

站內容。透過這些腳本，開發與部署流程變得一致且簡化。此時，前端部分已相對完善並準備好與後端進行串接，學習者對 Node.js 生態與前端構建工具也有了初步認識。

第 5 週：引入 React 前端框架（部落格前端升級）

在充分體驗原生開發後，本週開始使用 React 框架重構前端。React 將大幅提升我們構建複雜互動界面的效率。我們會從基礎的 React 組件、JSX 語法入手，逐步將之前的靜態頁面轉換為 React 應用，也為之後和後端 API 串接奠定基礎。學習者將理解如何在工程中使用現代前端框架，並體驗與 AI 合作進行框架開發的流程。

- **Day 22：**
  理論概念： 認識 React 的核心理念：UI = f(state)，即使用組件的狀態決定UI輸出。學習 JSX 語法（在 JavaScript 中撰寫類似 HTML 的標記）以及 Babel 轉譯它的原理。了解功能型組件（Function Component）的基本寫法，以及最簡單的 Hook：`useState`（管理狀態）的用法。
  學習資源： 閱讀 React 官方中文文檔[35]中的「快速入門」章節，了解如何創建與嵌套組件、傳遞 props 等[36]。觀看 Bilibili 上翻譯的 React 教程（如 simviso-cs 翻譯的2019最佳React教程系列）以快速掌握 React 基礎[37]。可參考阮一峰的 React 入門文章，從簡單範例了解 JSX 與組件渲染。
  AI 工具運用： 使用 Cursor 建立 React 開發環境：可以讓 AI 指示使用 Vite 初始化React TypeScript模板，例如執行 `npm create vite@latest my-blog -- --template react-ts`。若過程中有不明步驟，隨時詢問 AI。寫組件時，利用 AI 自動完成：輸入 `function PostList(){` 後讓 AI 補全基本結構。AI 也可協助將先前 HTML 轉為 JSX（可直接貼給 Codex，請它轉換為 React 組件的樣式）。
  實作任務： 初始化一個新的 React + TypeScript 專案（使用 Vite 或 Create React App）。確保專案結構生成後可以 `npm run dev` 跑起開發伺服器並顯示預設頁面。建立基本的 React 組件架構：創建 `App.tsx` 作為根組件，然後將之前靜態首頁的結構拆分為幾個子組件，例如 `Header`（含導航）、`PostList`（內含多個 Post 項目組件）、`Footer` 等。先不引入狀態，僅用靜態資料（可直接寫在陣列中）渲染文章列表。驗證頁面能正常顯示這些組件輸出。
  驗收標準： React 開發環境順利運行[38]；新架構下的部落格首頁能透過 React 正確渲染，畫面內容與之前的靜態版相仿。打開 React DevTools，可以看到組件樹結構與各組件對應的DOM元素。開發過程中沒有出現嚴重 TS 錯誤，組件都已正確定義 props 型別（若有使用）。這標誌著我們的前端正式從靜態頁面升級為由 React 組件驅動。

- **Day 23：**
  理論概念： 學習在 React 中管理狀態和處理事件。複習 `useState` Hook 的使用，了解設定狀態值會觸發組件重新渲染的機制。學習如何在 JSX 中綁定事件處理器，例如按鈕的 onClick 等。比較 React 合成事件與原生 DOM 事件的區別。
  學習資源： 閱讀 React 官方文件「State 與 Lifecycle」部分，理解使用 Hook 前的類別組件 setState 概念，可幫助理解 Hook 的由來。參閱 React 文件中的 Handling Events 章節，學習事件在 React 中的寫法。觀看 YouTube 上React Hook 講解中文視頻，理解 Hook 解決了什麼問題（狀態邏輯複用等）。
  AI 工具運用： 在 Cursor 中撰寫 `useState` 時讓 AI 自動補全初始值和型別。當不確定事件對象類型時，可詢問 Codex 例如「React onClick 事件處理函式的 TypeScript 型別是什麼？」AI 將給出 `MouseEvent<HTMLButtonElement>` 等提示。讓 AI 幫助優化：如提供目前使用狀態管理的一段代碼，請 AI 檢查是否有不必要的狀態或可以合併的地方。
  實作任務： 為 React 版的部落格加入互動：實現之前的「Like」按鈕功能。可以在 Post 組件中使用 `useState` 定義一個 likes 狀態初始為0，渲染一個按鈕顯示 Likes 數，點擊按鈕時呼叫設定函式遞增 likes。確保每個 Post 組件各自維持獨立的 likes 狀態。再實作搜尋篩選功能：在主 App 組件中用一個 state 保存目前的搜尋關鍵字，為搜尋欄位的 onChange 綁定事件更新此 state，並將關鍵字作為 props 傳給 PostList 組件。在 PostList 組件中據此過濾要渲染的 Post 列表（例如透過 Array.filter）。
  驗收標準： React 應用成功實現兩個狀態驅動的互動功能：點擊 Like 按鈕只影響對應文章的計數，切換搜尋關鍵字即時影響文章列表顯示。透過 React DevTools 檢視各組件 state 值隨操作正確改變。所有事件處理在 React 中以推薦方式（例如不直接操作 DOM，而是改變 state）完成，頁面沒有手動 DOM 操

作卻能跟隨狀態自動更新。TypeScript 嚴格模式下無報錯，表示我們正確定義了 props 和 state 的型別。

- **Day 24：**
  理論概念： 了解 React 中的 props 父子溝通機制，強調單向資料流概念。學習如何將函式以 props 傳遞，讓子組件呼叫以觸發父組件的狀態改變（例如子組件內按鈕點擊通知父組件）。討論何時需要將狀態提升（Lifting State Up）：當多個子組件需要共享某狀態時，將狀態提升到它們共同的父組件。
  學習資源： 參閱 React 官方文件「Lifting State Up」章節，學習經典的攝氏華氏溫度轉換範例，理解如何讓兩個輸入框共享同步一個狀態。閱讀一篇中文博客「React 單向數據流與狀態提升」，理解在 React 中數據流動的方向有助組件結構設計。也可查看 Bilibili 教學影片中對 props 傳值的演示，加深印象。
  AI 工具運用： 設計父子通信時，可讓 AI 協助驗證思路：例如告訴 Codex 「我想在子組件點擊按鈕時更改父組件的狀態」，看 AI 是否給出傳遞 callback 作為 props 的方案。也可以請 AI 優化我們的組件結構，比如當某 state 其實可由父管理時，它會建議我們提升狀態並通過 props 傳下。
  實作任務： 將搜尋關鍵字的狀態提升：目前我們可能在 App 組件中處理了搜尋，繼續確保這種架構——App 持有 keyword 狀態並將其與 setter 作為 props 傳給搜尋欄子組件（如我們創建一個 SearchBar 組件專職渲染輸入框和按鈕）。讓 SearchBar 子組件透過 props.onSearchChange 通知父組件 App 更新關鍵字。測試確保功能仍正常。額外練習：實作一個可以全局控制顯示模式的狀態（例如深色/淺色主題切換），將此狀態提升至最頂層 App 並透過 Context 或 props 下傳，使按鈕點擊可以切換主題（簡單處理可在 body 套用不同 className）。
  驗收標準： 搜尋功能在狀態提升後依然運作良好，結構更加清晰：SearchBar 不再擁有自己獨立的關鍵字狀態，而是透過 props 與 App 同步。檢查 App 組件確認其狀態確實被子組件事件觸發更新。主題切換按鈕能影響整體外觀（例如背景或文字顏色切換），並且運用了 React 的單向數據流或 Context 進行狀態共享。專案的組件拆分更加合理，每個組件職責單一明確。

