

https://youtu.be/hdOabtYl1yw?list=PLIAziZzCjtLJQzoK-1n6R_KVTXm4tHZqP

https://youtu.be/T26wqcqoSug?list=PLIAziZzCjtLJQzoK-1n6R_KVTXm4tHZqP

這份教學資源旨在引導學生從課堂練習過渡到具備商業潛力的 **Side Project** 開發，強調在 AI 協作時代，真正的競爭力已不再是單純的寫程式，而是具備 **想法與策劃能力**。講者鼓勵學生以自身需求出發，透過 **自然成長 (Organic Growth)** 的模式打造產品，並強調建立 **CICD (持續整合與部署)** 流程以確保系統的穩定性與專業度。文中特別指出，開發者應學會利用 **行為追蹤碼** 來收集客觀的數據，藉此洞察使用者行為並進行精準的功能優化。最終，這場教學希望學生能跳脫傳統被動學習，藉由 **專案驅動的自學** 模式，在軟體工程人才日益兩極化的趨勢下，培養出難以被取代的實戰思維與產品感知力。

這份教學原始碼主要探討在 AI 浪潮下，軟體工程師如何利用輔助工具提升專案價值，並將重心從單純的開發轉向更具層次的 **決策與規劃**。講者強調 AI 雖然能高效產出程式碼，卻無法取代人類對於 **使用者需求與創新想法** 的掌握，因此開發者應將自己定位為 **管理 AI 的「主管」**，負責引導方向並處理複雜的系統架構。

文中詳細介紹了兩大類實務工具：首先是 **使用者行為追蹤**，如開源的 **PostHog** 以及廣泛應用的 **GA4** 與 **Amplitude**，旨在透過數據分析（如留存率、漏斗分析與 A/B 測試）來優化產品體驗。其次是 **網路爬蟲技術**，涵蓋了處理靜態網頁的 **BeautifulSoup**、模擬真人互動的動態工具 **Puppeteer**，以及適合大規模抓取的框架 **Scrapy**。最後，講者特別提醒開發者必須遵循 **robots.txt 協定** 與法律規範，以確保數據採集的 **合法性與安全性**。

Discuss 自發性專案開發.

自發性專案開發（通常稱為 Side Projects）在現代軟體工程與 AI 協作時代中，已成為個人能力成長與職業競爭力的核心。根據來源內容，這類專案不應僅是課程作業的終點，而是一個持續進化的開始，能幫助開發者在技術、產品感官與策劃能力上脫穎而出。

以下是針對自發性專案開發的深入探討：

1. 為什麼要進行自發性專案開發？

- **因應軟體產業的 M 型化趨勢**：未來的軟體工程人才分佈將趨向 M 型化，純粹的程式開發門檻因 AI 輔助而降低，真正的競爭力在於 **想法與策劃能力**。
- **在競爭激烈的市場中脫穎而出**：在求職困難的環境下（如美國 CS 畢業生面臨的挑戰），具備自發性專案經驗的人，對產品的理解與開發管理能力通常更強，這成為讓履歷被看見的關鍵。
- **實踐「專案驅動學習」(Project-driven education)**：比起傳統被動接收知識，透過解決專案中的未知問題與進行系統整合，能更有效率地掌握新技術並體會其背後的原理。

2. 如何開啟一個成功的專案？

- **從「自己會用的需求」開始**：不知道做什麼時，應尋找 **需求驅動** 的題目。最理想的起點是開發一個「自己每天都會用」的工具，例如個人筆記系統（如 Notion 的克隆版）、自動化郵件管理或 AI 小幫手。
- **避免「腦補式」開發**：不要為了挑戰技術而設計一堆無人使用的功能，否則專案會變成缺乏邏輯的「LP project」。專案的複雜度應隨著真實需求的增加而自然演進。

- **專注於「簡單夠用」**：不需擔心市場上已有成熟產品，重點在於能否定義出一個「簡單夠用」且具備自訂靈活度的工具，並讓它維持在「好用」的狀態。

3. 開發過程的核心心法

- **建立 CI/CD 流程**：養成良好的軟體工程習慣，確保程式碼隨時可以部署（Deploy）到可試用的環境，並利用 Git 進行版本管理與測試。
- **追求自然成長（Organic Growth）**：在 AI 時代，流量與算力成本昂貴，不建議早期花錢做行銷。應**低調地把產品做好**，吸引身邊的「鐵粉」使用，並透過他們的真實反饋來修正產品。
- **利用追蹤碼理解使用者行為**：透過在代碼中埋入追蹤碼，客觀地觀察使用者的操作流程（User Journey），找出他們為何放棄使用的原因，這比政治正確的訪談更具參考價值。

