

[https://youtu.be/cWJs\\_\\_opmlk?list=TLGGGeeldyYU14k4yNjEyMjAyNQ](https://youtu.be/cWJs__opmlk?list=TLGGGeeldyYU14k4yNjEyMjAyNQ)

[https://youtu.be/8eRkEb7JbbY?list=PLIAzIZzCjtLJQzoK-1n6R\\_KVTXm4tHzqP](https://youtu.be/8eRkEb7JbbY?list=PLIAzIZzCjtLJQzoK-1n6R_KVTXm4tHzqP)

這份教學原始碼主要介紹 **Next.js** 作為全端框架的核心優勢，強調其能透過單一開發環境整合前後端，並利用 **伺服器端渲染 (SSR)** 提升搜尋引擎最佳化與載入效能。文中詳述了 Next.js 如何處理三種渲染模式的混合應用，包括即時互動所需的 **客戶端渲染 (CSR)** 與預先建置的 **靜態網站生成 (SSG)**，讓開發者能根據功能需求彈性配置。除了技術架構，作者也深入探討了 **AI 輔助程式設計 (AI Coding)** 的必然趨勢，認為理解底層技術與架構概念是引導 AI 產出高品質代碼的關鍵。最後，教學轉向實際應用場景，透過實作如 **OAuth 第三方授權登錄** 與多人協作待辦事項清單，展示如何在現代開發流程中達成快速交付與高效專案管理。

這份教學原始碼詳盡解析了 **Next.js 框架** 的實作核心，從基礎的專案初始化到複雜的 **多用戶代辦事項 App** 開發。課程首先強調 **App Router** 檔案系統與路徑的關聯，深入說明 **Layout**（框架定義）與 **Page**（頁面內容）的協作機制，並示範如何透過 `use client` 告訴將 **伺服器組件** 轉換為具備互動功能的 **客戶端組件**。技術核心涵蓋了利用 **OAuth 2.0**（如 Google、GitHub 及 Facebook）整合第三方登入驗證，以及透過 **Prisma** 管理後端資料庫模型。最後，作者強調了現代軟體開發中 **CICD 自動化測試** 與持續部署的重要性，鼓勵學生藉由積極經營 GitHub 專案來累積實戰資歷。

Discuss Next.js 全端框架。

**Next.js** 是一個基於 **React** 的全端 (**Full-stack**) 框架，由 **Vercel** 公司開發與維護。它的核心使命是讓開發者能使用同一套 React 技術，同時處理前端、後端與 API 開發，並提供「開箱即用」的配置，讓開發者能專注於應用程式本身，而非繁瑣的環境設定。

以下是根據來源對 Next.js 全端框架的深入探討：

## 1. 為什麼選擇 Next.js（與傳統架構的差異）

在傳統開發中，前端通常使用 React，後端則使用 Node.js（如 Express），這需要分別管理與部署兩個服務。然而，**Next.js 將前後端整合在一個框架中**，這帶來了幾個優勢：

- **簡化部署**：不再需要為前後端開啟不同的端口（如 3000 與 5000）或管理多個部署流程，整個應用程式可以統一部署在如 Vercel 等雲端平台上。
- **語言一致性**：前後端皆使用 TypeScript 或 JavaScript，開發者不需在不同框架間切換思維。
- **開發效率**：隨著 AI Coding（如 Cursor 或 Claude）的興起，使用 Next.js 這類高度整合的框架能讓開發者更快速地從創意走向線上體驗。

## 2. 核心渲染機制

Next.js 最強大的地方在於它提供了混合渲染 (**Hybrid Rendering**) 的能力，開發者可以根據頁面需求選擇最合適的模式：

- **伺服器端渲染 (SSR, Server-Side Rendering)**：伺服器會動態產生 HTML 並直接吐回前端，這對於 **SEO**（搜尋引擎最佳化）非常有利，因為搜尋引擎可以爬取到完整的網頁內容，而非空殼。

- **客戶端渲染 (CSR, Client-Side Rendering)**：適合互動性強的元件（如按鈕、切換開關）。在 Next.js 中，需在檔案頂部標註 "use client" 來啟用此模式，以管理本地狀態（Local State）。
- **靜態網站生成 (SSG, Static Site Generation)**：在編譯階段（Compile time）就預先生成 HTML 檔，適合內容不常變動的頁面（如產品說明、文件），其效能最佳且成本最低。

