PiTube:
從入門到棄用

計算機網路
專案作品說明

關於本地安裝方式，請見 README.md 中說明。

目次

作品介紹 **2**

已完成的評分項目 2

功能說明 **3**

網站介面 3

登入/登出 3

個人頁面 4

影片串流 4

音訊串流 4

運作原理 **5**

各個 Handler 5

特別發現 6

風格參考 6

部署狀況 **7**

網路配置 7

作品介紹

PiTube (ㄗㄨㄥTube，即資工Tube)(我知道這個梗很爛……)，是一個集視訊、音訊、留言板等眾多功能於一身的網路平台。

本作品已於 GitHub 上公開 (https://github.com/kinoras/NTU-CN23)。

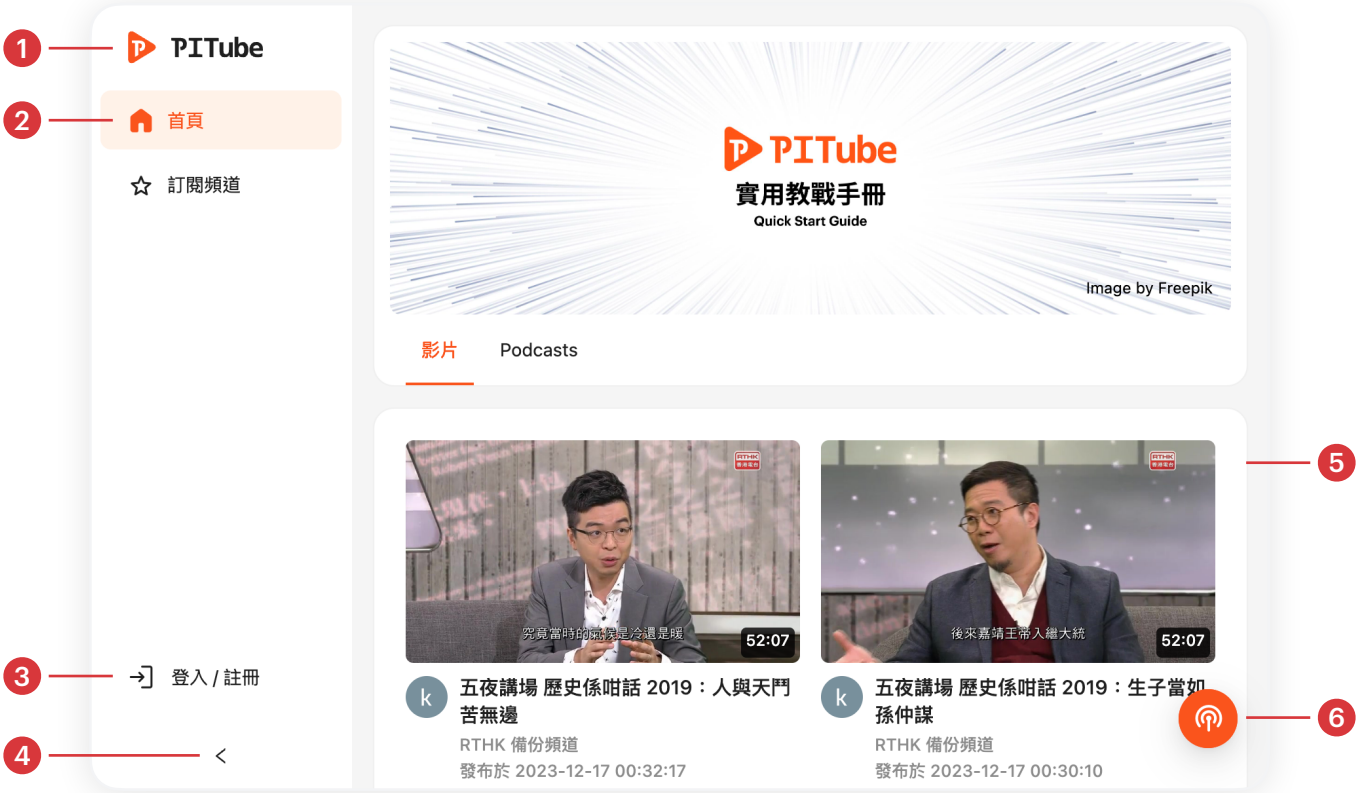
已完成的評分項目

- **Socket 傳文字** — 所有檔案、API 請求/回應都是以 socket 傳送 (node.js net/tls 套件)。
- **Profile 頁面** — 以頻道方式呈現，可顯示姓名、學號、自介及上傳的影片與 podcast。
- **留言板** — 影片下方設有留言區，已登入使用者可發布或刪除留言。
- **登入註冊** — 支援 Google 帳號登入。使用者憑證使用 JWT 實作，儲存在 localStorage 。
- **聲音串流** — 支援 mp3 格式的音樂/podcast 串流。
- **影片串流** — 支援 HLS 影片串流。
- **額外功能**
  - **HTTPS** — 可使用第三方憑證或透過 OpenSSL 自行簽署 (自簽者仍會標示「不安全」)。
  - **HTTP Cache** — 透過相關 header 令檔案 (含前端網頁) 能支援快取，降低載入時間。
  - **介面及功能** — 網站外觀簡潔，並提供淺/深色模式切換、訂閱等功能 (登入後可使用)。
  - **模組化 & 擴充性** — 前後端皆採模組化開發，日後擴充只需新增相應的 handler 即可。

功能說明

在介面和使用方式上 PiTube 與 YouTube 相近，使用者可透過熟悉的操作方式使用 PiTube。

網站介面



圖例 —

1 首頁按鈕　2 導覽列　3 登入按鈕 (未登入) / 個人資料按鈕 (已登入)　4 側邊欄縮放按鈕

5 內容　6 Podcast 播放器顯示按鈕 (紅色 = 播放中；白色 = 暫停播放；未顯示 = 未播放任何音訊)

