

https://youtu.be/_eF9S8wCHzY?list=PLIAzIZzCjtLJQzoK-1n6R_KVTXm4tHZqP

https://youtu.be/Mq5Wkol2tjl?list=PLIAzIZzCjtLJQzoK-1n6R_KVTXm4tHZqP

這份教學原始碼主要針對網頁開發中的**品質保證與自動化測試**工具進行深入講解，旨在提升學生的側邊專案與產品化技術。講師首先詳細介紹了用於靜態程式碼檢查的 **ESLint** 以及自動格式化工具 **Prettier**，強調這些工具能確保程式碼風格一致並提前修復潛在錯誤。接著，內容轉向動態測試領域，介紹了目前主流的單元與整合測試框架 **Jest**，以及由微軟開發、能模擬多種瀏覽器行為的端對端測試工具 **Playwright**。此外，文中也明確規範了期末專題的**評分機制與繳交流程**，特別提到將全面採用**雲端部署版本**進行評核。最後，講師透過一套**二次方程式定義貢獻度比重**，藉此鼓勵團隊成員間的勞務平均分配，並期望學生在 AI 輔助開發的時代仍能維持高品質的軟體工程實踐。

這份教學資源詳細說明了軟體開發中從**專案初期提交到後期維運監控**的完整流程與數位工具應用。課程首先規範了期中原型繳交的細節，強調以 **Markdown 檔案與部署連結** 作為核心，並鼓勵學生建立 **CI/CD 自動化流程**，利用 **GitHub Actions** 實現持續整合與部署。為了確保服務上線後的穩定性，講者進一步介紹了 **Datadog** 與 **Sentry** 等應用程式效能管理 (APM) 工具，協助開發者即時偵測錯誤並優化使用者體驗。最後，內容探討了多樣化的**專案管理框架與工具**，如傳統的 **Jira**、**Trello** 以及新興具備 AI 輔助功能的 **Linear**，旨在幫助開發團隊在時程壓力下維持高效的溝通與文件化協作。

Discuss 期末專題規範.

根據提供的來源，關於「期末專案 (Final Project)」的規範與要求，可以整理成以下幾個核心部分：

1. 繳交規範與目錄結構

- **目錄命名：**專案必須放置在 WP141 儲存庫下的 **final 目錄**中。如果有多個服務 (multiple services)，可以在該目錄下自行安排子目錄，但應盡量遵循慣例以利助教閱讀代碼。
- **GitHub 設定：**儲存庫必須保持為 **Private (私有)**。若未來想開源，建議另外建立一個 Public 儲存庫，並將同組成員加入為 Collaborators。
- **時程提醒：**剩餘時間約三週，學生需掌握進度。

2. 雲端部署與地端測試

- **雲端部署為重：**評分將以**雲端部署版本**為主，不再強制地端 (Local host) 必須能跑起來作為評分標準。
- **註冊服務與安全：**若專案包含登錄功能，必須提供註冊服務。學生需對專案的**網路安全性**負責，若因疏忽導致被駭，需自行承擔責任。
- **流量說明：**由於雲端服務多有流量限制，若有特殊流量需求或開啟時間限制，需在 README.md 中詳細說明。
- **代碼完整性：**雖然以雲端為主，但所有代碼仍需完整 Check-in 至 GitHub，以便助教在有疑慮時拉回地端進行人工 Debug 或檢視代碼狀況。

3. 必要繳交材料與宣傳

除了程式碼，每組還必須在截止日前完成以下事項：

- **Demo 影片**：長度以 **6 分鐘**為限。影片需包含成員介紹、題目簡介、功能展示、系統架構及使用的技術。
- **臉書社團貼文**：需在指定社團發文分享，標題格式為 [14-1 WP]，內容需包含專案名稱、影片連結、服務描述、部署連結（選填）、GitHub 連結（若開源）及技術心得。
- **資訊調查表**：需填寫 Google 表單，提供組別、姓名、題目名稱、部署網址、GitHub 網址、影片及臉書連結，以便評分時查找。
- **README.md 文書**：需包含上述臉書貼文內容，並額外標註**地端安裝測試步驟**、**成員分工項目**、外掛成員（如有）、引用資源說明以及對課程的建議。

4. 評分機制與同儕互評

- **評分團隊**：每組專案由**教授（佔 40%）與四位助教（各佔 15%）**共同評分。
- **互評表 (**eval.txt*)**：每組需在 `final` 目錄下建立一個 `eval.txt`，列出成員學號與**貢獻度分數**（加總需等於 100）。
- **貢獻度權重**：個人最後得分會根據貢獻度進行非線性調整。若貢獻度低於平均，分數會大幅下降（貢獻度 0 則分數 0）；若高於平均，加分幅度較小，最高上限為組別分數的 **110%**。