### 3. 關鍵技術特色

- **App Routing (檔案系統路由)**：從 13 版開始引入。路由不再需要透過程式碼定義，而是與**檔案夾結構**一致。例如，`app/about/page.tsx` 會自動對應到 `/about` 網址。
- **內建最佳化**：包含圖像優化（Image Optimization）、腳本優化以及極致的**緩存 (Caching)** 機制，能將頻繁存取的資料放在記憶體中，提升反應速度。
- **原生支援 Tailwind CSS**：相較於提供現成組件的 Material UI，Next.js 傾向與 Tailwind CSS 結合，提供開發者更高設計彈性的樣式工具。
- **身份驗證與資料庫**：在全端開發中，常搭配 **Auth.js (NextAuth)** 進行第三方授權登錄（如 Google, GitHub, Facebook），並結合 **Prisma ORM** 來操作數據庫。

### 4. 實際應用案例

Next.js 非常適合開發具備即時互動性的社群服務（如 X 或 Threads 的克隆版）。來源提到一個**多用戶待辦事項清單**的案例：

- 利用 Next.js 處理前後端邏輯。
- 結合 **Pusher** 技術實現主動廣播（Broadcast），讓多個用戶能即時同步工作狀態。
- 透過後台綁定帳號，實現無密碼的 OAuth 安全登錄。

---

💡 **比喻理解**：如果開發一個網頁就像**裝修房子**，傳統做法是你要先去木材行買原木（React），再去五金行買工具（Node.js），最後自己想辦法把兩邊湊在一起。而 **Next.js** 就像是一間「**特力屋**」：它幫你切割好了木材並磨光（Tailwind CSS），還提供了一套完整的組裝說明與工具箱，讓你可以根據需求，決定哪些家具要直接買現成的（SSR），哪些要自己動手組裝（CSR），大幅縮短了從空地到完工入住的時間。

Discuss 渲染模式比較。

根據提供的來源，Next.js 作為一個全端框架，其強大之處在於提供了**混合渲染 (Hybrid Rendering)** 的能力，允許開發者在同一個應用程式中根據不同需求選擇最合適的模式。

以下是針對 **SSR**、**CSR** 與 **SSG** 三種渲染模式的詳細比較與討論：

1. **伺服器端渲染 (SSR, Server-Side Rendering)**
- **運作機制**：伺服器在接收到請求時，會動態地將資料打包成完整的 HTML 檔案並直接吐回給瀏覽器。
  - **優點**：

- **SEO (搜尋引擎最佳化) 極佳**：搜尋引擎爬蟲可以直接抓取到完整的網頁內容，而非只有前端的一個「殼子」。
- **載入速度快**：使用者能更迅速地看到渲染好的內容，不需等待前端下載大量 JavaScript 後才開始繪製。
- **適用情境**：適合需要即時資料且對 SEO 有高度要求的頁面，例如社群動態、數據儀表板、購物車等。

## 2. 客戶端渲染 (CSR, Client-Side Rendering)

- **運作機制**：伺服器僅傳送最小限度的 HTML，由瀏覽器下載並執行 JavaScript 來渲染內容。在 Next.js 中，必須在檔案頂部標註 "use client" 才能啟用此模式，因為其預設為伺服器端渲染。

- **優點**：

- **互動性強**：適合處理複雜的 UI 互動，如開關切換或表單輸入，這些動作若每次都跑回後端會顯得過於緩慢且無效率。

- **狀態管理**：方便使用 **React Hooks** 來管理本地狀態 (Local State)。

- **缺點**：**SEO 較差**，且首次載入頁面時需要較多時間下載與執行腳本。

- **適用情境**：適合**互動性元件**，或是使用者登入後不需被搜尋引擎爬取的私有操作介面。

## 3. 靜態網站生成 (SSG, Static Site Generation)

- **運作機制**：在編譯階段 (**Compile Time**) 就預先生成好 HTML 檔案。一旦部署，伺服器就會直接提供這些靜態內容。

- **優點**：

- **效能最佳**：不需在執行時動態運算，反應速度最快。

- **成本最低**：對伺服器資源的消耗最少。

- **適用情境**：適合**內容不常變動**的頁面，例如**產品說明頁、文件網站、部落格文章**。

## 渲染模式比較總結表

特性	SSR (伺服器端渲染)	CSR (客戶端渲染)	SSG (靜態網站生成)
<b>生成時間</b>	請求時 (Run Time)	瀏覽器執行時 (Run Time)	建置時 (Compile Time)
<b>SEO 效果</b>	非常好	較差	非常好
<b>首屏載入</b>	快	較慢	最快
<b>互動彈性</b>	較低	最高	較低
<b>主要用途</b>	即時、動態、需 SEO	高互動、管理狀態	穩定內容、高效能