- **Day 25：**
  理論概念： 初步了解 React Router 等路由方案的概念，實現單頁應用中的多頁面切換。本日先聚焦概念：什麼是 client-side routing，為何 SPA 需要路由庫。了解 React Router 提供的 `<BrowserRouter>`, `<Routes>`, `<Route>` 基本用法，以及 Link 元件實現無刷新導航。
  學習資源： 閱讀 React Router 官方中文文件的快速開始部分，了解如何在 React 中設定路由表。參閱一篇中文教程「用React Router建立單頁多視圖應用」，其中圖解了BrowserRouter的工作方式。若時間允許，可觀看極客時間或YouTube上對SPA路由原理的講解，加深理解（例如 HashRouter vs BrowserRouter 的差異）。
  AI 工具運用： 讓 AI 幫助設定基本路由：諮詢 Codex「如何在React中建立兩個頁面：首頁和文章詳情頁？」AI 會提供使用 React Router 的步驟與程式碼範例。將程式碼段貼入 Cursor，可快速安裝套件並生成樣板。AI 也能解釋路由元件屬性，例如 `path="..."` 與 `element={<Component/>}` 的意思。
  實作任務： 在專案中引入 React Router（`npm install react-router-dom`）。於應用入口（通常是 `main.tsx` 或 `App.tsx`）設置路由：建立 `<Routes>`，包含至少兩條路由：`"/"` 對應部落格首頁組件，`"/posts/:id"` 對應文章詳情頁組件（可暫時創建一個簡單的 PostDetail 組件，未來再充實其內容）。更新文章列表，使點擊文章標題時使用 `<Link>` 導向該文章的詳情路由。確保 Router 正常工作：在開發伺服器中測試點擊連結切換頁面，URL 有更新且顯示對應組件。
  驗收標準： React Router 成功啟用，切換路由時無頁面整體重載而是即時切換組件 39 。首頁文章列表中的連結可以正確導航到文章詳情頁，並且瀏覽器返回按鈕也能返回首頁。暫時的 PostDetail 組件可先顯示文章ID等占位訊息，以確認路由參數的取得（透過 `useParams`）。整體應用仍能正常運行，沒有路由配置錯誤導致的白頁或報錯。

- **Day 26：**
  理論概念： 優化 React 應用性能與開發體驗：介紹 React 開發中常用的工具與技巧，如開發環境下的 Hot Module Replacement（HMR）讓修改後界面即時更新，Redux 等狀態管理庫的簡介（本計畫可能

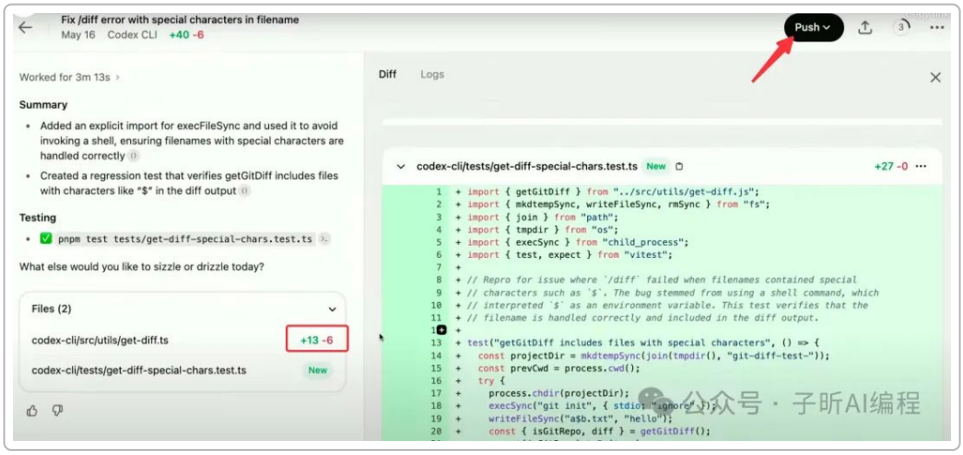
不深入Redux，但要知道其解決的痛點）。提及 React 中的性能優化 Hook：`React.memo`、`useCallback`、`useMemo` 簡單原理（記憶化避免不必要渲染）。
學習資源： 查閱 Vite 官方文檔確保已充分利用其 HMR 特性（通常開箱即有）。閱讀 React 官網上的「Optimizing Performance」指南，瞭解生產環境建置、代碼拆分等手段。找一篇中文文章「React Hooks 性能優化實踐」看看在適當時機如何使用 memo 等技術，但也強調過早優化的危害。
AI 工具運用： 讓 AI 協助檢視我們的 React 組件性能。例如詢問 Codex：「在我的PostList組件中，每次輸入文字是否重新渲染了所有Post？有無優化空間？」AI 若分析出不必要的重複渲染，可能建議使用 React.memo 包裝 Post 組件，使其在 props 不變時不重渲染。嘗試在 AI 指導下實施一兩處 memoization，觀察效果。
實作任務： 對應 AI 建議，對某些組件實施 Memo 或 Hook 優化：例如將 PostList 下的單篇 Post 組件用 `React.memo` 包裝，減少整個列表重繪。或者在 SearchBar 中使用 `useCallback` 儲存搜尋框 onChange 的處理函式，避免每次 App render 都傳遞新函式導致 SearchBar 重渲染。使用 React Developer Tools 的 Profiler（若有時間學習）來比較優化前後的渲染次數。
驗收標準： 引入的優化沒有破壞原有功能，應用行為與之前一致。透過簡單的 console 日誌或 React DevTools Profiler，可以驗證例如輸入搜尋關鍵字時未改變的 Post 組件不再重新渲染（Console log 可在 Post 組件的函式體內放置以觀察）。雖然本專案規模較小性能不是大問題，但學習者已掌握基本的優化理念，知道在需要時如何應用這些手段。