5. 扣分與更新規定

- **更新彈性**：截止日後仍可持續更新雲端版本，但助教會參考截止當日 9 點的 Source Code 以確保代碼與服務對應。若在評分後才更新，助教不予重新評分。
- **常見扣分 (Penalty)**：
 - 環境變數 (`.env`) 寫不清楚導致無法測試。
 - UI 瑕疵、操作不順或功能異常（視嚴重程度扣 3% 至 30% 以上）。
 - 逾期繳交或漏交影片、臉書貼文、互評表、調查表或 README 內容不全。

為了提升專案品質與代碼的可讀性，建議學生使用 **ESLint** 進行靜態檢查，並搭配 **Prettier** 自動格式化程式碼。在測試方面，推薦使用 **Jest** 進行單元測試，以及 **Playwright** 進行模擬瀏覽器行為的 End-to-End 測試，這能有效避免在新增功能時破壞舊有功能。

你可以將這個評分系統想像成一場「非線性加權」的接力賽：雖然全組共享一個完賽成績，但如果某位隊員完全沒起跑（貢獻度為 0），他的成績將會是 0 分；而若某位隊員跑得特別努力，他能獲得的額外獎勵也是有限的，這旨在鼓勵全體隊員平均貢獻，而非由單人扛起所有工作。

影片長度與應包含的內容為何？

Jest 的主要特色與功能有哪些？

貢獻度分數如何影響最終成績？

Discuss 品質保證工具.

在軟體開發中，**品質保證 (Quality Assurance, QA) 工具**不僅能協助開發者解決眼前的專案問題，更能引入先進的軟體工程觀念，提升技術造詣。根據來源內容，品質保證工具主要可分為**靜態代碼檢查與格式化**，

以及**自動化測試**兩大類：

1. 靜態檢查與代碼格式化：ESLint 與 Prettier

這類工具旨在確保程式碼的整潔與風格統一，避免低級錯誤（Stupid bugs）。

• ESLint (Linting)：

- **概念：**「Lint」原意指衣物上的棉絮，在程式開發中則是指**檢查程式碼中不美觀、容易出錯或不符合最佳實踐的寫法**。

- **功能：**ESLint 是針對 JavaScript 和 TypeScript 的強大**靜態檢查工具**，能強制執行統一的編碼風格，並檢查潛在的邏輯問題。

- **自訂化：**開發者可以自訂規則（Rules），或套用業界公認的規範，如 **Airbnb**、**Google** 或 **StandardJS**。

- **自動修復：**使用 `--fix` 參數可以自動修復大部分的格式錯誤。

• Prettier (Formatting)：

- **概念：**是一個「有主見（Opinionated）」的**自動化格式化工法**。

- **功能：**它專注於代碼的視覺美感，例如對齊、行長度等。當開發者存檔時，Prettier 會自動觸發並美化程式碼。

- **團隊協作：**透過統一的 `.prettierrc` 設定，可以確保團隊成員寫出的代碼外觀一致，避免因為空格或縮排差異引發無謂的爭議。

2. 自動化測試工具：Jest 與 Playwright

測試工具分為單元測試（Unit Test）、整合測試（Integration Test）以及端對端測試（End-to-End Test）。

• Jest (單元與整合測試)：

- **特點：**由 Meta (Facebook) 開源，是目前最受歡迎且穩定的測試框架，能與 React 或 Next.js 等框架完美整合。

- **功能：**支援 **Watch Mode**（監控模式）、**Coverage Reports**（測試覆蓋率報告）以及 **Mocking**（模擬數據，例如模擬特定日期的運行狀況）。

- **開發習慣：**建議開發者在編寫模組時同步撰寫測試檔（通常放在 `__tests__` 目錄），確保留下的代碼邏輯正確。

• Playwright (端對端測試)：

- **特點：**由微軟開發，是 Puppeteer 團隊的後續作品，支援**所有主流瀏覽器**及多種程式語言。

- **功能：**模擬真實人類行為，如**點擊、打字、滾動、上傳**等。它能處理伺服器端渲染（SSR）的畫面，並進行視覺化回歸測試（如截圖或錄影）。

- **平行測試：**內建平行執行機制，能同時模擬多個操作流程以提高效率。

3. AI 輔助與回歸測試

在現代開發流程中，AI 工具（如 Cursor）能大幅減輕撰寫測試的負擔。來源建議養成好習慣：每當新增功能時，應請 AI 協助編寫相應的測試，並執行**回歸測試（Regression Test）**，確保新功能不會破壞原有的舊功能。

品質保證工具就像是建築工地的「自動檢測儀」與「水平儀」：ESLint 和 Prettier 確保每一塊磚頭都擺放得整齊、符合標準；而 Jest 和 Playwright 則像是壓力測試，確保房屋在實際使用（如強風或地震）時不會倒塌。有了這些工具，開發者就不需要用肉眼辛苦檢查每一行代碼，能更專注於創造核心功能。

如何定義期末專題的「貢獻度分數」？

請列舉三種被廣泛接受的程式碼風格模板。

ESLint 與 Prettier 在功能上有何主要差異？

Discuss ESLint 與 Prettier.