💡 **比喻理解**：

- **SSG** 就像「便利商店的御飯糰」：在工廠（編譯時）就已經做好了，使用者拿了就能馬上吃，速度最快、成本最低，但不適合客製化。
- **SSR** 就像「現做便當店」：你點餐後，廚房（伺服器）才開始根據目前的菜單（最新資料）幫你裝盒，雖然要等一下，但保證拿到的是最新的，且大家都能一眼看出裡面賣什麼（利於 SEO）。
- **CSR** 就像「自助火鍋」：店員只給你鍋子和生食材（HTML/JS 框架），你得自己在桌上煮熟（瀏覽器渲染）才能吃，雖然開動前準備較久，但吃法最靈活、互動感最強。

Discuss 第三方授權登錄。

在全端開發中，**第三方授權登錄（OAuth）** 已經成為現代網路服務的主流身分驗證方式。根據來源，這種機制讓使用者無需為每個新服務重新建立帳號密碼，而是透過現有的平台帳號（如 Google、GitHub、Facebook）進行授權與登錄。

以下是針對第三方授權登錄的詳細討論：

### 1. 核心概念與優勢

第三方授權登錄的核心在於**簡化流程與提升安全性**：

- **免去傳統帳密管理**：使用者不再需要記憶無數組帳號密碼，減少了因密碼強度不足或重複使用而產生的安全風險。
- **降低開發成本**：開發者不需要自行維護複雜的密碼加密存儲系統或重設密碼的邏輯，而是將驗證過程委託給專業的大型平台（如 Google 或 Facebook）。
- **提升信任感**：使用知名平台的授權機制，能增加使用者對新應用程式的信任。

### 2. Next.js 中的實作工具：Auth.js (NextAuth)

在 Next.js 框架中，處理第三方授權登錄最知名的套件是 **Auth.js**（原名為 **NextAuth**）。

- 它提供了一套完整的 API 與前端元件，讓開發者能快速整合多個授權供應商。
- 對於全端應用程式來說，身分驗證是帳號管理中極其重要的一環，尤其當應用程式需要部署到雲端供多人使用時。

### 3. 實務運作邏輯：綁定與授權

來源中提到了一個有趣的應用場景，說明了在小規模團隊或特定專案中，如何運用第三方授權來管理成員：

- **後台綁定（Binding）**：為了安全起見，系統可以不開放自由註冊，而是由管理員在後台先將特定的第三方帳號（例如某個 Google 帳號）與系統身分進行「綁定」。
- **登錄流程**：使用者在前端點擊「Google 登錄」後，系統會跳轉至 Google 的授權頁面。一旦使用者選擇了正確且已綁定的帳號，系統便會確認身分並建立 Session（會話），讓使用者成功登錄。
- **多身分登錄**：同一個系統可以支援多種供應商，例如 GitHub、Google 與 Facebook。使用者可以使用不同的第三方身分來登錄同一個應用程式，例如在示範案例中，開發者可以切換不同帳號來操作同一個多用戶待辦清單。

#### 4. 及時同步與工作流

在全端服務中，第三方授權登錄常與資料庫（如 PostgreSQL）及 ORM 工具（如 Prisma）結合，用來追蹤是「誰」執行了哪些操作。

- 例如在**多用戶合作專案**中，透過第三方授權確認身分後，每位成員所新增的任務或標記（Tag），都能即時同步給其他成員。
- 這種機制不需要繁瑣的溝通成本（如頻繁開會或傳通訊軟體），只要登錄後，所有狀態更新都能在雲端平台上清楚呈現。

---

**💡 比喻理解：**第三方授權登錄就像是使用「**飯店感應卡**」來進出房間。你不需要自己去打一把實體鑰匙（註冊新帳密），而是透過一個你已經信任的發卡機構（如 Google 櫃檯）。只要你證明你是感應卡的持有人（授權成功），飯店大門與房間（應用程式）就會為你開啟。對你來說更方便，對飯店來說，也省去了維護複雜鎖頭的麻煩。

Discuss AI 輔助編程.