4. AI 時代的新挑戰與工具

- **溝通勝於背誦**：現在寫程式更像是用自然語言與 AI 溝通，開發者必須對背後的技術架構有概念，才不會整晚與 AI 「打太極」。
- **善用新型協作協議**：例如 **MCP（Model Context Protocol）**，它能讓不同的工具（如設計工具與開發工具）在 AI 介入下實現順暢的資訊串聯與自動化。

總結來說，自發性專案開發就像是經營一座自己的私人花園：你不需要一開始就蓋出一座森林，而是從種下一棵自己喜歡的樹（解決自己的需求）開始。隨著你不斷修剪（根據行為追蹤修正）與細心灌溉（持續整合與部署），這座花園會吸引志同道合的人駐足，最終它展現出的獨特性與生命力，將是你個人能力最好的證明。

評分標準中，專案順利 deploy 可獲得幾分？

合法進行行為追蹤需確保哪兩個關鍵權利？

軟體工程中的 CI/CD 分別代表什麼含義？

Discuss 使用者行為追蹤。

使用者行為追蹤（User Behavior Tracking）在軟體開發中，是將開發者的「想像」轉化為「真實需求驅動」的關鍵技術。根據來源內容，這項技術不只是為了收集數據，更是為了確保專案能朝著正確的方向進化，避免開發出無人問津的產品。

以下是針對使用者行為追蹤的詳細討論：

1. 核心目的：從客觀數據發現真實需求

- **超越不透明的訪談**：來源指出，傳統的使用者訪談往往不可靠，因為受訪者常基於禮貌或「政治正確」給出虛假的正面回饋。相比之下，行為追蹤能提供最客觀的數據，揭示使用者在產品中的真實操作情形。
- **找出「棄用」的原因**：透過追蹤，開發者可以精確觀察使用者在哪個環節被嚇跑或感到困惑，進而找出產品失敗的真正原因（通常是因為沒有真實需求或流程不順）。
- **驅動專案演進**：藉由了解行為，開發者能將專案從「簡單夠用」的狀態，根據回饋注入複雜度，使其維持在「好用」的狀態，實現**自然成長（Organic Growth）**。

2. 實作方式與追蹤指標

- **埋設追蹤碼**：這是在原始碼中加入特定邏輯，記錄使用者與 APP 的互動。
- **追蹤範疇**：
 - **基礎事件**：包含頁面瀏覽 (Page View)、登入 (Login) 及各種點擊行為。
 - **進階行為**：甚至包含頁面停留在哪、捲動速度等，用以判斷使用者對內容的感興趣程度。
- **分析模型**：
 - **使用者旅程 (User Journey)**：觀察使用者從進入到離開的完整流程。
 - **漏斗分析 (Funnel)**：藉此觀察在註冊、使用到留存的各個階段，使用者是在哪裡流失的。

3. 隱私權與法律合規性

行為追蹤是一把「**雙面刃**」，開發者必須在優化產品與侵犯隱私之間取得平衡。

- **台灣法規與告知義務**：在台灣，合法追蹤的關鍵在於**明確告知**收集目的、使用範圍與保存期限，並取得使用者同意。
- **隱私保護措施**：
 - **匿名化處理**：使用 Hash 或隨機 UID 代替真實身分，並遮罩部分 IP 地址。
 - **賦予選擇權**：必須讓使用者有權拒絕追蹤，且拒絕後仍能使用基本服務，並提供刪除紀錄的管道。
 - **第三方服務規範**：若使用如 **PostHog** 或 **Amplitude** 等第三方分析工具，應在隱私聲明中明確告知。

4. 相關工具與流程整合

- **分析工具**：**PostHog** 被提及可用於匿名化事件分析，而 **Amplitude** 則可用於更深入的數據統整。
- **視覺化整合**：現代行為追蹤工具通常內建視覺化看板 (Dashboard)，開發者無需額外串接複雜的視覺化工具即可判讀數據。
- **開發循環**：行為追蹤應與 **CICD** 流程結合，確保開發者能隨時根據數據調整並快速部署新版本。

使用者行為追蹤就像是為產品裝上了**全天候的行車記錄器**：它不只是在意外（使用者流失）發生時提供證據，更重要的是它能讓你隨時校準導航方向，確保你不是在自己腦補的荒野中漫無目的地駕駛，而是正行駛在使用者真正需要的道路上。

課程期末專案在 12 月 2 日的評分重點為何？

為何建議從「自己會用的需求」開始開發 Side Project？

開發者應如何合法地對使用者進行行為追蹤？

Discuss 隱私權保護規範。

根據來源內容，**隱私權保護規範**在現代軟體與專案開發中扮演著「**雙面刃**」的角色。開發者透過埋設追蹤碼來了解使用者行為，藉此優化產品，但這同時也涉及個人隱私的侵犯。