在現代軟體開發流程中，**ESLint** 與 **Prettier** 是確保程式碼品質（Quality Assurance）與團隊協作效率的兩大核心工具，它們能協助開發者產出整潔、統一且易於維護的程式碼。

以下根據來源內容，對這兩項工具進行詳細討論：

1. ESLint：靜態代碼檢查（Linting）

- **核心概念**：「Lint」一詞原意是指衣服上沾染的**棉絮**，在程式開發中則是指那些讓程式碼看起來不舒服的錯誤、風格問題或不符合最佳實踐的寫法。
- **工具特性**：ESLint 是針對 **JavaScript** 與 **TypeScript** 的強大檢查工具，主要進行**靜態檢查**（Static Check），即在編譯時期（Compile time）而非執行時期（Runtime）檢查程式碼邏輯。
- **功能與優勢**：
 - **避免低級錯誤**：能識別潛在的邏輯漏洞或曖昧不明的寫法，減少「Stupid bugs」的產生。
 - **自訂規範**：支援自訂規則（Rules），也允許開發者套用業界公認的規範模板，如 **Airbnb**（被認為最漂亮）、**Google**（嚴謹且企業化）或 **StandardJS**（不使用分號）。
 - **自動修復**：提供 `--fix` 參數，能自動修正大部分符合規範的格式錯誤。

2. Prettier：自動化格式化工具（Formatting）

- **核心概念**：Prettier 是一個「**有主見（Opinionated）**」的格式化工具，它擁有一套嚴格的規則，旨在讓團隊成員編寫的程式碼外觀完全一致。
- **工具特性**：它像是一個隨時待命的代理人（Agent），通常設定在**存檔時（Save）自動觸發**，主動將程式碼調整得美觀整齊。
- **功能與優勢**：
 - **視覺美化**：負責處理程式碼的對齊、縮排、行長度等細節，讓程式碼讀起來更輕鬆。

- **終結團隊爭議：** 透過 `.prettierrc` 設定檔，強制所有人遵守同一套標準（如縮排兩格或四格、左括弧位置等），避免成員因個人審美差異而互相比評代碼「很醜」。

3. 兩者的整合與最佳實踐

- **協同運作：** 雖然 ESLint 也有格式化功能，但兩者通常會搭配使用（如透過 `eslint-config-prettier`），讓 ESLint 專注於代碼邏輯檢查，而將美化工作交給 Prettier。

- **工作流建議：**

- **養成好習慣：** 建議在執行 `git push` 前先執行 Lint 檢查，確保進入儲存庫的代碼是乾淨的。
- **輔助 Code Review：** 統一的風格能大幅減輕在進行代碼審查（Code Review）時的痛苦，讓審查者專注於邏輯而非瑣碎的格式問題。
- **搭配 AI 開發：** 即使現在許多代碼由 AI（如 Cursor）生成，依然需要透過這些工具來編排與檢查，以維持專案整齊一致。

ESLint 與 Prettier 的關係就像是「衛生檢查員」與「儀容整理師」： ESLint 是衛生檢查員，確保你的身體健康、沒有隱疾（邏輯錯誤）；而 Prettier 則是儀容整理師，在你出門（存檔）前，自動幫你打領帶、衣服燙平，確保你跟團隊其他成員看起來都穿著同樣專業的制服。

期末專題影片的內容須包含哪些要點？

貢獻度評分公式如何影響組員的最終分數？

請簡述 ESLint 與 Jest 兩者功能的主要差異。

Discuss Jest 單元測試.

Jest 是目前在 JavaScript 開發社群中最受歡迎且穩定的**單元測試（Unit Test）與整合測試（Integration Test）**工具，由 Meta（原 Facebook）的前工程師 Christoph Nakazawa 開發並開源。

以下根據來源內容，詳細討論 Jest 的特性、使用方式及其在現代開發中的角色：

1. Jest 的核心地位與優勢

- **主流地位：** 雖然市場上曾有 Mocha、Vitest 或 AVA 等工具，但在近幾年，Jest 幾乎統一了測試工具的市場。
- **高度整合：** 它能與 **React**、**Next.js** 或 **Node.js** 等現代框架完美銜接，基本上是「開箱即用」（Zero-config），開發者不需像早期一樣為了處理不同版本的 JavaScript 語法而苦惱於 Babel 或 Webpack 的複雜設定。
- **強大功能：**
 - **Mocking（模擬）：** 內建強大的模擬功能，例如可以模擬特定的日期或數據流，以便在特定情境下測試模組。
 - **Coverage Reports（覆蓋率報告）：** 內建測試覆蓋率檢查，讓開發者知道目前寫的測試涵蓋了多少比例的原始碼。