根據來源內容，**AI 輔助編程（AI Coding）** 正處於一個從傳統手寫代碼轉向新時代的**重要過渡時期**。這不僅僅是工具的演進，更是整個軟體開發範式的轉變。

以下是針對 AI 輔助編程的深入討論：

##### 1. 現狀：從非 AI 到 AI 的過渡期

目前的開發環境正處於一個轉型階段，雖然「完全不需要寫代碼」（No-code）的境界尚未完全成熟，但 AI 輔助開發已成為主流。

- **進入門檻降低：**即使是非資訊背景、不會寫程式的人，也能利用如 **Cursor** 等 AI 工具，在不完全理解底層技術（如 Next.js 或 Prisma）的情況下，獨立完成部落格等專案開發。
- **開發效率極大化：**AI 能顯著減少手寫代碼的時間，並自動處理許多因打錯字或瑣碎邏輯產生的 **Bug**。對開發者來說，這種體驗被描述為「太香了」、「太舒服了」，是一條**不可逆轉的道路**。

##### 2. 為什麼仍然需要了解底層技術？

儘管 AI 非常強大，但來源強調，作為 AI 編程的先驅，開發者仍不能完全放棄對技術原理的學習。

- **處理複雜性：**當專案變得複雜或 AI 的**上下文視窗（Context Window）** 不足時，AI 可能會給出錯誤建議，若缺乏基礎知識，開發者將很難解決問題，甚至會在原地繞圈子。
- **提供專業引導（Guidance）：**目前 AI 編程的最佳實踐（Best Practice）是：**由人類提供專業的技術引導，讓 AI 執行具體撰寫**，這樣產出的代碼 Bug 會更少、品質更高。
- **技術天花板的擔憂：**現有的 AI 是學習人類過去十年的技術結晶（如各種框架）而成。如果未來沒人研究底層技術，軟體技術的進步可能會遇到天花板。

##### 3. 歷史的必然性：與文字處理器的類比

來源將 AI 輔助編程比作早期從「手寫報告」轉向「電腦文字處理」的過程,：

- 在過去，寫錯一個字可能要用橡皮擦擦掉一堆或重新排版；而 Word 或 TeX 的出現，雖然早期工具並不完美，但其帶來的便利性讓人類再也回不去手寫時代。
- 同樣地，現代開發者只要體驗過 AI 幫忙修 Bug、寫重複性代碼的效率，就不可能再回到純手寫的模式,。

#### 4. 對未來的影響與態度

- **社會實驗**：現在的開發者正在參與一場大型的社會實驗，探索 AI 編程的最佳路徑。
- **不可拒絕的趨勢**：軟體工程師不應成為「拒絕 AI」的人，否則將無法跟上時代進步的速度。未來，代碼可能不再由人類親自撰寫，但理解觀念與技術依然是確保軟體品質的關鍵,。

---

💡 **比喻理解**：AI 輔助編程就像是從「手動駕駛」進化到「自動導航輔助」。雖然導航和自動駕駛能幫你處理大部分枯燥的路徑（寫重複代碼、修語法錯誤），但你依然需要是一位懂得交通規則的駕駛員。當遇到複雜的路況（系統架構出錯）或導航失靈（AI 邏輯混亂）時，只有了解車子性能和道路邏輯的人，才能及時接手方向盤，確保車子平安到達目的地。

Discuss 專案管理實務.

根據來源內容，**專案管理（Project Management）** 是一門深奧的學問，但其執行方式必須根據團隊規模與開發速度進行彈性調整，否則管理成本反而會拖累開發進度。

以下是針對專案管理實務的詳細討論：

##### 1. 規模與管理成本（Overhead）的權衡

- **大型團隊**：通常需要專業的**專案經理（PM）**，負責將客戶需求轉化為產品需求文件（PRD），再拆解成細節任務與工程師討論。
- **小型團隊（1-3 人）**：來源強調，對於極小型團隊來說，傳統的專案管理流程往往 **Overhead（額外開銷）太大**。如果溝通成本大於開發速度，則不適合採用複雜的流程。在程式碼撰寫極其快速的環境下，一週開一次會可能已經累積了上百項變動，導致資訊無法同步。