- **Day 27：**
理論概念： 總結 React 前端的成果與不足，為下週開始的後端開發做準備。討論目前前端仍使用假資料，下一步需要透過 API 從後端獲取真實資料。複習 AJAX/Fetch 概念，了解如何在 React 中進行資料請求以及生命周期（或 useEffect）的使用。提前構思前後端介面定義，比如部落格文章的 GET/POST API 會是什麼路徑、傳回何種 JSON 資料格式。
學習資源： 閱讀 MDN Fetch API 文件，熟悉使用 fetch 發送 HTTP 請求與處理 Response 的基本流程。參考 FastAPI 官方文件中關於建立簡單 API 的範例，對照思考前端需要如何呼叫。查看 GitHub 上 nextjs-fastapi 樣板專案的 API 介面（如有文件）或代碼，瞭解我們或許可以直接採用那些設計 [31] [40] 。
AI 工具運用： 在與 AI 的對話中模擬前後端合作：可以讓 Codex 充當後端，列出可能的 API 端點（如 GET `/api/posts` 返回所有文章列表）。再讓 AI 生成對應的 fetch 調用代碼，這樣前端開發者能預先寫好調用邏輯和型別定義。這種對話有助於在正式寫後端前理清思路。
實作任務： 撰寫一個模擬用的資料抓取函式。在前端創建一個 `api.ts` 模組，內含函式例如 `fetchPosts()`，目前不真的從後端抓取，而是返回 Promise.resolve(假資料)。使用 `setTimeout` 模擬網路延遲。然後在 React 組件（例如 App 或 PostList）中使用 `useEffect` 在組件載入時呼叫這個 fetchPosts，將取得的資料放入 state，據此渲染文章列表。這基本流程類似將來串接真實 API，只是目前用假資料置換。
驗收標準： 前端代碼結構中已考慮到資料請求的非同步性：例如初始時文章列表為空，加載完畢後狀態更新出現內容，這過程沒有阻塞 UI。可以在畫面上加一個簡單的 “Loading…” 提示，在資料尚未返回時顯示，驗證 useEffect 和狀態更新機制有效。透過 Network 面板確認目前沒有真正的網路請求（因為我們用假資料），下一週我們將把這部分替換為真實後端呼叫。

- **Day 28：**
理論概念： 了解如何將現有前後端整合，搭配 Git 的工作流程。探討 Monorepo（前後端同倉）的優劣，以及如何設定不同專案的執行（我們可考慮前後端各自一個資料夾）。理解 Pull Request 在跨技術棧開發時的管理方式，例如前端與後端可以分開PR或一起PR取決於功能完整性。
學習資源： 閱讀 Netify 或 Vercel 官網關於 Monorepo 部署的最佳實踐文章，了解部署平台如何識別前後端分別構建。參閱前人博客「如何在單一倉庫中管理React前端與FastAPI後端」分享的經驗，吸取其關於專案結構和自動化流程的建議。
AI 工具運用： 諮詢 AI 關於倉庫組織：例如「建議一個包含 React 和 FastAPI 的專案結構」。AI 或會建議將 frontend/ 與 backend/ 兩個目錄並列 [34] [41] ，各自有自己的 README 等。我們可依此調整目前專案，把之前的內容移動到 frontend 資料夾，並初始化 backend 資料夾以待後端代碼填入。

實作任務： 重組專案目錄：新建 `frontend/` 將整個 React 專案搬入其中，確保 `package.json` 等也隨之移動且能正常執行。新建 `backend/` 資料夾，創建一個簡單的 FastAPI 啟動檔（如 `main.py`）先打印「Hello FastAPI」確認環境。同時在倉庫根目錄添加一個 README，說明專案目的和子資料夾結構。調整 Git 設定（如需要，可設定不同資料夾有不同 .gitignore）。將此重組提交至 Git，開啟一個 PR，檢查前端部署（若有CI設定）是否仍然成功。
驗收標準： 前端開發伺服器在新路徑下仍可啟動（`cd frontend && npm run dev` 正常）。後端目前只是佔位，但 `uvicorn` 啟動 backend/main.py 時能運行無誤（雖然沒功能）。Git 倉庫反映出新的結構，CI/CD（如Vercel針對 frontend 資料夾部署）配置相應更新。PR 描述中清楚記錄了此次重大調整。至此，前後端環境已就緒，後續可以專注後端開發並逐步串接雙方。

第 6 週：FastAPI 後端開發入門



圖注：OpenAI Codex 執行任務後會生成程式碼差異（Diff）供開發者審查，確認無誤後可一鍵 Push 到 GitHub 完成自動提交 [42]。Codex 能自動編寫並測試後端 API 碼，若測試未通過會自行修正直到通過 [2]。本週開始後端開發，以 Python + FastAPI 構建部落格的API伺服器。我們將從基礎的API路由、請求/回應模型著手，每天實作不同的後端功能模組。課程涵蓋資料模型定義、與前端串接，以及善用AI加速API開發、除錯的技巧。

- **Day 29：**
  理論概念： 認識 FastAPI 框架的特點：現代、快速 [43]、基於 Python 型別提示自動生成文件等。學習建立一個最小的 FastAPI 應用：包含一個路由（endpoint），例如 GET `/` 返回歡迎訊息。瞭解路由函式需要用 async 定義以充分利用異步特性，以及使用 Python 型別宣告請求參數/回傳模型的基本方式。
  學習資源： FastAPI 官方中文用戶指南 [44] 從 Hello World 教學開始，一步步展示各項特性，可據此操作 [45]。參閱菜鳥教程的 FastAPI 教程 [22]，快速抓取建立 API 的重點步驟。觀看 Bilibili 上「FastAPI 快速入門」視頻（苑昊老師系列）第一節 [46] [47]，跟隨視頻在本地構建第一個 API。
  AI 工具運用： 在 Cursor 開啟 backend 資料夾，新建 main.py。利用 Codex 自動生成 FastAPI 啟動模板：可提示「撰寫一個FastAPI應用監聽 `/` 路徑返回 JSON: {'message': 'Hello'}」。AI 應會生成包含 `FastAPI()` 實例化和 `@app.get("/")` 路由的代碼。我們可以直接採用並稍做調整。也請 AI 說明 uvicorn 啟動的命令，以便運行測試（例如 `uvicorn main:app --reload`）。
  實作任務： 在 backend/main.py 中建立基本的 FastAPI 應用。定義 GET `/` 路由，回傳一段歡迎訊息 JSON，如 `{"message": "Blog API up and running"}`。使用 Uvicorn 開發伺服器啟動應用（`uvicorn main:app --reload`），檢查終端無誤後，瀏覽器訪問 http://localhost:8000/ 出現正確訊息。接著訪問 http://localhost:8000/docs，確認自動生成的互動式 API 文檔（Swagger UI）能顯示該 GET 路由 [48]。
  驗收標準： FastAPI 應用成功運行，基礎路由返回預期結果（可嘗試使用 `curl` 測試 GET `/` 返回 200 並含正確 JSON）。Swagger UI [48] 頁面正常打開且列出目前唯一的 API 端點。這表示後端開發環境搭建完畢，框架運作如預期。將此基礎結構提交至 Git（backend 資料夾可能需要一個簡單的 **init**.py 保證其為模組），開始版本控制後端代碼。