- **Watch Mode (監控模式)**：能夠隨時監測代碼變動並即時跑測試，不需要等手動執行時才發現錯誤。

2. 單元測試的概念與實作

• **單元測試的定義**：指的是針對程式碼中**最小的邏輯單元 (模組)** 進行獨立測試，確保每個功能塊在進入 GitHub 進行整合前都是正確的。

• **撰寫方式**：

- 開發者通常會在 `__tests__` 目錄下建立 JavaScript 測試檔。
- 測試語法包含一個 `test` 函數 (描述測試內容) 以及內部的 **expect 斷言 (Assertion)**。
- **範例邏輯**：例如測試一個加法器，會呼叫該函式並使用 `.toBe(3)` 來期待結果等於 3。

• **執行自動化**：透過在 `package.json` 中設定 `npm test` 指令，可以一鍵跑完專案中所有的測試案例。

3. AI 時代下的測試習慣

來源強調，過去寫測試往往要花費與寫程式碼相當的時間，甚至更多，且代碼更新後若測試沒同步更新，會導致測試失效。但在當前 AI 輔助開發 (如使用 Cursor) 的背景下，負擔已大幅減輕：

- **自動生成**：建議開發者在完成一個階段 (Milestone) 時，請 AI 協助針對該段代碼編寫測試檔。
- **回歸測試 (Regression Test)**：這是最重要的習慣之一。每當新增功能時，必須重新執行之前的測試，確保新加入的代碼沒有把舊有的正確功能「弄壞」。

Jest 單元測試就像是建築工程中的「材料應力測試」：在整座大樓 (專案) 蓋好前，我們先確保每一根鋼筋、每一塊磚頭 (代碼模組) 都能承受預期的壓力。雖然這在前期會增加一些工作量，但它能確保當整座建築合攏時，不會因為其中一個小零件的瑕疵而導致整座建築崩塌。

課程期末專題的影片長度限制為幾分鐘？

ESLint 與 Prettier 在功能上有何主要差異？

Playwright 具備哪些模擬人類操作網頁的功能？

Discuss Playwright 測試工具.

Playwright 是近年來迅速崛起且功能強大的**端對端 (End-to-End, E2E) 測試工具**，主要用於模擬使用者在瀏覽器上的實際操作行為，以確保整個系統整合後的效能與使用者體驗 (UX) 符合預期。

以下根據來源內容，對 Playwright 的特性、功能與應用進行詳細討論：

1. 開發背景與技術優勢

- **頂尖團隊打造**：Playwright 是由原本在 Google 開發 Puppeteer 的團隊轉入 **Microsoft (微軟)** 後所開發的作品。
- **跨瀏覽器與語言支援**：相較於 Puppeteer 主要支援 Chromium，Playwright **支援所有主流瀏覽器**，且相容於多種程式語言。

- **處理伺服器端渲染 (SSR)**：對於需要在伺服器端渲染後才傳送到前端的網頁，Playwright 能像真實瀏覽器一樣等待畫面渲染完成再進行爬取或測試，這是一般單純爬蟲難以做到的。

2. 強大的模擬與測試功能

Playwright 能夠全方位模擬人類在網頁上的互動，其核心功能包括：

- **行為模擬**：包含**點擊 (Click)**、**打字**、**懸停 (Hover)**、**檔案上傳**、**滾動 (Scroll)** 等操作。
- **多裝置模擬**：可模擬不同尺寸的螢幕、行動裝置，甚至是特定的 User Agent (如 LINE 內建瀏覽器)。
- **內建平行測試**：為了反映人類平行操作網頁的真實情況，Playwright 內建了**平行執行 (Parallel)** 測試的機制，提高測試效率。
- **視覺化回歸測試**：提供 **Screenshot (截圖)** ****錄影功能****，讓開發者能直觀地檢視測試過程，進行視覺化的回歸測試，確認 UI 是否在更新後跑版。

3. 實作與開發習慣

- **腳本編寫**：雖然工具強大，但開發者仍需自行編寫測試腳本 (或請 AI 輔助)，明確定義要測試的流程。例如：測試 Google 搜尋時，需寫下前往頁面、填寫搜尋詞、按下 Enter 並預期 (Expect) 得到什麼結果的指令。
- **AI 時代的應用**：在使用 Cursor 等 AI 編碼工具時，建議養成每開發一個里程碑 (Milestone) 就請 AI 撰寫測試檔的習慣，並定期執行**回歸測試 (Regression Test)**，確保新功能沒有破壞原有的舊功能。
- **市場地位**：憑藉著與現代開發生態系的良好結合，Playwright 在短時間內已逐漸超越 Selenium、Cypress 等老牌測試工具。

Playwright 測試工具就像是專案的「機器人試車手」：如果說 Jest 負責檢查零件 (單元測試) 是否合格，那麼 Playwright 就是坐在駕駛座上，實際發動車子、踩油門、轉方向盤，確保整台車 (端對端系統) 在各種路況 (瀏覽器環境) 下都能順暢行駛，並把異常狀況拍照存證。

如何使用 ESLint 的 `--fix` 參數處理程式碼錯誤？

請列舉三項 Playwright 可模擬的人類網頁操作行為。

期末專題影片的長度限制與內容應包含哪些要素？

Discuss CI/CD 自動化.