##### 2. 輕量化的任務管理實務

針對快速迭代的小型專案，來源建議採用一種「乾淨且即時」的任務清單模式：

- **即時同步**：不需要複雜的分類或截止日期（Deadline），因為目標通常是「今天或明天就要做完」的事項。
- **標籤化管理（Tagging）**：利用標籤來區分負責人或專案類別。例如，標註特定負責人的姓名標籤，透過 **Filter（過濾）** 功能即可快速查看個人任務，不需將不同人的事項強行拆分在不同系統中,。
- **有效溝通工具**：使用 LINE 或 Messenger 等通訊軟體管理進度其實非常沒效率，必須有一個能管理「狀態（Status）」的地方,。

##### 3. 「斷捨離」的專案哲學

在軟體開發中，應區分「管理工具」與「開發工具」的職責：

- **版本控制與任務關聯**：所有的代碼變更應由 GitHub 負責版本控制。
- **任務清理**：來源提出一個觀點：**代碼一旦完成並 Checking 進入 GitHub，對應的任務（To-do item）就應該刪除**。不需要在管理工具中保留過時的歷史紀錄，若發現錯誤，重新開啟一個新任務即可。這種「斷捨離」能保持管理介面的精簡，符合極速開發的節奏。

#### 4. 團隊建設與營運建議

- **保持極簡（Minimal）**：早期開發團隊建議維持在 **10 人以下**。當感到忙不過來時，應先調整工作流程或提高效率，而非急著招人，因為過早擴張會導致燃燒資金的速度遠超獲利。
- **0 到 1 的焦點**：在創業初期（0 到 1 階段），**產品設計與技術開發**是最重要的，行銷（Marketing）等其他事項在初期並不關鍵。
- **權限管理**：專案管理也包含安全管理，例如利用第三方授權（OAuth）在後台手動綁定成員帳號，確保只有團隊內部的 3 到 4 個人可以登錄與操作系統，避免不必要的外部干擾。

---

💡 **比喻理解**：專案管理就像是「規劃一場旅行」。如果你是帶著 50 人的大旅行團（大型團隊），你必須有詳細的行程表、導遊與領隊（PM），每一步都要嚴謹確認。但如果你只是跟 2 個好朋友出遊（小型團隊），與其花時間寫幾十頁的計劃書，不如準備一份簡單的清單，列出今天想去的地方（標籤化任務），去過一個就劃掉一個（斷捨離），隨時根據當下的天氣與心情彈性調整，這樣反而能玩得更盡興、更有效率。

試比較 SSR、CSR 與 SSG 的差異與適用情境。

Next.js 框架的主要特色與開發使命為何？

傳統 HTTP 機制與 Pusher 技術在即時溝通上有何不同？

Discuss Next.js 路由機制。

Next.js 的路由機制（Routing）在 13 版之後引入了 **App Router**，這是一套基於**檔案系統（File-system based）**的路由架構，與傳統 React 開發需要手動定義路由表的方式完全不同。其核心理念是「目錄即路徑」，讓開發者透過檔案組織來直觀地管理網頁路徑。

以下是根據來源對 Next.js 路由機制的詳細分析：

##### 1. 檔案系統路徑 (File-based Routing)

Next.js 的路徑是由**資料夾結構**定義的。每一個資料夾代表一個**路徑段（Route Segment）**，多個段落組合起來即形成完整的**URL 路徑（Path）**。

- **路徑對應**：例如在 `src/app` 目錄下建立一個 `test-one` 資料夾，該路徑就會自動對應到 `domain.com/test-one`。
- **公開訪問限制**：一個目錄必須包含 `page.tsx`（定義 UI）或 `route.ts`（定義 API 邏輯）檔案，才能被外部訪問。如果目錄下沒有這些特定名稱的檔案，該路徑就無法在瀏覽器中開啟。

##### 2. 特殊檔案規範

Next.js 規定了幾種具有特殊功能的檔案名稱，用來處理不同的 UI 需求：

- **page.tsx**：用於定義該路由段的獨特 UI（即該頁面的核心內容），。
- **layout.tsx**：用於定義共享佈局（Shared UI），。佈局會包裹住子頁面（Page）或子佈局，且在路由切換時不會重新渲染（保持狀態），。
- **template.tsx**：與 layout 類似，但它在每次導航時都會重新初始化狀態\*\*，適合用於處理進退場動畫或統計數據追蹤，。
- **loading.tsx**、**error.tsx**、**not-found.tsx**：分別用於定義載入中、錯誤處理以及 404 頁面的 UI，增強使用者體驗，。