- **Day 30：**
  理論概念： 設計並實作部落格的資料模型和對應的API骨架。確定後端需要的核心模型：使用者(User)、文章 (Post)、分類 (Category) 等。學習使用 Pydantic 模型定義請求和回應的結構 [48]（FastAPI 利用 Pydantic 自動進行資料驗證）。理解如何在 FastAPI 路由中使用這些模型作為參數和回傳值，以自動生成文件和驗證輸入。
  學習資源： FastAPI 官方文件「模型」章節講解了如何繼承 BaseModel 創建 Pydantic 模型並用於請求/回應 [48]。參考前面查找的部落格資料庫設計文章 [49]，對照思考我們的模型屬性。閱讀 GitHub 上 FastAPI Prisma 模板 [50] 或 Next.js FastAPI 模板文件，瞭解其資料模型如何組織（例如 User 包含 email 等欄位）。
  AI 工具運用： 讓 Codex 輔助生成 Pydantic 模型。例如要求「定義一個 Pydantic 模型 Post，有 title:str, content:str, created_at:datetime 屬性」，AI 將生成相應類別定義。我們也可請 AI 產生多個模型並考慮關聯。Codex 或許會建議一些實用欄位例如 id: int，我們可以酌情採納 [51] [52]。
  實作任務： 在 backend/main.py 中，定義 Pydantic 模型：`PostCreate`（發表文章請求用，包含標題、內容、作者等必要欄位），`Post`（回應用，繼承 PostCreate 並加上 id、created_at 等只讀欄位）。定義類似的 User 模型（UserCreate, User）和 Category 模型。如未使用資料庫，暫時可用 Python 結構暫存資料：例如在模組層宣告 `posts: List[Post] = []` 充當簡易DB。撰寫以下API雛形：GET `/posts` 回傳所有文章列表，GET `/posts/{id}` 回傳單篇文章（根據 id 查找），POST `/posts` 接收 PostCreate 建立新文章並回傳建立後的 Post 資料。此時僅模擬資料操作（對 posts 陣列 append 等），後面再接資料庫。
  驗收標準： 啟動應用，訪問 `/docs` 可看到我們新增的三個 posts API 端點及其請求/回應模型結構 [48]。透過 Swagger UI 測試：先 POST 新文章，應得到 id=1 等回應；再 GET `/posts` 能取回剛新增的文章列表；GET `/posts/1` 則取單一文章。暫時使用記憶體資料，重啟服務資料會丟失，這點允許。這表明我們已成功建立起基本的文章 API，並對 FastAPI 模型與路由配置有了初步實踐。

- **Day 31：**
  理論概念： 完善更多API端點：實作分類 (Category) 和使用者 (User) 相關的基礎API。理解RESTful風格對這些資源的操作：如 GET `/categories` 列出分類、POST `/users` 創建使用者等。考慮資料之間的關聯：例如文章屬於某一分類，如何設計API讓前端獲取文章時知道其分類名稱，或過濾某分類的文章。這涉及查詢參數的使用以及關聯查詢。
  學習資源： FastAPI 官方文件「路由（Operation）」部分講解了路徑參數、查詢參數的宣告方法 [48]。閱讀 Stack Overflow 或 FastAPI GitHub 討論上一些「如何處理外鍵關係」的問答，瞭解是否直接在Pydantic模型中嵌套其他模型或使用ID即可（Pydantic 允許用另一模型作字段型別，會自動解析）。查詢上一篇我們找到的CSDN部落格資料表設計 [52] [53]，確認Category應有的欄位如名稱、描述等。
  AI 工具運用： 讓 AI 幫忙生成 Category 相關代碼：如要求「寫一組FastAPI API讓我可以新增分類、取得所有分類清單」。Codex 將產出模型定義和對應路由。也詢問 AI 關於在 Post 回應模型中包含 Category 名稱的實踐方法，AI 可能建議在 Post Pydantic模型中嵌套 Category 模型或使用 category_id 取代。這些建議可供權衡取捨。
  實作任務： 定義 CategoryCreate 和 Category 模型，並新增 GET `/categories` 和 POST `/categories` 端點，類似之前文章的實作方式。接著修改 Post 模型，在其中增加 category_id 欄位以表示文章分類（為簡化，可不嵌套整個 Category 模型，只存ID）。實作 GET `/posts?category_id=x` 的過濾：在取得文章列表的函式中，檢查如果 query參數category_id有值，就過濾僅回傳該分類的文章。也實作使用者相關基礎：定義 UserCreate, User 模型，提供 POST `/users` 端點創建使用者，GET `/users/{id}` 取得使用者資訊（例如名稱、註冊時間等）。
  驗收標準： 新增的分類與使用者API在 `/docs` 文件中清晰展現，結構正確無誤。透過 Swagger 實際操作：新增一個分類（如 "Tech"），取得分類列表能看到它；新增一個使用者（如 username: Tom），取得其ID並可透過 GET 查詢到詳細。再建立一篇文章時在請求體內指定 category_id 與 user_id，資料成功寫入；之後調用 GET `/posts?category_id=...` 僅返回對應分類文章。所有回傳 JSON 結構符合定義，fastAPI 自動校驗功能可在傳錯資料時給出 422 錯誤。此步完成後，我們的後端API大致框架已備齊。