以下是根據來源整理出的隱私權保護規範要點：

1. 個人資料收集的定義

在開發專案（如 Side Projects）時，若涉及以下任何一項行為，通常會被歸類為「收集個資」：

- **識辨個人身份的資料**：包含電子郵件（email）、IP 地址、手機號碼、帳號 ID 等。
- **識辨個人行為模式**：間接紀錄使用者的操作流程與習慣。
- **使用追蹤技術**：在未經同意下使用 Cookie 或追蹤腳本（Tracking code）。
- **第三方分享**：將上述資料傳送給第三方服務（如 Google Analytics、Posthog 或 Amplitude）進行統計分析。

2. 法律合規的核心原則（以台灣為例）

來源指出，要使收集行為合法且不構成侵犯隱私，最核心的原則是「**明確告知與同意**」。具體規範包括：

- **告知義務**：必須明確告知使用者收集資料的**目的、使用範圍以及保存期限**。
- **使用者同意**：在使用者開始使用服務前，應提供一個隱私條款步驟，讓使用者點擊同意。
- **保護聲明**：聲明資料是否經過匿名化處理，以及是否會轉交給第三方用於改善體驗。

3. 開發實務中的保護措施

為了符合隱私規範，開發者在實作時應採取以下具體行動：

- **提供拒絕權**：應讓使用者有拒絕被追蹤的權利，且在拒絕後仍能繼續使用服務。
- **被遺忘權**：若使用者事後反悔，開發者必須具備能從資料庫中刪除該使用者紀錄的能力。
- **匿名化處理（Anonymization）**：
 - 使用雜湊（Hash）或隨機產生的 **UID** 來代替真實身份。
 - 對 **IP 地址** 進行去識別化（例如僅保留 IP 前半段）。
 - 設定追蹤工具（如 Posthog）啟用「匿名事件」功能。
- **縮短保留時間**：個資不應無限期保存，應設定合理的自動刪除時間。
- **倫理審查**：若收集的資料是為了用於學術研究或訓練 AI 模型，則需事先通過**倫理審查委員會**的審查。

4. 結語：雙贏的局面

雖然隱私規範看似限制了開發，但其本質是為了建立開發者與使用者之間的信任。來源強調，這是一個**雙贏（Win-win）**的局面：開發者透過合法且透明的行為追蹤來發現產品斷點、優化流程；使用者則能獲得更順暢、更符合需求的服務體驗，同時不用擔心私生活被無端窺探。

你可以將隱私權規範想像成「**商店的監視器告示**」：店家為了防盜或優化動線（改善體驗）而安裝監視器，但必須在門口張貼告示（明確告知），確保顧客知道自己被錄影，且錄影畫面僅用於安全管理而非販售個資（特定目的與範圍），這樣才能在維護營運需求的同時，保障顧客的安心感。

如何定義與達成專案的 MVP 目標？

取消作業七後，其配分如何遞補至總分？

在 AI 時代，軟體工程師應具備哪些核心能力？

Discuss CICD 持續整合。

在現代軟體工程，特別是開發 Web App 或自發性專案（Side Projects）時，**CICD（持續整合與持續部署）**已成為開發者必須嚴格遵守的核心習慣。根據來源內容，CICD 不僅是技術流程，更是一種確保專案隨時處於「可用狀態」的管理哲學。

以下是針對 CICD 的詳細討論：

1. 持續整合（Continuous Integration, CI）

持續整合的核心在於確保代碼的品質與系統的穩定性，特別是在團隊協作或頻繁更新時。

- **防止系統崩潰**：CI 的目的是確保當開發者將程式碼 Check-in（提交）到分支（Branch）時，不會破壞（Break）其他人已經寫好的功能。
- **確保整合順暢**：透過自動化的流程，讓程式碼的整合過程可以持續且順利地進行，避免在專案後期才面臨巨大的整合困難。

2. 持續部署與交付（Continuous Deployment / Delivery, CD）

持續部署強調的是自動化發佈的效率，讓想法能快速轉化為可運行的成品。

- **隨時可部署與試用**：開發者應確保專案隨時都能部署到一個可以自行試用的環境。這對於專案的開發與測試至關重要。
- **自動化更新**：當遠端倉庫（Remote）的版本更新時，系統應具備自動部署的能力，使產品隨時處於可展示（Demo）或可交付（Deliver）的狀態。