### 3. 進階路由模式

- **動態路由 (Dynamic Routes)**：當路徑包含變數（如使用者 ID 或文章標題）時，使用**方括號**命名資料夾，例如 `[id]`，。這樣 `blog/[id]` 就可以匹配 `blog/1` 或 `blog/abc` 等不同網址，。
- **路由組 (Route Groups)**：使用**小括號**命名的資料夾（如 `(marketing)`）可以用來組織檔案，但該名稱不會出現在 URL 路徑中。
- **API 路由 (API Routes)**：透過 `route.ts` 檔案，開發者可以定義 GET、POST 等後端 API 邏輯，讓前端能透過特定的 URL 存取資料（例如回傳 JSON 檔），而非整個 HTML 頁面，。

### 4. 路由與渲染的結合

Next.js 的路由機制與伺服器端渲染（SSR）高度整合。當使用者訪問特定路徑時，伺服器會根據該路徑的檔案架構，在後端將資料打造成 HTML 並吐回前端，這對於 **SEO (搜尋引擎最佳化)** 有極大幫助，因為爬蟲能抓取到完整的網頁內容。

---

💡 **比喻理解：** Next.js 的路由機制就像是一座「自動編號的辦公大樓」。

- 資料夾就是大樓裡的房間。你只要在門牌上掛上「行政區」（資料夾名稱），這條路徑就通了。
- **page.tsx** 是房間裡的內容（比如辦公桌）；**layout.tsx** 則是房間的公用設施（比如冷氣與電燈），不管你進出幾次，公用設施都在那裡，不需要重新安裝。
- 如果你想去不同的分公司（動態路由），只要在門牌掛上一個「變數標籤」`[分店名稱]`，不管是台北店還是台中店，都能自動導向同一個裝潢規格的辦公空間。

何謂 OAuth 及其驗證流程？

Next.js 中 server 與 client 元件有何差異？

軟體測試中 CI 與 CD 的核心概念為何？

Discuss Layout 與 Page.

在 Next.js 的 App Router 架構中，**Layout (佈局)** 與 **Page (頁面)** 是建構使用者介面最核心的兩個組成部分，它們共同定義了網頁的結構與內容。

以下根據來源對 Layout 與 Page 的詳細討論：

## 1. 核心定義與職責

- **Layout (佈局)**：負責定義網頁的框架（**Skeleton**）。它的主要作用是建立**共享 UI**，例如導航欄（Navbar）、側邊欄（Sidebar）或頁腳（Footer）。Layout 在路由切換時**不會重新渲染**，這有助於維持元件狀態並提升效能。
- **Page (頁面)**：負責定義該路由路徑對應的**獨特內容 (Meat)**。每個路徑段落（Route Segment）都必須包含一個 `page.tsx` 檔案，該路徑才能被外部公開訪問。

## 2. 運作機制：`{children}` 屬性

Layout 與 Page 之間透過 React 的 **children 屬性**進行協作。在 `layout.tsx` 中，開發者會定義一個佔位符（placeholder），通常寫作 `{children}`。

- 當使用者訪問某個路徑時，Next.js 會將對應的 `page.tsx` 內容放入 Layout 的 `{children}` 位置中顯示。
- **根佈局 (Root Layout)**：位於 `app` 目錄最頂層的 `layout.tsx` 是必須的，它定義了整個應用的 `<html>` 和 `<body>` 標籤。

## 3. 嵌套結構 (Nesting)

Next.js 支援**嵌套路由與佈局**。這意味著一個子目錄可以擁有自己的 `layout.tsx` 和 `page.tsx`。

- **層層包裹**：子路徑的 Layout 會被父路徑的 Layout 所包裹。例如，在 `test-one` 目錄下的 Layout 會嵌套在根目錄的 Layout 內，而 `test-one` 的 Page 則會顯示在自己的 Layout 之中。
- **範例**：根佈局可以包含全站通用的主選單（Main Menu），而特定功能的子佈局則可以定義該功能專用的次級選單或標題。

## 4. 渲染特性與開發規範

- **預設為伺服器元件**：在 Next.js 中，Layout 和 Page 預設都是**伺服器端渲染 (Server Components)**。如果需要在頁面中加入互動邏輯（如按鈕點擊事件），必須在檔案頂部標註 `"use client"` 將其轉為客戶端元件。
- **特殊檔案順序**：在渲染層次上，Layout 會包裹住其他特殊檔案，如 `template`、`error boundary`、`suspense`（`loading`），最後才是 `page`。
- **命名限制**：框架對這些檔案名稱有嚴格規定（Convention over Configuration），必須命名為 `layout.tsx` 或 `page.tsx` 才能發揮預期功能。