- **Day 32：**
  理論概念： 開始引入資料庫永續化資料。討論選擇資料庫類型及ORM工具：本專案可採用 SQLite 或 PostgreSQL 作為存儲，為便於開發先用 SQLite 文件數據庫。介紹 SQLAlchemy（Python主流ORM）或 Prisma Client Python 的用法，考慮哪個更適合本計畫。由於我們目標全端實務，Prisma 可提供與前端類似的schema定義體驗 54 ，但SQLAlchemy更成熟且中文資源多。此處決定用哪個皆可，下面以SQLAlchemy為例。
  學習資源： FastAPI 官方文件有專章介紹如何與SQLAlchemy整合，包括創建 SessionLocal、定義 ORM 模型類等 48 。參考 Prisma Client Python 文件 55 如果選用 Prisma，學習如何在 Python 中調用。閱讀一篇中文博客「FastAPI 連接資料庫（以SQLite為例）」掌握實際操作。
  AI 工具運用： 諮詢 Codex 關於使用 SQLAlchemy vs Prisma 的意見，AI 可能會給出各自優缺點如 54 所述：Prisma 可前後端共享 schema，SQLAlchemy 原生 Python 兼容佳等。我們決定路線後，請 AI 協助生成對應代碼：例如「使用SQLAlchemy創建SQLite連接並定義Post資料表ORM模型」。AI 將產生 Base = declarative_base(), engine = create_engine('sqlite:///...') 等樣板，我們可調整後使用。
  實作任務： 安裝必要套件（SQLAlchemy 或 prisma）。如果用 SQLAlchemy：定義 ORM 模型類，如 PostORM, UserORM, CategoryORM，對應數據表欄位。配置資料庫連線（Sqlite 本地檔案），在 FastAPI 啟動時創建資料表（透過 SQLAlchemy 的 create_all）。更新之前的路由實作，使其不再使用內存清單，而是通過 ORM 查詢資料。例如 GET `/posts` 使用 Session 查詢 PostORM 表；POST `/posts` 建立 ORM 實例後 commit 保存。確保異步環境下使用適當方法（注意SQLAlchemy預設同步，可使用 `run_in_threadpool` 或改用 async ORM）。
  驗收標準： 重新啟動後端應用，資料庫檔案成功建立，表結構包含我們定義的欄位 51 52 。透過以前相同的API調用流程測試：新增分類/使用者/文章，並查詢列表，應得到與之前相同的結果，但這次資料已存入資料庫。重啟服務器後先前創建的資料不丟失（可透過再次GET驗證）。此時我們的後端已真正實現資料持久化，為多用戶提供服務打下基礎。

- **Day 33：**
  理論概念： 實現使用者註冊和登入認證功能。介紹 JSON Web Token (JWT) 作為 stateless 認證方案 56 ：伺服器透過密鑰簽發JWT給驗證通過的用戶，日後請求帶該token即可證明身份。學習 FastAPI 提供的 OAuth2PasswordBearer、OAuth2PasswordRequestForm 等實用工具簡化驗證流程。理解哈希密碼的重要性，不以明文儲存密碼。
  學習資源： FastAPI 官方指南「安全驗證」部分詳細範例了 OAuth2 密碼模式與 JWT 的整合，可作為模板來實現 48 。閱讀 CSDN 上JWT/OAuth中文解釋 57 58 ，理解兩者區別和聯繫，知JWT屬於一種token機制而OAuth2是授權框架。也可參閱 Authlib 或 PyJWT 文檔瞭解簽發JWT的步驟。
  AI 工具運用： 讓 AI 輔助編寫登入流程：諮詢 Codex 「如何在FastAPI中實現JWT驗證？」AI 可能會產生一個教學風格的代碼，包括生成 access token 的函式和一個 login 路由，參考值極高。我們可調整 Secret Key等配置並採用。也請 AI 審視我們的註冊流程是否安全（如是否有檢查重複使用者、密碼強度），獲取改善建議。
  實作任務： 安裝 `passlib[bcrypt]` 用於密碼哈希。新增 POST `/users/signup` 路由：接受使用者註冊資料（UserCreate），哈希處理密碼後存入資料庫。新增 POST `/users/login` 路由：驗證用戶提供的username/password，正確則透過 `jwt.encode` 簽發 token 回傳（可使用 FastAPI OAuth2PasswordBearer自動處理HTTP Header）。設置一個受保護路由作測試，例如 GET `/users/me` 需驗證 JWT 才能返回當前用戶資訊。為此，在 FastAPI dependency 中使用 OAuth2PasswordBearer，撰寫 verify_jwt 的協助函式。
  驗收標準： 能成功註冊新用戶，資料庫確實儲存了哈希後的密碼而非明文。調用登入API，以正確憑證獲得JWT token。之後在Swagger UI嘗試使用「Authorize」按鈕填入JWT，再調用受保護路由 `/users/me`，應得到對應使用者資訊而非401錯誤 59 。如果提供錯誤密碼，登入API返回401且不發Token。整個流程符合作安全常規 58 ，JWT內容（可解碼檢查）包含用戶識別資訊如user_id，簽名正確。此時我們的應用已具備基礎認證機制，支援多使用者環境下的權限控管。

- **Day 34：**
  理論概念： 實現部落格主要功能：文章的新增、編輯、刪除（CRUD）操作，並確保只有有權限的用戶

可以執行相應操作（例如只有作者或管理員能編輯刪除自己的文章）。討論權限設計：簡單起見可引入用戶角色(role)如admin/user，在JWT中攜帶 60 。學習使用 FastAPI 的Depends機制將權限檢查封裝成 reusable dependency。

學習資源： FastAPI 安全章節中關於角色權限的案例。參考資料庫中我們設計的文章表 52 和用戶表，決定是否增加字段如 is_admin。在知乎等平台搜索「FastAPI 權限管理」，看看社群建議的簡單方案。

AI 工具運用： 讓 Codex 幫助寫一個 Depends 權限檢查：例如「撰寫一個FastAPI dependency，檢查當前用戶是否為某文章作者」。AI 可能會寫一個函式從 token解析 user_id，再比對文章作者id，不符則 raise HTTPException(403)。我們據此改良並整合到路由定義中。

實作任務： 新增以下路由：PUT `/posts/{id}` 更新文章（需要作者本人或admin權限），DELETE `/posts/{id}` 刪除文章（同權限要求）。實作時，在依賴參數中加入 Depends(get_current_user) 取得當前登錄用戶，然後檢查其身份：如果文章作者id不符且非admin，回傳403。成功則進行資料庫更新或刪除（SQLAlchemy提供簡單的方法，或用Session操作）。確保對應的 Pydantic 模型和Swagger文件正確描述這些操作（如PUT需要文章內容，DELETE無內容）。