3. 開發實務與工具習慣

要成功實踐 CICD，開發者需要建立良好的軟體工程規範：

- **Git 為核心**：熟練使用 Git 是實踐 CICD 的基礎。開發者應利用 GitHub 進程式碼管理，並養成適時使用 Tagging（標記）、撰寫測試（Test）以及進行重構（Refactoring）的習慣。
- **專案管理的指標**：在課程專案中，**順利部署（Deploy）**是評分的重要標準。即使架構簡陋，只要能成功部署並穩定運作，就代表具備了基本的專案管理能力。
- **解決環境差異**：CICD 能幫助開發者及早發現 Local（本地）開發環境與雲端部署環境之間的差異與錯誤，避免在最後一刻才面臨無法部署的窘境。

4. CICD 的重要性：從課程到職業生涯

來源強調，CICD 不應只為了應付期末報告，而是為了培養**想法與策劃能力**。在 AI 協作時代，寫程式的門檻降低，開發者更需要透過 CICD 流程來管理日益複雜的專案需求，並透過持續的迭代與部署，讓專案從一個簡單的 Prototype（原型）演進為成熟的產品。

結語與比喻： **CICD** 就像是為專案安裝了一套「**自動化輸送帶與品質檢測儀**」：CI 是檢測儀，確保每一件送上帶子的零件（程式碼）都不會卡住機器或弄壞其他零件；CD 則是輸送帶，確保只要零件合格，就能立刻被送到展示櫃（部署環境）供人參觀或使用。這讓開發者不必等到整台機器蓋完才擔心它動不動得了，而是能**邊蓋邊跑，隨時修正**。

如何合法地進行使用者行為追蹤？

課程中 P 階段的評分標準為何？

為何建議從自己會用的需求開始做專案？

Discuss AI 協作開發.

在 AI 協作開發的時代，軟體工程的本質正在發生劇烈變化。根據來源，寫程式已從單純的代碼撰寫轉向「**對話式**」開發，這不僅降低了技術門檻，也重新定義了開發者的核心競爭力。

以下是針對 AI 協作開發的深入討論：

1. 從「編寫代碼」轉向「溝通與規劃」

- **對話式開發模式**：現在開發者更常以自然語言與 AI 溝通來產出程式碼。AI 可以快速整理資源、提供範例並大幅提升學習速度。

- **技術架構的重要性**：來源特別強調，即便有 AI 輔助，開發者若對**技術架構沒有概念**，很容易陷入「跟 AI 打太極」的窘境，耗費整晚溝通卻無法解決問題。有效的協作建立在開發者能精確地進行**溝通、規劃與設計**之上。

- **複雜度的管理**：在實作專案時，開發者負責整體的框架規劃，AI 則能根據規劃迅速寫出多層次的代碼。然而，確保這些代碼能「動起來」並順利**部署（Deploy）**到環境中，仍是開發者的核心責任。

2. 核心競爭力的轉位：想法與策劃

- **人才分佈 M 型化**：隨著 AI 降低了開發門檻，傳統的軟體能力分佈將從常態分佈轉向 **M 型化**。

- **想法與策劃能力（Planning Ability）**：在技術與開發不再是高門檻後，未來的勝負關鍵在於**想法與策劃能力**。這包括對需求的敏銳度、產品感官（Product Sense）以及管理複雜專案的能力。

- **解決「未知」的能力**：專案開發過程中會遇到許多未知的整合問題，開發者必須具備探索與研究的精神，而非單純依賴 AI 產出已知的知識。

3. AI 協作的新技術與工具

- **模型上下文協議（MCP, Model Context Protocol）**：這是為了讓不同 AI 工具（如 Cursor、Figma 等）之間能順利串聯而提出的協議。它允許在不同工具間傳遞溝通的結論與設計檔，達成**自動化的工作流**。

- **雲端環境的除錯（Debugging）**：雖然 AI 能幫忙寫 code，但將專案部署到雲端時，常會遇到本地環境沒有的錯誤。開發者需要具備底層知識（如網路、流量管理、資料庫優化），才能下出正確的指令（Prompt）讓 AI 協助排除雲端環境的故障。

4. 結語：AI 時代的開發者角色

AI 協作開發讓開發者能更快速地將想法落實為應用程式。然而，這也意味著開發者必須從「工人」轉型為「建築師」：

AI 協作開發就像是擁有了一名**技術極強但缺乏主見的實習生**。如果你清楚建築的藍圖（技術架構）並能下達清晰的指令（策劃能力），這名實習生能幫你以驚人的速度蓋好大樓；但如果你自己都不知道房子的結構，你們只會在地基周圍原地打轉，最終蓋出一座無法入住的廢墟。

如何合法地進行使用者行為追蹤？

課程取消作業七的具體原因為何？

講師對「專案驅動教育」的定義為何？

Discuss AI 與工程師合作.