---

💡 **比喻理解**：Layout 就像是「房子的格局」，規定了哪裡是客廳、哪裡是走廊，不論誰住進來（換到哪個頁面），這些基礎結構都不會變動。而 Page 就像是「房間裡的房客」，每個房間（路由）可以住不同的房客，提供完全不同的生活內容（UI 資訊），但他們都必須遵守房子的整體動線與配置。

試比較 Server-side Rendering (SSR) 與 Client-side Rendering (CSR) 的差異。

請問 Next.js 13 版本後採用的 App Routing 機制為何？

試述在軟體開發流程中，實施 CICD 與自動化測試的重要性。

Discuss Server 與 Client.

在現代 Web 開發中，**伺服器（Server）** 與**客戶端（Client）** 的角色與互動方式正在發生重大演變。根據來源，Next.js 這類全端框架的出現，打破了傳統前後端完全分離的疆界，以下從多個面向討論兩者的關係：

### 1. 從「分立」走向「全端整合」

- **傳統模式**：以往開發者通常將前端（如 React）與後端（如 Node.js/Express）視為兩個獨立的專案，甚至需要分別部署在不同的埠號（例如前端 3000，後端 5000）。
- **全端一體化**：Next.js 的核心使命是讓開發者使用**同一套 React 技術**同時編寫前後端代碼。在部署時，不再需要分開管理多個進程，而是整合成一個服務，大幅降低了環境配置與溝通的複雜度。

### 2. 渲染職責的重新分配

來源深入探討了三種主要渲染模式，決定了 Server 與 Client 各自負責的工作量：

- **伺服器端渲染 (SSR)**：Server 在接收請求時，會根據資料動態生成完整的 HTML 檔案並吐回 Client。這對 **SEO (搜尋引擎最佳化)** 至關重要，因為搜尋引擎爬蟲能直接讀取完整網頁，而非僅是一個「殼子」。此外，Server 還能利用**快取機制 (Caching)**，將頻繁存取的資料存放在記憶體中，加速後續請求的反應速度。
- **客戶端渲染 (CSR)**：Server 只傳送最小限度的 HTML，剩下的渲染邏輯交由瀏覽器（Client）執行 JavaScript 來完成。在 Next.js 中，為了與 Server Side 區隔，開發者必須在檔案頂部標註 "**use client**" 才能使用 React Hook 等互動功能。
- **混合渲染 (Hybrid)**：這是 Next.js 的強項，開發者可以靈活決定：與資料獲取相關的邏輯放在 Server，而需要即時互動（如按鈕點擊、本地狀態管理）的部分放在 Client。

### 3. 資料互動與通訊機制

- **傳統 API 呼叫**：Client 透過 HTTP (如 GET 請求) 向 Server 獲取 JSON 資料，再重新渲染畫面。Next.js 提供最佳化過的 `NextRequest` 與 `NextResponse` 來處理這些通訊。
- **及時雙向溝通**：對於像 X (Twitter) 這種需要即時互動的服務，傳統的被動收訊機制 (HTTP) 不足，需要透過如 **Pusher** 等技術實現 **廣播 (Broadcast)**，讓 Server 主動將新訊息推送到所有已訂閱的 Client 身上。

### 4. 身分驗證與狀態維持

- **Session (會話) 管理**：為了維持使用者的登入狀態，Server 與 Client 之間需要 **Session** 機制。當使用者透過第三方（如 Google, GitHub）授權登入後，Server 會建立一個會話，讓使用者下次打開瀏覽器時不需要重複登入，兼顧了便利性與安全性。
- **身分核實**：在全端架構下，Client 不能直接操作資料庫。所有的動作都必須透過 API 到 Server 端進行身分檢驗，確保只有授權使用者能存取資料。

 **比喻理解：**Server 與 Client 的關係就像是「高級餐廳的廚房與餐桌」。

- **SSR (伺服器渲染)** 就像是廚師在廚房把菜煮好、擺盤完美（生成 HTML）後才端上桌，客人（使用者與爬蟲）一看到就是完整的佳餚。
- **CSR (客戶端渲染)** 則是廚房只端出半成品和食譜，讓客人在餐桌上自己動手煮熟。雖然客人可以根據口味即時調整（高互動性），但開動前的等待與準備時間較長。
- **Next.js** 則是提供了完美的**外場經理**，會根據每道菜的特性，決定哪些在廚房處理好再端出來，哪些讓客人在桌上享受DIY的樂趣。