驗收標準： 登入普通用戶A後嘗試編輯別人文章，API返回403禁止 60 ；編輯自己的文章則200成功且資料庫更新正確。登入admin用戶執行刪除，能成功刪除任意文章；非admin刪除別人文章則被拒絕。使用Swagger進行這些操作時，能清楚地在介面上看到需要 Bearer Token 才能調用（因路由用 Depends OAuth2PasswordBearer 保護）。整套文章CRUD流程完整，權限控制如預期生效，說明我們後端業務邏輯趨於完善。

- **Day 35：**

理論概念： 部署前的準備與檢查：討論後端的結構是否可以更模組化（例如將路由按資源拆分進不同文件，使用 `APIRouter` 組織）。了解環境變數與設定管理，確保如資料庫URL、秘鑰等敏感資訊不寫死在代碼中。熟悉 FastAPI 提供的中間件和請求日誌功能，如有需要可添加簡單日誌。

學習資源： FastAPI 官方文檔「應用結構」部分，建議的專案佈局和使用 APIRouter 的方式。閱讀 RealPython 關於 FastAPI 部署的文章，檢查還需考慮的配置如 UVicorn worker 數，是否要用 Gunicorn 搭配。查閱 Python 套件 python-dotenv 或 Pydantic 的 BaseSettings，用於管理環境配置的簡易方法。

AI 工具運用： 讓 AI 分析我們的 backend/main.py，如果檔案已變得很長，請它建議拆分方案。AI 或許會建議創建 `routers/post.py`, `routers/user.py` 等並使用 APIRouter 34 。我們可依據此進行調整。此外，請 AI 列出部署FastAPI到雲端的步驟，為下週自動化部署做心理準備。

實作任務： 重構後端專案結構：建立 `routers` 資料夾，將文章相關路由定義搬至 `routers/post.py`（使用 `router = APIRouter()` 註冊路徑，再在 main.py 引入）。同理，分類和使用者路由也拆分。設定檔部分：建立 `.env` 檔保存 DATABASE_URL、SECRET_KEY 等，使用 python-dotenv 在應用啟動時讀取，或使用 Pydantic BaseSettings 建立 Settings 類方便全局使用。確認 .gitignore 已排除 .env 以免洩漏。最後，完整跑一遍測試流程，確保重構沒有引入新錯誤。

驗收標準： 專案結構更加清晰合理，打開 main.py 看到主要是引入路由、設定中介件、事件處理等，全局邏輯一目了然。各子路由模組能獨立運行測試，不相互依賴。應用行為與重構前保持一致，所有測試用例（可考慮之前Swagger操作當作手動測試）全部通過。設定檔作用正確，例如故意更改 .env 的 DB 路徑可以影響應用連接不同DB。至此，我們的後端開發工作告一段落，下一步將進入部署與效能優化階段。

第 7 週：前後端串接與整合測試

經過前幾週的開發，我們已有功能齊全的前端 React 應用與後端 FastAPI API。本週目標是在本地將兩者串接起來，實現從前端操作真正調用後端 API，驗證整個系統端到端工作正常。我們也將編寫自動化測試（單元測試和簡單的端對端測試）來確保主要功能不被日後修改破壞。此外，本週也關注開發體驗：設置跨域請求（CORS）允許，以及 Cursor/Codex 在整合調試時的應用。

- **Day 36：**

理論概念： 跨域資源共享 (CORS) 概念及必要性：前端開發伺服器和後端API一般不同源，瀏覽器默認會

攔截跨域請求，需後端設置CORS頭允許。了解在 FastAPI 中啟用 CORS 的方式（Middleware 或 `FastAPI().add_middleware` 簡單配置）。同時複習前端怎樣發送 HTTP 請求：使用 fetch 或 Axios 庫。討論 error handling：例如如果後端返回非200，前端應如何處理（提示用戶等）。
學習資源： FastAPI 官方文件「CORS」章節提供了使用 `CORSMiddleware` 的範例。MDN 上關於 CORS 的文章講解了預檢請求、允許頭等細節。查看 Axios 中文教程（若選用Axios）了解用它進行 AJAX 的基本姿勢。也可重溫先前MDN fetch資料。
AI 工具運用： 使用 Codex 寫一段 FastAPI CORS 中間件配置，比如允許 http://localhost:5173（Vite默認）來源的請求。AI 會給出具體代碼設定 allow_origins 等，我們可直接貼入。 41 。前端部分，如果要用 Axios，可請 AI 幫忙安裝和示範GET/POST代碼。若用 fetch，也可讓 AI 改寫我們之前的假資料函式為真正的 fetch 請求。
實作任務： 在 FastAPI 應用啟動部分，加入 CORS 中間件：允許本地開發的前端網址及需要的HTTP方法、Headers（記得包括 Authorization 供JWT）。確認後端重啟無誤。前端這邊，移除之前假資料的 `fetchPosts` 模擬實作，改用 `fetch('http://localhost:8000/posts')` 真正抓取。把其他相關的前端 API 調用都串接上：如登入表單呼叫 `/users/login` 獲取 token，文章管理功能呼叫對應 API。利用 React 的 useEffect 等，在適當時機發送請求並處理回應更新狀態。
驗收標準： 啟動後端 (`uvicorn main:app`) 和前端 (`npm run dev`)，在瀏覽器中操作前端UI能夠真正影響後端資料：例如登入後進入前端文章編輯頁，修改內容點保存，前端發PUT請求後後端資料庫確實更新。開發者工具Network標籤顯示請求和回應都成功，沒有因CORS被拒絕 61 。在未登入狀態下調用需要JWT的操作，前端正確收到401並可提示用戶登入。基本上，前後端順利溝通，各功能流程從UI到資料庫閉環打通。