在 AI 協作開發的浪潮下，軟體工程師的角色正經歷一場「典範轉移」。根據來源內容，AI 與工程師的合作不再是單向的自動化，而是一種**管理與策劃的深度協作**。

以下是針對 AI 與工程師合作關係的詳細討論：

1. 職位角色的重新定義：主管與「超級菜鳥」

- **AI 是「超級菜鳥」(Super Junior)**：來源將 AI 定義為一名技術強大但缺乏主見的初級開發者。它能透過一兩個指令（Prompt）快速產出功能看似完整的 App 程式碼，甚至能根據開發者的規劃一次寫出七、八層代碼。

- **工程師是「主管」與「建築師」**：工程師的角色轉變為 **Senior 或主管**。主管不需要親自寫每一行程式碼，其核心任務是**提供方向、給予指導、進行策劃**，並決定產品為何存在。

2. 核心競爭力的轉移：想法與策劃

- **軟體人才 M 型化**：隨著 AI 降低了純程式撰寫的門檻，傳統技術能力的分佈將從常態分佈走向 **M 型化**。

- **想法與策劃能力 (Planning Ability)**：在 AI 時代，真正的勝負關鍵在於**想法與策劃能力**。AI 雖然可以克隆（Clone）現有的產品（如 Notion），但無法取代人類對需求的觀察與創新的決策。開發者必須能縮短「從眼睛到腦袋」的距離，主動發現需求並將其落實為應用。

3. 技術架構：與 AI 溝通的共同語言

- **避免「打太極」**：來源警告，如果工程師對**技術架構沒有概念**，在與 AI 協作時會花費整晚「打太極」（溝通無果）。有效協作的前提是工程師具備足夠的專業知識來下達精確的指令（Professional Prompt）。

- **對話式寫程式**：現代開發已轉變為「對話式」過程。工程師利用自然語言與 AI 溝通，讓 AI 整理資源、提供範例，從而大幅提升學習效率與開發速度。

4. 工具與協議的演進：MCP 的應用

- **模型上下文協議 (Model Context Protocol, MCP)**：為了讓不同的 AI 工具（如 Cursor、Figma、Linear 等）能順暢協作，MCP 應運而生。它允許在不同工具間傳遞溝通結論與設計檔，達成**跨工具的自動化工作流**。例如，將設計工具的輸出直接交給開發工具作為參考，實現無縫開發。

5. AI 無法取代的領域：整合與運維

- **解決雲端與部署問題**：雖然 AI 能寫 Code，但專案部署 (Deploy) 到雲端時常遇到環境差異導致的錯誤。工程師仍需具備底層知識 (如網路、資料庫優化、流量管理) 來進行 **Debugging** 與效能調優。
- **持續整合 (CI/CD) 的落實**：確保專案隨時處於「可用且能動」的狀態，仍需依靠工程師建立良好的 **CI/CD** 習慣與軟體工程實務。

總結來說，AI 與工程師的合作就像是「導遊與全能司機」的關係：AI 是那位體力無限、熟悉所有道路的司機 (提供強大的開發火力)，但導遊 (工程師) 必須決定目的地在哪裡、規劃導覽邏輯，並在遇到道路封閉 (雲端報錯或環境不相容) 時，利用專業經驗指引司機繞道前行。沒有導遊的規劃，司機只會載著你在原地打轉。

如何評分 Final Project 的 Prototype？

收集各資時應採取哪些匿名化措施？

軟體開發中的 CI 與 CD 分別指什麼？

Discuss PostHog 行為分析.

PostHog 是一個**開源的產品分析平台**，被視為目前開源界中最成熟、功能最完整的開發者工具之一。它不僅能記錄基本的數據，更提供深度洞察，協助開發者將自發性專案從「腦補需求」轉化為「數據驅動」的成功產品。

以下是關於 PostHog 行為分析的詳細討論：

1. 核心功能與應用

根據來源，PostHog 整合了多種強大的分析功能，讓開發者能全方位掌握使用者動態：

- **事件追蹤 (Event Tracking)**：能自動或手動捕捉 App 上的任何互動 (如點擊、登入、觸發特定功能)。
- **行為錄影與熱點圖 (Session Recording & Heatmaps)**：開發者可以**回放使用者的操作歷程**，觀察他們在頁面上的真實動線，或透過熱點圖了解哪些區域最常被點擊。
- **漏斗分析 (Funnel Analysis)**：定義一套使用者流程 (如：進入頁面 → 點擊註冊 → 完成任務)，藉此觀察使用者在哪個步驟「斷掉」並放棄使用。
- **留存率分析 (Retention)**：分析有多少使用者在初次使用後會再次回訪。來源特別提到，軟體公司非常看重如「過去七天內有四天使用」這類指標，以識別真正的「鐵粉」。
- **AB 測試與功能開關 (Feature Flags)**：開發者可以在後台設定開關，讓一部分用戶使用新版面、另一部分使用舊版面，從而比較行為指標的差異。