持續整合 (Continuous Integration, CI) 與**持續交付/部署 (Continuous Delivery/Deployment, CD)** 是現代軟體開發最核心的觀念之一，許多技術都是圍繞著此觀念進行開發。其核心目標是透過自動化流程，確保多人協作的代碼能穩定整合並快速上線。

以下根據來源內容，詳細討論 CI/CD 的流程、環境區分及相關工具：

1. 持續整合 (CI) 的核心流程

CI 的重點在於**持續性的整併與驗證**，確保每位成員提交的代碼不會破壞主分支 (Main/Master) 的功能。

- **自動化檢查**：當代碼 Check-in 後，系統會自動執行 **Build（建置）、Test（測試，如 Jest）與 Linting（靜態檢查，如 ESLint）**。
- **成功指標**：只有通過上述所有自動化檢查的整合，才被視為一次成功的整合。這能有效防止開發者將「爛掉的代碼」推送到開發分支或主分支中。

2. 持續交付與部署 (CD) 的階層

CD 旨在確保只要代碼有更新，就能自動部署到相應的環境中。來源中特別強調了不同環境的區分：

- **Staging Environment（測試/預覽環境）**：當功能在開發分支（Develop branch）完成後，會先部署到 Staging 環境。此環境的配置與正式環境完全一致，用於進行最後階段的整合測試與**使用者驗收測試（UAT）**。
- **Production Environment（正式環境）**：只有在 Staging 環境確認無誤並合併至主分支（Main branch）後，才會自動更新至正式環境供所有使用者使用。
- **進階功能**：完善的 CD 流程還包含自動通知、**Rollback（回滾）** 機制，若部署後發現問題可快速恢復至舊版本。

3. CI/CD 自動化工具

來源介紹了幾款主流的 CI/CD 工具，開發者可根據專案需求選擇：

- **GitHub Actions**：目前 GitHub 生態系中的標準解決方案。它透過在 `.github/workflows` 目錄下的 **YAML 檔** 定義工作流程（Workflow）。它支援多種觸發事件（如 Push, Pull Request），並能安全地儲存 API 金鑰或 Token 等敏感資訊（Secrets）。
- **CircleCI**：輕量且介面親善的雲端平台，提供一定的免費額度，適合新創專案。
- **Jenkins**：歷史悠久且功能完備的開源自動化伺服器，是早期的主流選擇。
- **其他工具**：包含 GitLab CI/CD、Travis CI 等。

4. 實踐建議與重要性

雖然 CI/CD 的建置需要初期投入時間學習工具，但一旦建立完成，能大幅節省手動操作的無聊時間，並作為一種**防禦機制**。來源建議，在 AI 輔助開發的時代，開發效率極快（可能 3 到 5 天就是一個 Sprint），若沒有自動化的 CI/CD 流程，管理成本（Overhead）將會過高。

CI/CD 自動化就像是軟體工廠裡的「自動化生產線與品檢關卡」：每位工匠（開發者）完成零件後，不需要手動搬運或檢查，生產線（CI）會自動進行壓力測試與品質掃描，一旦合格便自動組裝並送往展示間（Staging），最後確認無誤後才正式上架（Production）。這確保了即便工廠日夜趕工，出貨的產品依然能維持穩定的品質。

如何透過 ESLint 與 Prettier 提升程式碼品質？

進行期末專題影片上傳有哪些具體規範？

Sentry 與 Datadog 在應用程式監控中有何功能？

Discuss GitHub Actions 應用.

GitHub Actions 是由 GitHub 官方開發與維護的自動化工具，旨在直接於 GitHub 平台內實現 **CI/CD**（**持續整合與持續交付/部署**）的工作流程。它已成為 GitHub 生態系中標準的自動化解決方案，能有效減少人工操作測試、建置與部署時的枯燥感與時間浪費。

以下根據來源內容，詳細討論 GitHub Actions 的核心應用與機制：

1. 實現 CI/CD 工作流程

GitHub Actions 的核心在於自動化處理軟體開發的多個階段：

- **持續整合 (CI)**：當團隊成員將代碼 Check-in 後，系統會自動執行 **Build**（**建置**）、**Test**（**測試**）以及 **Linting**（**靜態檢查**）。這能確保新加入的代碼不會破壞原有的功能（即執行回歸測試），並維持代碼風格的一致性。
- **持續交付與部署 (CD)**：確保代碼在通過測試後，能自動部署到最新版本，如 **Staging**（**測試環境**）或 **Production**（**正式環境**）。