- **Day 37：**
理論概念： 編寫自動化測試以覆蓋主要功能：包括後端單元測試/集成測試和前端的元件測試或簡單E2E測試。學習 PyTest 這個Python測試框架，用於測試FastAPI路由（FastAPI有TestClient可模擬請求）。同時了解React應用的測試方法，如使用 React Testing Library 測試元件，或用 Cypress 進行瀏覽器端E2E測試。本日重點在於建立基本測試，未求全面。
學習資源： FastAPI 官方文件「測試」章節示範了如何使用 Starlette 的 TestClient 測試api請求流程。閱讀 PyTest 中文指南快速入門，學習編寫測試函數及斷言方法。React Testing Library 官方文件和示例程式碼，可參考簡單範例（如渲染一個組件，斷言文字是否出現）。若考慮E2E，Cypress 官網有快速開始教學 62 。
AI 工具運用： 請 AI 生成幾個後端測試案例：例如「使用PyTest測試GET /posts端點，期望返回200和JSON列表」。Codex 會輸出TestClient用法，很實用。我們也能讓 AI 幫忙寫React Testing Library測試：如「測試PostList組件在給定posts列表時正確渲染標題」。AI 產出的代碼往往可直接運行或稍作修改。
實作任務： 在 backend 建立 tests/ 資料夾，撰寫 test_main.py 等。撰寫測試：1) 無需驗證的API，如 GET /posts 初始應該空列表，先POST創建文章，再GET應有資料。2) 有驗證的API，如未附JWT訪問保護路由應得401。可使用 TestClient 幫忙處理 Cookie/header。對前端，若時間允許建立一兩個元件測試檔案，如 App.test.tsx：模擬渲染App並測試初始畫面出現"登入"按鈕等等。將這些測試整合進 `npm test` 腳本或 CI 流程。
驗收標準： 在本地運行 `pytest` ，所有後端測試通過，覆蓋了主要業務邏輯 (文章CRUD, Auth)。運行 `npm test` 前端測試通過（若使用React官方CRA則內建，Vite需自行配置）。為驗證，故意改壞某功能（例如改錯某路由URL），確認測試會失敗，然後復原功能測試又通過。測試為我們提供了基本的回歸保護。把測試代碼也提交至 Git，確保後續可以在 CI 中自動跑測。

- **Day 38：**
理論概念： Continuous Integration (CI) 流程設置：在 GitHub 上配置 GitHub Actions，自動執行測試與建構，為之後部署做準備。理解 YAML 工作流程文件的基本構成，如 on: [push] , jobs: build , steps 等。學習使用 Actions 提供的官方套件，如設定 Node 環境、Python 環境的 action。討論將來CD（部署）步驟也可一併納入。
學習資源： GitHub Actions 官方文件的入門指南，了解如何在專案中新增 workflow 檔案。參考現成範

本，例如 Vite 專案的CI配置或 FastAPI 專案CI配置（vintasoftware模板可能附帶GitHub Actions設定[63] [64]）。閱讀一篇「前後端一起CI的實踐經驗」的部落格，吸取要點如分開測試矩陣等。

AI 工具運用： 讓 AI 幫忙生成 GitHub Actions 設定。例如要求：「寫一個GitHub Actions workflow，在ubuntu上分別設定Node16和Python3.9，安裝依賴後跑前後端測試」。Codex 有極高機率給出正確的YAML配置（包括 actions/setup-node 和 actions/setup-python 等步驟）。我們可按需修改版本號和腳本命令。

實作任務： 在專案 `.github/workflows/ci.yml` 撰寫 CI 腳本：當 push 或 PR 時觸發。工作流程包含兩部分：前端 - 使用 Node 環境，執行 `npm ci` 安裝和 `npm run build` 構建以及 `npm test`；後端 - 使用 Python 3.9+, 安裝依賴（`pip install -r requirements.txt`），啟動一個資料庫服務（如 postgres service，或 SQLite 無需），執行 `pytest`。可以將兩部分放在同一job的不同steps，也可拆為併行jobs。Push 後觀察Actions執行結果，若有失敗修正之。

驗收標準： 在 GitHub 上看到 Actions 工作流程被正確觸發且成功通過。日誌顯示前端建構與測試都OK[65]、後端測試也全部通過。為驗證可靠性，可故意破壞一項測試再推送，觀察 Actions 變為失敗。確認回復後再次推送又綠燈。CI pipeline 現在保障了我們的代碼品質，每次修改都會自動驗證不出錯，這為持續部署奠定基礎。

- **Day 39：**

理論概念： 應用效能與監控：討論在開發階段如何進行效能分析，例如前端使用 Lighthouse 評分，後端考慮UVicorn日誌中請求處理時間等。理解N+1查詢等常見後端效能坑。介紹簡單的優化策略：如啟用後端的緩存機制（FastAPI可結合Cache library），前端打包時開啟代碼分割減少首屏資源。監控方面，了解日誌和錯誤報告工具的重要性，部署後可引入 Sentry 等服務。

學習資源： Google Developers 文檔「Web 性能最佳實踐」, 了解幾項提升性能的要點如資源緩存、減少重排等。FastAPI 官方的性能比較頁面（與Node、Go對比）[66]，說明FastAPI已很快，但我們仍需謹慎編寫查詢。閱讀 SQLAlchemy 官方文件關於 lazy loading 的說明，避免不必要查詢。Sentry 官方網站瞭解接入簡要步驟。

AI 工具運用： 問 Codex 「如何找出我FastAPI應用的效能瓶頸？」AI 可能建議使用Profiler或APM工具，或在中間件計時。讓 AI 審視我們的某個查詢代碼是否存在N+1，AI 若識別出比如我們查詢文章列表時對每篇又查分類，可建議用JOIN優化。這對我們改善代碼有啟發。

實作任務： 使用 Chrome Lighthouse 對前端進行一次測試，記錄性能分數並分析瓶頸（若有圖像未壓縮、JS包過大等）。對後端，以開發模式壓力測試（可寫個腳本多次調用 API）觀察響應時間。如果發現明顯問題，嘗試優化：例如為文章列表引入簡易快取，第一次請求後結果緩存，短時間內再次請求直接返回快取。或在資料庫查詢上加上 `.options(selectinload(...))` 之類避免N+1。將這些優化點記錄在專案文件或注釋中。

驗收標準： 優化後再次測試性能：Lighthouse 分數有所提升（如從80提高到90以上），後端大批量請求壓力下平均響應時間有所下降。這些改善可能不巨大，但培養了性能意識。確認功能沒有因優化而變化或引入bug（回歸跑測試確保）。至此，我們的專案在本地環境運行良好且較為高效，準備進入最後的部署階段。

- **Day 40：**

理論概念： 部署上線前的最後檢查與準備。統整專案文件：更新 README 描述專案功能、架構及使用方法。添加適當的註解與文件，方便未來維護。學習容器化概念，討論是否需要用 Docker 部署（通常後端會容器化，前端可以 Vercel 平台直接拉取建置）。瞭解部署目標平台的要求，以 Vercel+Fly.io 為例，一個擅長靜態/前端部署，一個適合容器化後端。

學習資源： Vercel 官方教學「部署Next.js或靜態前端」以及 Fly.io 官方文檔「部署帶資料庫的後端應用」。回顧 nextjs-fastapi 模板部屬文章[30] [39]，了解我們將實現類似目標。閱讀 Docker 最佳實踐，準備撰寫 Dockerfile。