請簡述 Next.js 框架中 SSR 與 CSR 的差異。

什麼是 OAuth？其授權登錄的基本流程為何？

軟體測試中的單元測試與整合測試有何不同？

Discuss OAuth 身份認證。

**OAuth (Open Authorization, 開放授權)** 是一種開放標準的授權協議，旨在讓使用者能夠授權第三方應用程式存取其在其他服務（如 Google、GitHub、Facebook）上的資訊，而無需提供其原始帳號密碼。

以下根據來源討論 OAuth 身份認證的核心機制與實務應用：

### 1. OAuth 的核心機制與優勢

OAuth 的主要目的在於**槓桿化 (Leverage) 現有的服務**，解決傳統開發中需要自行維護帳號密碼的麻煩。

- **授權流程**：當使用者點擊登錄時，網址會導向服務提供商（Provider）的授權頁面。授權成功後，網誌會從提供商那裡拿到一個 **Access Token**（存取權杖）。
- **臨時通行證**：網站使用這個 Token 來獲取使用者的基本資料（如姓名、Email）。這個 Token 就如同一個臨時通行證，代表該應用程式已被授權存取使用者的特定資訊。
- **安全性**：使用者不需要在每個服務都設定一組密碼，不僅方便，也降低了密碼遭竊的風險。

### 2. 在 Next.js 中的實作工具

在 Next.js 框架中，開發者通常會使用一個名為 **Auth.js**（原名 **NextAuth**）的套件來快速建構授權機制。它支援多種供應商的登錄，讓全端應用程式的帳號管理變得更加簡單。

### 3. 不同平台（Providers）的設置細節

根據來源，整合不同平台的 OAuth 流程略有差異，但都需要獲取 **Client ID** 與 **Client Secret**：

- **Google**：需要在 Google Cloud Console 建立專案並開啟 OAuth 授權。來源提醒，Google 的文件有時較為混亂（例如某些舊文件會提到已停止服務的 Google Plus），開發者需仔細核對。
- **GitHub**：設定相對直觀，在開發者設定（Developer settings）中建立 OAuth App 即可獲得憑證。
- **Facebook**：機制較為複雜，因為它將開發者區分為一般使用者與店家。開發者必須建立一個「Facebook 應用程式（App）」來扮演中介機器人的角色，藉此獲取使用者資料。

#### 4. 實務設計與權限管理

在專業的開發專案中，OAuth 不僅用於登錄，還用於精確的權限控制：

- **帳號綁定（Binding）：**系統可以設計為「**不開放註冊**」，而是由管理員在後台手動將特定第三方帳號（如某人的 Google 帳號）進行「綁定」。只有被綁定的帳號在授權後才能進入系統，確保安全性。
- **環境變數保護：**獲取的 Client Secret 極其重要，通常只能查看一次。開發者必須將其存放在 .env 檔案中，並額外產生一個 NEXTAUTH\_SECRET（可用 OpenSSL 指令生成）來加密 Session 資訊。

#### 5. Session（會話）管理

OAuth 登錄成功後，系統會建立 **Session** 來維持登錄狀態。

- **提升體驗：**使用者下次開啟瀏覽器時不需重新登錄。
- **裝置安全性：**Session 通常會綁定特定的瀏覽器與裝置，且設有有效期限（Expiration time），若長時間未操作或在不同裝置登錄，則需重新驗證。

---

💡 **比喻理解：OAuth 就像是「社區門禁卡」。**你不需要為社區內的所有設施（如健身房、游泳池、閱讀室）都各自打造一把鑰匙（註冊新帳號）。你只需要拿著你受信任的身份證明（如 Google 帳號）去物業管理處（OAuth 授權頁面）蓋個章，物業就會給你一張臨時的感應卡（Access Token）。這張卡只能在特定時間內進入你被允許的場所，且隨時可以被註銷，既安全又方便。

何謂 Next.js 中的伺服器端渲染 (SSR)？

請比較 Client Component 與 Server Component 的使用時機。

在開發團隊中，實行 CICD 流程的主要目的為何？