2. 觸發機制與執行環境

GitHub Actions 透過特定的事件來觸發自動化任務：

- **觸發事件**：可設定於 **push****（推送代碼）、**pull_request****（發起合併請求）、**release****（發布版本）** 或自訂事件時啟動。
- **workflow 定義**：所有的工作流程都定義在專案目錄下的 **.github/workflows/**（注意非 **.git/**）資料夾中，並使用 **YAML 格式** (**.yml**) 檔案來編寫。
- **執行環境**：支援多種作業系統環境，如 **Ubuntu** 或 **Windows**，也可以使用 **Docker** 容器進行自架（Self-hosted）執行。

3. 安全管理與秘密資訊 (Secrets)

在執行自動化部署時，通常需要 API Key 或 Token，GitHub Actions 提供了一套安全機制：

- **秘密資訊存儲**：開發者可以在儲存庫的 **Settings -> Secrets and variables -> Actions** 中儲存敏感資訊。
- **安全性**：這樣可以讓 CI/CD 工具在不洩漏金鑰給外部人員的情況下，安全地完成部署工作。

4. 品質門檻與結果追蹤

- **保護分支**：可以將 GitHub Actions 設定為嚴格的檢查關卡，如果測試或 Lint 未通過，系統會 **Reject**（**拒絕**）該 PR，防止「爛掉的代碼」進入 **main** 或 **develop** 分支。
- **可視化檢視**：執行結果會顯示在儲存庫的 **Actions** 標籤頁中，成功會顯示綠色，失敗則顯示紅色，方便開發者快速定位問題。

5. 內建功能與擴充性

GitHub Actions 內建了許多常用動作，大幅簡化了腳本編寫：

- **Checkout 代碼**：自動拉取原始碼。

- **Setup 環境**：自動設定開發語言環境。
- **上傳成果**：將編譯後的 Artifacts 上傳或發布 Release。

GitHub Actions 應用就像是專案的「自動化管家」：當你把新的代碼交給它（Push），它會自動幫你檢查這段代碼是否乾淨（Linting）、結構是否穩固（Testing），確認沒問題後，再自動幫你把貨物送到正確的店面（Deployment）。這讓開發者可以從繁瑣的流程中抽身，專注於編寫核心功能。

說明 ESLint 與 Prettier 在程式開發中的功能差異。

試述 staging 與 production 環境在部署流程中的區別。

比較 Trello 與 Jira 在專案管理風格上的主要不同。

Discuss 應用程式效能管理。

應用程式效能管理（Application Performance Management，簡稱 APM），或稱為**監控（Monitoring）**，是軟體開發與維運（DevOps）中不可或缺的環節。其核心目標是讓開發者與維運團隊能夠即時**觀測、分析並優化**應用程式的效能，在問題真正影響到廣大使用者之前就發現瓶頸，確保服務的穩定性與優良的使用者體驗。

根據來源提供的資訊，應用程式效能管理可以從以下幾個層面深入探討：

1. APM 的兩大主要類別

目前的市場趨勢將監控工具分為兩大類，分別針對不同規模與需求的專案：

- **全方位觀測平台（Full-stack Observability）**：以 **Datadog** 為代表，這類工具提供極其完整的監控生態系，包含基礎設施、應用程式、網路安全及日誌分析。
- **錯誤追蹤與前端專注（Error Tracking & Frontend Focus）**：以 **Sentry** 為代表，這類工具對於新創專案或規模較小的 Side Project 非常友善，專注於捕捉程式碼中的異常（Exception）及監控前端渲染效能。

2. 核心監控功能與指標

一套完善的 APM 系統通常包含以下功能，協助團隊全方位掌握系統狀態：

- **基礎設施監控（Infrastructure Monitoring）**：即時監控伺服器、雲端資源或容器（如 Kubernetes）的使用率與健康狀況。
- **請求與效能監控**：追蹤端對端（End-to-End）的使用者請求。例如，若使用者的某次點擊回應超過兩秒，系統應能發出警訊。
- **日誌（Log）分析與過濾**：許多 Bug 無法從 UI 直接觀察，但會記錄在日誌中。APM 工具能自動分析 Log 並針對特定錯誤訊號發出警告。
- **合成監控（Synthetic Monitoring）**：透過工具「假裝」成使用者去模擬操作網頁，主動測試服務是否依然存活，而非被動等待使用者回報。
- **安全監控**：偵測是否有不正當的手法繞過登入機制，或是否正遭受 DDoS 攻擊。
- **數據庫監控**：觀察資料庫的操作是否耗時過久，或使用量是否異常暴增。