2. 技術實作與架構

PostHog 專為技術導向的開發團隊設計，具備高度的靈活性：

- **SDK 與初始化**：開發者需透過 SDK 初始化 PostHog 物件，並設定特定的 API Key 與雲端資料庫進行串接。
- **Capture 函式**：在程式碼中的關鍵邏輯處 (如 `use_effect` 或點擊事件中) 加入 `capture` 指令，即可將特定行為數據傳送至後台。

- **ClickHouse 資料庫**：PostHog 內建 **ClickHouse**，這是一種**縱向儲存 (Columnar Storage)** 的資料庫。相對於傳統的橫向儲存，它能極快地進行重向統計分析（例如：有多少人點了某個按鈕），讓原本需要幾十秒的分析運算在瞬間完成。
- **自託管 (Self-hosting)**：由於它是開源的，開發者可以選擇自建平台，特別適合對技術有研究興趣或需要高度客製化的團隊。

3. 對專案開發的戰略意義

在自發性專案開發中，使用 PostHog 的意義在於：

- **獲取客觀回饋**：比起受訪者可能因為禮貌而給出的虛假稱讚，PostHog 記錄的行為數據是最客觀的。
- **驅動「自然成長」(Organic Growth)**：開發者可以根據數據修正產品斷點，讓產品在不花錢做行銷的情況下，透過良好的體驗吸引並留住使用者。
- **隱私合規**：PostHog 提供**匿名化事件**設定，能自動協助處理如 IP 位址去識別化等隱私保護措施，讓開發者在優化體驗的同時也符合隱私規範。

結語與比喻：PostHog 行為分析就像是為你的應用程式安裝了一套「實驗室等級的精密監測儀」。它不僅能告訴你病人（使用者）在哪裡感到不適（流失），還能讓你像科學家一樣，透過對照組實驗（AB 測試）找出最有效的治療方案，確保你的專案不是在黑暗中摸索，而是朝著正確的方向進化。

簡述 Midterm Project Review 的評分標準與項目。

法律上如何合法進行使用者行為追蹤？

比較 Beautiful Soup 與 Puppeteer 的技術差異。

Discuss 使用者數據追蹤。

使用者數據追蹤 (User Data Tracking) 是現代軟體開發中，將開發者的「想像」轉向「真實需求驅動」的關鍵機制。根據來源內容，這項技術不只是为了收集數據，更是為了確保專案具備生命力，避免開發出無人問津的產品。

以下是針對使用者數據追蹤的深入探討：

1. 核心目的：獲取客觀的真實回饋

- **超越不透明的訪談**：來源指出，傳統的使用者訪談往往不可靠，因為受訪者常基於禮貌或「政治正確」給出虛假的正面回饋。**數據追蹤能提供最客觀的資訊**，揭示使用者在產品中的真實操作情形。
- **發現真實需求**：許多專案失敗是因為「沒有需求」，而開發者往往透過數據追蹤才能發現使用者為何「棄用」產品，進而調整方向或功能，確保專案是**由需求驅動而非開發者腦補**。
- **識別「鐵粉」與推動自然成長**：透過分析留存率 (Retention)，例如觀察「過去七天內有四天使用」的指標，開發者可以找出真正的忠實用戶，並根據其行為優化體驗，達成 **Organic Growth (自然成長)**。

2. 實作方式與追蹤範疇

使用者數據追蹤主要透過在原始碼中埋設**追蹤碼 (Tracking code)** 來實現。

- **追蹤範疇**：包含基礎的頁面瀏覽（Page View）、登入（Login），以及各種點擊事件與觸發動作。甚至可以追蹤使用者捲動頁面的速度，以判斷其對內容的感興趣程度。

- **分析模型**：

- **漏斗分析（Funnel Analysis）**：觀察使用者在特定流程（如：進入頁面 → 註冊 → 完成任務）中在哪個階段流失。

- **使用者旅程（User Journey）**：透過日誌（Log）完整觀察使用者的操作流向。

- **Session Recording（行為錄影）**：如 PostHog 提供的功能，可以回放使用者的操作歷程，了解他們如何與介面互動。

3. 隱私規範與合規原則

數據追蹤是一把「**雙面刃**」，開發者必須在優化產品與侵犯隱私之間取得平衡。

- **法律合規（以台灣為例）**：只要**明確告知**收集目的、使用範圍與保存期限，並取得使用者同意，即不構成侵權。

- **保護措施**：開發者應提供**拒絕追蹤的權利**，並落實**匿名化處理**（例如使用隨機 UID 代替真實身分、遮罩部分 IP 位址）。

- **互利關係**：這應被視為一種 **Win-win（雙贏）** 的局面：開發者獲得優化依據，使用者則獲得更順暢、更符合需求的體驗。

4. 主流分析工具比較

根據來源，目前開發者常用的工具有以下幾種：

- **PostHog**：最成熟的開源工具，具備完整的功能（包含 AB 測試、Feature Flags），並使用 **ClickHouse 資料庫**來實現極速的縱向統計分析。