AI 工具運用： 讓 AI 列出我們專案部署的步驟清單，作為行動計劃。諮詢 Codex 生成一個 Python FastAPI 的 Dockerfile，以及 Node 前端的 Dockerfile，做為參考起點。AI 產出的Dockerfile通常頗為標準（使用 official image、install deps、expose port 等），我們稍作修改即可。

實作任務： 編寫 Dockerfile for backend：使用 Python 3映像，複製程式碼、安裝 pip 依賴，設定CMD使用 uvicorn 啟動。編寫 Dockerfile or config for frontend：如果選擇用 Vercel，可以省略，否則可用

Node映像 build 靜態檔，再用 nginx 提供靜態服務。將這些 Docker / 部署相關文件加入Git。更新 README 部署部分，列出環境變數需求、啟動指令、Docker用法等。
驗收標準： 在本地使用 Docker 構建並執行容器測試：例如 `docker build -t myblog-backend .` 然後 `docker run -p 8000:8000 myblog-backend` ，確認服務正常啟動並可提供 API。前端若有容器也類似測試。README 清晰易懂，新來的人跟隨即可跑起系統。本地一切就緒後，我們將在下週正式將服務部署到線上平台，完成最終目標。最後 commit 並確保 CI 綠燈，萬事俱備，只待上線。

---

1 2 42 Cursor要凉？OpenAI发布的Codex让我彻底躺平了！ - 53AI-AI知识库|大模型知识库|大模型训练|智能体开发
https://www.53ai.com/news/LargeLanguageModel/2025051951782.html

3 6 7 8 Git 工作流程 | 菜鸟教程
https://www.runoob.com/git/git-workflow.html

4 Git初学者教程：1小时内学会Git〖中英字幕，分P〗_哔哩哔哩_bilibili
https://www.bilibili.com/video/BV1pt4y1q7Kv/

5 GitHub - doggy8088/Learn-Git-in-30-days: 這是 Will 保哥在 2013 第 6 屆 iT 邦幫忙鐵人賽年度大獎的得獎著作〖30 天精通 Git 版本控管〗，歡迎大家 fork 我，如果有看見任何文字勘誤，也歡迎利用 pull request 來通知我修正，謝謝！
https://github.com/doggy8088/Learn-Git-in-30-days

9 學習Git 版本控管：新手上路篇(命令列操作) - YouTube
https://www.youtube.com/watch?v=WxFSad6II34

10 git使用流程图转载 - CSDN博客
https://blog.csdn.net/fuzongjian/article/details/50762993

11 12 git flow完整工作流流程图模板 - ProcessOn
https://www.processon.com/view/628f1c48e401fd21dc2d5da5

13 Git 工作流程- 阮一峰的网络日志
https://www.ruanyifeng.com/blog/2015/12/git-workflow.html

14 15 16 17 18 19 20 21 3小时前端入门教程（HTML+CSS+JS）_哔哩哔哩_bilibili
https://www.bilibili.com/video/BV1BT4y1W7Aw/

22 45 FastAPI 教程| 菜鸟教程
http://www.runoob.com/fastapi/fastapi-tutorial.html

23 24 Python调用OpenAI API示例_openai python-CSDN博客
https://blog.csdn.net/buchizhuti/article/details/129442647

25 TypeScript 入门教程
https://ts.xcatliu.com/

26 前言| TypeScript 新手指南 - Will 保哥的IT 創業之路
https://willh.gitbook.io/typescript-tutorial

27 入门教程- TypeScript 中文文档
https://nodejs.cn/typescript/handbook/

28 《TypeScript 教程》发布了- 阮一峰的网络日志
https://www.ruanyifeng.com/blog/2023/08/typescript-tutorial.html

29 TypeScript 入门, 基础语法, 从入门到精通, 持续更新中 - GitHub
https://github.com/946629031/typeScript

30 Creating a Scalable Full-Stack Web App with Next.js and FastAPI
https://medium.com/@pottavijay/creating-a-scalable-full-stack-web-app-with-next-js-and-fastapi-eb4db44f4f4e

31 32 33 34 40 41 63 64 65 GitHub - vintasoftware/nextjs-fastapi-template: State of the art project template that integrates Next.js, Zod, FastAPI for full-stack TypeScript + Python projects.
https://github.com/vintasoftware/nextjs-fastapi-template

35 React 官方中文文档
https://zh-hans.react.dev/

36 React.js 中文开发入门教学- 第一个React 组件_哔哩哔哩
https://www.bilibili.com/video/BV1dQ4y1h7Qy/

37 〖2019油管最好的一套React学习教程〗全面学习React： 01 React课程介绍和学习方法_哔哩哔哩_bilibili
https://www.bilibili.com/video/BV1f441117Bf/

38 Next.js FastAPI Starter
https://vercel.com/templates/next.js/nextjs-fastapi-starter

39 Best Practices for Structuring a Next.js + FastAPI + Supabase Project?
https://forum.cursor.com/t/best-practices-for-structuring-a-next-js-fastapi-supabase-project/49706

43 61 66 交互式API 文档升级 - FastAPI
https://fastapi.tiangolo.com/zh/

44 教程- 用户指南 - FastAPI
https://fastapi.tiangolo.com/zh/tutorial/

46 47 fastapi框架快速学习_哔哩哔哩_bilibili
https://www.bilibili.com/video/BV1Ya4y1D7et/

48 50 55 GitHub - JedersonLuz/fastapi-prisma: A template that starts up a FastAPI server that uses Prisma to store data.
https://github.com/JedersonLuz/fastapi-prisma

49 51 52 53 个人博客数据库设计（无E-R图 粗略版）_博客用户信息表数据库设计-CSDN博客
https://blog.csdn.net/d875708765/article/details/108418291

54 Python 如何连接并操作数据库（如Postgresql），有没有对应的ORM ...
https://www.zhihu.com/question/716684730

56 57 58 59 〖方向盘〗通俗易懂版讲解JWT和OAuth2，以及他俩的区别和联系（Token鉴权解决方案）_spring-boot-security ouath2 jwt 的区别-CSDN博客
https://blog.csdn.net/f641385712/article/details/83930699

60 什么是OWASP？什么是OWASP Top 10？ - Cloudflare
https://www.cloudflare.com/zh-cn/learning/security/threats/owasp-top-10/

62 使用Cypress框架敲开E2E自动化测试的大门原创 - CSDN博客
https://blog.csdn.net/ljx11606/article/details/141476655