3. 實作機制：從捕捉到通報

在實作上，APM 工具通常需要透過以下步驟整合進專案：

- **安裝 Agent 或 SDK**：開發者需要在伺服器安裝 **Data Collector (Agent)**，或在程式碼中引入特定的 SDK（如 Node.js 的 `dd-trace` 或 Sentry 的 `library`）來收集數據。
- **優雅降級 (Graceful Landing)**：透過 `try-catch` 機制捕捉異常 (Exception)。來源強調，當錯誤發生時，開發者應透過 APM 工具確保程式能「優雅地跌倒」，而非直接讓螢幕白成一片，並將未預期的錯誤傳送至監控平台。
- **自動化告警**：開發者可以自訂警示規則 (Alerting Rules)，當錯誤率或回應時間超過閾值時，系統會自動透過 **Slack**、**Discord** 或電子郵件通知團隊。

4. 為什麼 Side Project 需要 APM？

來源指出，即便專案在部署 (Deploy) 上線時是完好的，也可能因為網路不穩、記憶體洩漏 (Memory Leak) 或同步非同步因素，導致服務在運行一段時間後進入崩潰 (Panic) 狀態。透過 APM 工具，開發者不需派人 24 小時守在螢幕前，就能確保服務「活著」。

此外，現代 AI 工具（如 Cursor）也能提供安裝指南，協助開發者快速在程式碼中建立這些監控設定。

應用程式效能管理就像是為你的專案裝上「全天候生命監控儀」：如果說 CI/CD 是確保產品在出廠前是健康的，那麼 APM 就是在產品上路後，隨時偵測心跳與血壓的醫療系統。它能確保當你的「數位產品」感到不適（出現 Bug 或變慢）時，能在病發（服務中斷）前第一時間發出求救訊號。

如何透過 ESLint 與 Prettier 提升程式碼品質？

簡述 CI 與 CD 在軟體開發中的核心觀念。

比較 Playwright 與 Puppeteer 兩款測試工具的差異。

Discuss 專案管理模式.

在軟體開發與專案執行中，**專案管理模式**的核心目標是在時間、預算與資源的限制下，進行有效的規劃、執行與交付。根據來源內容，專案管理不僅是進度追蹤，更包含溝通協調、風險管理、品質控制以及文件化。

以下詳細討論來源中提到的幾種主要專案管理模式與實踐建議：

1. 傳統與現代專案管理模式

- **瀑布式 (Waterfall)**：這是一種線性且階段性的過程，包含需求分析、設計、開發、測試到最後的上線。雖然被視為較傳統的模式，但其優點在於結構穩定且嚴謹。
- **敏捷式 (Agile/Scrum)**：強調將工作拆解成較小的**里程碑 (Milestone)** 與短週期的**衝刺 (Sprint)**。傳統的 Sprint 通常為期兩週（第一週開發，第二週測試優化），但在 AI 輔助開發的時代，合理的 Sprint 可能縮短至一週，甚至三到五天。
- **看板式 (Kanban)**：將所有任務視覺化為看板上的卡片，讓團隊成員能直觀地追蹤進度。這對於小型團隊或需要快速彈性調整的專案非常有效。
- **混合式 (Hybrid)**：結合了瀑布式的嚴謹性與敏捷式的快速反應能力，在穩定的框架下進行快速迭代。

2. 團隊協作與溝通機制

專案管理的成敗往往取決於溝通效率。來源建議避免使用 LINE 或 Messenger 等通訊軟體進行深度溝通，因為資訊容易混雜且難以管理。

- **集中化溝通**：推薦使用 **Slack**（訊息乾淨、具備專屬工作區）或 **GitHub Issues** 進行技術討論與 Bug 追蹤。
- **看板管理實務**：以 **Trello** 為例，建議建立不同的清單（List）來管理不同階段的任務：
 - **Features**：存放所有未來想做的想法與功能。
 - **To-do**：當前 Sprint 必須完成的目標。
 - **Next**：預計下一個階段要進行的項目。
 - **History/Done**：已完成並經過確認的歷史紀錄。

3. AI 時代下的角色與管理變革

隨著 AI 工具（如 Cursor）的介入，專案管理的門檻與角色界限正在模糊：

- **角色合併**：傳統上 PM、開發者、設計師（Designer）與品質保證（QA）由不同人擔任，現在透過 AI 輔助，單一成員可能身兼多職。
- **降低管理負擔 (Overhead)**：專案管理不應追求過度繁瑣的流程。來源強調應選擇「管理負擔最小」的工具，避免為了管理而管理。
- **自動化管理**：現代工具如 **Linear** 開始整合 AI，能 24 小時監控專案進度，並與 GitHub 串接，自動更新任務狀態，減少召開冗長管理會議的需求。