- **GA4 (Google Analytics 4)**：以**事件導向（Event-based）**為核心，強項在於利用 Google 的 AI 進行預測分析，並能與 BigQuery 整合進行複雜查詢。

- **Amplitude**：在 SaaS 新創公司中非常受歡迎，支援大數據量的分析，並提供對初期專案非常友好的免費額度（每月約 10 萬次記錄）。

使用者數據追蹤就像是為產品裝上了**全天候的行車記錄器**：它不只是在發生意外（使用者流失）時提供證據，更重要的是它能让開發者隨時校準導航方向，確保產品是行駛在使用者真正需要的道路上，而非開發者自以為是的荒野中。

如何利用 PostHog 的 Feature Flag 進行功能管控？

靜態爬蟲與動態網頁爬蟲的主要技術差異為何？

執行網路爬蟲時需注意哪些法律風險與江湖規矩？

Discuss 網頁爬蟲技術.

網頁爬蟲技術（Web Crawling）是現代軟體開發中獲取外部資訊的重要手段。根據來源，爬蟲是一種自動化工具，其運作核心是給予一個網址，讓程式能讀取該網頁的原始碼並將所需的資訊「爬」取下來，進而進行結構化處理（如轉換為 JSON 或 XML 格式）以供後續分析或應用。

以下是針對網頁爬蟲技術的分類、工具及法律風險的詳細討論：

1. 爬蟲技術的主要分類

根據網頁的性質與抓取需求，爬蟲技術可分為以下幾類：

- **靜態爬蟲**：專門抓取內容不依賴 JavaScript 動態載入的靜態網頁。其優點是速度快、消耗資源極低，但無法處理需經點擊或動態加載才能顯示的內容。
- **動態爬蟲**：當網頁需要點擊按鈕或執行 JavaScript 才能載入內容時，需使用此類技術。它能完整模擬使用者的行為（如點擊、捲動），確保能抓取到動態生成的資料。
- **API 爬蟲**：透過服務提供者（如 Facebook）官方開放的 API，在獲得授權後合法抓取資料。這比模擬點擊更穩定且安全，常用於粉專管理或自動化回覆。
- **框架型與大量爬蟲**：用於系統化地抓取大規模資料，例如爬取整個電商網站的所有商品分類或所有新聞種類。
- **攔截網路請求**：這是一種較為進階且處於灰色地帶的技術，直接攔截瀏覽器的請求來獲取資料。

2. 主流爬蟲工具介紹

來源中介紹了三種在 Python 與開發圈內常見的工具，各有其適用場景：

• Beautiful Soup：

- **特性**：極為輕量且常用於 Python 生態圈。
- **用途**：主要處理 HTML 檔案，將其轉換成樹狀結構（類似 DOM），方便開發者透過標籤、屬性或 CSS 類別來選取資料。

• Puppeteer：

- **特性**：以「無頭瀏覽器」（Headless Browser）的形式運作，不需實際展示網頁畫面即可模擬瀏覽行為。
- **用途**：模擬使用者登入、點擊按鈕或截圖，非常適合用於處理動態網頁或進行自動化測試。

• Scrapy：

- **特性**：功能強大的專業框架，適合大規模、長時間的爬取任務。
- **用途**：具備內建的排程器（Scheduler）與下載器（Downloader），能有效管理資料流（Pipeline）並進行 ETL（資料轉換）處理。

3. 法律風險與道德規範

開發者在實作爬蟲時必須謹慎面對法律紅線：

- **智慧財產權**：若網頁內容註明禁止複製或受版權保護，不可隨意爬取。

- **非法入侵**：禁止繞過登入驗證、防爬機制或圖形驗證碼（CAPTCHA），這可能涉及未經授權存取的法律問題。
- **禁止 DDoS 行為**：爬蟲應加入延遲（Delay），模擬人類瀏覽頻率（如每秒 1-2 次），避免因短時間大量存取造成對方伺服器癱瘓。
- **江湖規矩（Robots.txt）**：應遵守網站根目錄下的 robots.txt 檔案，其中標註為 Disallow 的部分應禁止爬取。