4. 貢獻度與同儕評估機制

在學術或團隊專案中，**同儕互評 (Peer Review)** 也是專案管理模式的一環，用於確保公平性。

- **互評表 (**eval.txt*)**：團隊需共同討論並定義成員的貢獻度，加總為 100 分。
- **非線性評分**：為了鼓勵全員參與並防止「搭便車」現象，個人分數會根據貢獻度進行調整。貢獻度低於平均者扣分幅度較大，而貢獻度極高者則有適度的加分（上限 110%），旨在獎勵實質投入而非鼓勵一人單挑所有工作。

專案管理模式就像是一場「接力賽的配速策略」：瀑布式像是傳統長跑，講求每一步的穩健；敏捷式則是連續的小衝刺，讓你能在每一段路程後根據體力（進度）調整呼吸。而現代 AI 工具則像是配備了智慧導航與數據監控的跑鞋，讓團隊能以更短的週期完成衝刺，並在發生偏離時即時修正航道。

請說明期末專題影片應包含的具體內容。

試比較 ESLint 與 Prettier 在代碼品質管理上的功能差異。

根據課程內容，CI/CD 流程中的「Staging」環境有何作用？

Discuss AI 輔助開發.

在現代開發流程中，**AI 輔助開發**（AI-Assisted Coding）已成為核心趨勢。來源指出，這門課程是第一次嘗試將 AI 導入教學，顯現出開發模式正經歷巨大的範式轉移。

以下根據來源內容，探討 AI 在開發各個階段的應用與影響：

1. 程式碼生成與質量管理

- **主流工具應用**：**Cursor** 是來源中多次提及的關鍵 AI 輔助工具，它不僅能生成程式碼，還能協助專案的基礎設定。
- **角色轉變**：未來大部分的程式碼可能不再是由開發者親手撰寫，而是由 AI 產出。然而，這不代表開發者可以忽視細節；**ESLint** 和 **Prettier** 等工具依然至關重要，用於確保 AI 生成的代碼整齊、美觀且不具備曖昧出錯的寫法。
- **解決技術衝突**：AI 也能協助處理不同第三方套件之間的語法不一致或語法版本衝突問題。

2. 測試自動化與防禦機制

AI 大幅減輕了撰寫測試腳本的負擔，這在過去是非常耗時的工作：

- **單元測試與整合測試**：開發者可以請 **Cursor** 在每個里程碑（Milestone）協助編寫 **Jest** 測試檔。
- **回歸測試（Regression Test）**：一個良好的習慣是要求 AI 在新增功能後自動跑回歸測試，確保舊有的功能不會因為新代碼的加入而壞掉。
- **端對端測試**：對於複雜的 **Playwright** 模擬操作（如 Google 搜尋、頁面互動），同樣可以請 AI 輔助撰寫測試腳本。

3. CI/CD、監控與文書處理

- **自動化流程**：AI 能協助建立 **CI/CD（持續整合與部署）** 的流程與腳本，節省手動設定的時間。
- **工具整合指導**：當開發者需要安裝如 **Sentry** 或 **Datadog** 等 APM 工具時，AI 能夠根據最新的 Library 提供安裝與設定指南（Guidance），雖然有時需注意其資訊是否與最新版本界面對齊。
- **文件撰寫**：傳統上開發者不喜歡寫文件（Documentation），現在可以利用大語言模型（LLM）將這部分工作自動化。

4. 跨領域整合：從設計到管理

- **設計與代碼對接**：來源提到，開發者可以使用 **Figma** 進行設計，再透過 **MCP (Model Context Protocol)** 等技術，將設計意圖低階地餵給 AI 來產生前端程式碼。
- **智慧化專案管理**：新興工具如 **Linear** 嘗試導入 AI 監測開發進度，當代碼更新時，AI 會自動更新監控狀態，甚至扮演「24 小時智慧 PM」的角色。

5. 對職業與角色的影響

- **職能模糊化**：隨著 AI 工具的普及，工程師、設計師、PM 之間的界線將變得模糊。一個具備 AI 工具能力的開發者，可能同時具備 80 分以上的設計能力與管理能力。

- **轉型建議**：來源認為 AI 不會讓人類沒工作，但會取代舊有的工作形態。開發者應主動學習**專案管理、產品設計及使用者服務設計**，將自己轉向多面手。

AI 輔助開發就像是為開發者配備了一個「外骨骼動力裝甲」：你不再需要自己搬運沉重的程式碼磚塊（手動打字、調格式），裝甲會代勞。你的任務轉向了指揮裝甲的移動方向（系統架構設計）、確保裝甲的感應器正常運作（測試與監控），以及決定這台機器最終要服務於什麼樣的目標。

試比較 ESLint 與 Prettier 的主要功能差異。

課程中推薦哪兩款前端端對端（E2E）測試工具？

什麼是 UAT，其主要測試執行者是誰？