4. 專案開發中的應用策略

在 Side Projects 中，爬蟲常與其他技術結合。例如，利用爬蟲定期抓取專業新聞，再交由 **AI 進行摘要（Summary）**，能大幅節省閱讀時間。開發者可以根據需求彈性組合工具，例如先用 **Scrapy** 進行大規模抓取，再將特定網頁交給 **Puppeteer** 處理動態互動。

網頁爬蟲技術就像是一名數位速讀員：他可以幫你跑遍各大圖書館（網站），迅速翻閱並抄錄你指定的資訊。但這名速讀員必須遵守圖書館的開放時間與規則（robots.txt 與法律規範），且不能因為翻書太快把書弄壞（DDoS 攻擊），這樣才能持續且合法地為你的研究提供養分。

說明開發者應如何合法且具備倫理地進行使用者行為追蹤。

試從需求驅動的角度，論述 Side Project 的題材挑選原則。

比較靜態爬蟲與動態網頁爬蟲在技術實現上的核心差異。

Discuss 法律與道德規範。

在進行軟體開發與專案實作時，**法律與道德規範**是開發者必須嚴格遵守的底線，特別是在涉及「使用者行為追蹤」與「網路爬蟲」這兩大領域時。根據來源內容，這些技術雖是優化產品的利器，但若使用不當，極易觸犯隱私權或著作權。

以下是針對法律與道德規範的詳細討論：

1. 使用者行為追蹤與隱私保護

行為追蹤能幫助開發者了解使用者需求，但也可能侵犯隱私，是一把「**兩面刃**」。

- **個資收集的判定**：只要涉及收集可識別身份的資料（如 Email、IP 地址、手機號碼、帳號 ID），或間接識別個人行為模式、未經同意使用 Cookie 及追蹤腳本，都屬於收集個資的範疇。
- **合規核心原則（以台灣法律為例）**：合法的關鍵在於「**明確告知與同意**」。開發者必須清楚告知收集的**目的、使用範圍及保存期限**，並讓使用者在開始服務前點擊同意。
- **具體的道德與技術實踐**：
 - **匿名化處理**：應使用雜湊（Hash）或隨機產生的 **UID** 來代替真實身份，並對 IP 地址進行去識別化（例如僅保留前半段）。
 - **賦予選擇權**：必須提供讓使用者**拒絕被追蹤**的機制，且拒絕後仍能使用基本服務。
 - **被遺忘權**：若使用者事後反悔，開發者應具備能刪除其紀錄的能力。

- **第三方服務聲明**：若將資料傳送給如 PostHog 或 Amplitude 等第三方工具進行分析，必須在隱私條款中明確聲明。

- **特殊用途審查**：若資料用於學術研究或 AI 模型訓練，則需事先通過**倫理審查委員會**的審查。

2. 網路爬蟲的法律與道德邊界

來源指出，爬蟲本身是中性的技術手段，違法與否取決於「爬取的方式」與「資料的用途」。

• 著作權與存取限制：

- 不可隨意複製受著作權保護的內容。

- **嚴禁繞過驗證機制**：不可試圖繞過登錄驗證、反爬蟲機制或圖形驗證碼（CAPTCHA），這類行為可能涉及非授權存取罪。

• 尊重「江湖規矩」（Robots.txt）：

- 開發者應遵守目標網站的 robots.txt 協議，若該文件明文禁止爬取特定路徑，合法的爬蟲不應強行進入。

• 避免造成對方的負擔（防止 DoS 攻擊）：

- 爬蟲不應以過高的頻率存取網頁，以免造成對方伺服器癱瘓。開發者應加入適當的延遲（如每 1-2 秒一次），**模擬人類的瀏覽頻率**，避免演變成惡意的阻斷服務攻擊（DoS）。

• 資料使用倫理：

- 應優先抓取公開資料，並在引用或發表研究成果時清楚標註資料來源。

3. 建立互信的開發者文化

遵守規範不僅是為了避開法律訴訟，更是為了建立開發者與使用者之間的信任，達成一種**雙贏（Win-win）**的局勢。當開發者能透明、合法地處理數據時，使用者能獲得更好的體驗，產品也能在健康的環境下演進。

總結來說，法律與道德規範就像是「**交通規則**」：開發技術（如開車）雖然給予了我們極大的自由與速度，但如果無視紅綠燈（隱私告知）或強行衝撞私人領地（繞過驗證碼），最終不僅會造成他人損害，也會讓自己面臨法律的制裁。唯有在規則內行駛，技術的進步才能真正服務於社會。

如何定義一個好的 side project 題目？

PostHog 具備哪些主要的功能與特色？

網頁爬蟲在法律上有哪些潛在風險？