登入/登出

登入 — 未登入狀態下，按左下方登入按鈕即可登入。

登出 — 已登入狀態下，按左下方個人資訊以顯示個人選單，並按下登出按鈕，即可登出。

註：如果選用系上帳號 (@csie.ntu.edu.tw) 登入，使用者代號會是學號 (或系上帳號)；
如果使用其他 Google 帳戶登入，則會獲發隨機的使用者代號。

個人頁面

查看 — 透過個人選單，可以查看自己的個人頁面。點擊影片列表、Podcast 列表、影片資訊欄的頻道名稱，可以查看創作者的頻道主頁。點擊留言者的名稱，可查看該人的頻道。

修改 — 按下自己個人頁面的「編輯個人檔案」按鈕，可修改自己的頻道名稱及介紹。

訂閱 — 按下其他人個人頁面的「訂閱/已訂閱」按鈕，即可訂閱/取消訂閱該頻道。

影片串流

觀看影片 — 在影片列表、個人頁面中的影片可瀏覽影片。

留言 — 已登入使用者可在影片下方留言區發表留言/刪除自己過往的留言。

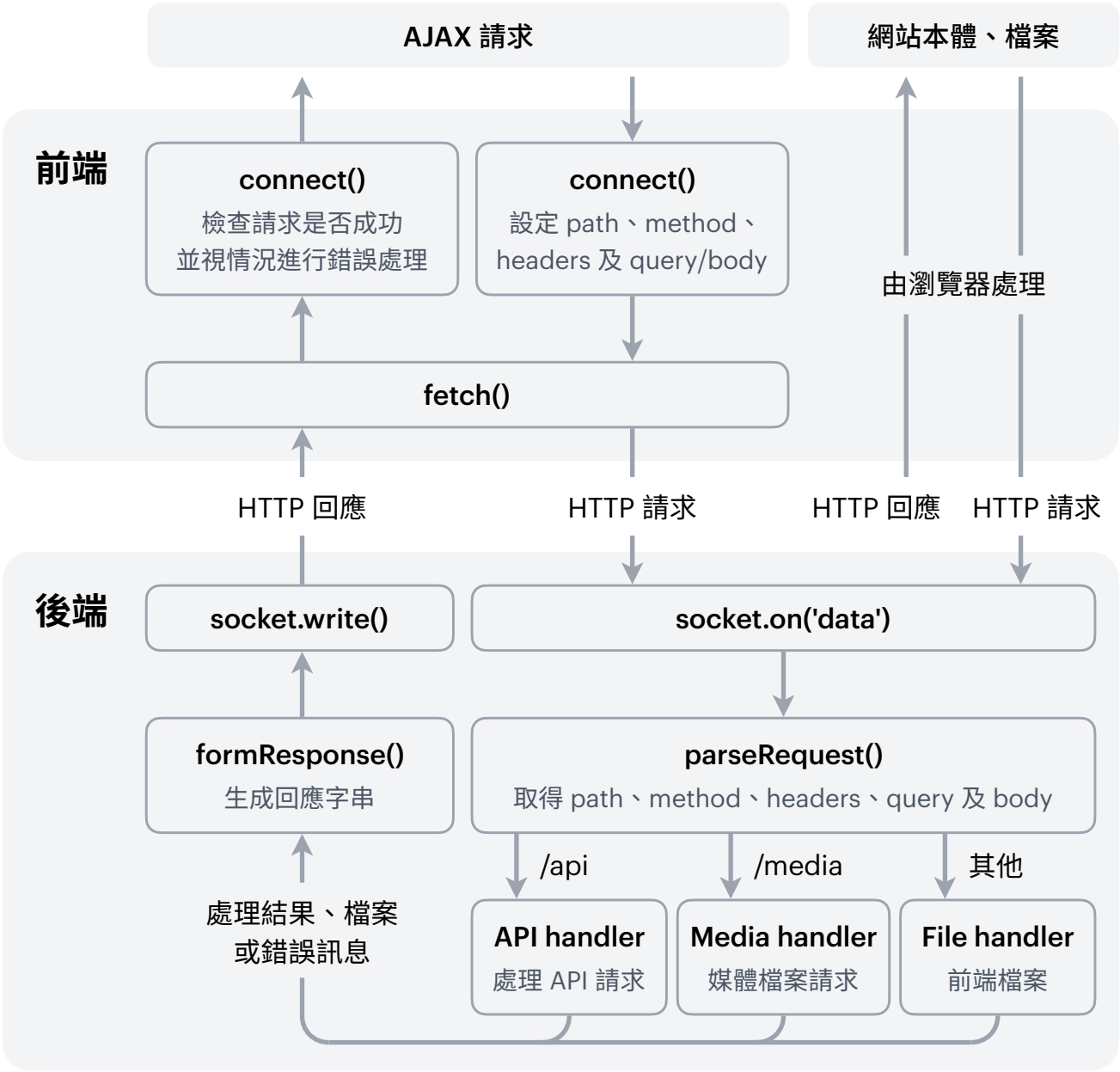
音訊串流

收聽 Podcast — 在 Podcast 列表、個人頁面中的播放按鈕即可播放。

控制播放 — 點擊視窗右下方之播放器按鈕，可開啟播放器浮窗並控制播放。

註：音訊開始播放時，同一視窗的影片會自動暫停；反之亦然。

運作原理



註：除 socket.on()、socket.write() 及 fetch() 為內建函數外，其他均為自訂函數。

各個 Handler

API Handler 處理所有 path 以 /api 開頭的請求。它會按照 path 和 method 決定是否需要驗證使用者身分，並把 query/body 等資訊傳給對應處理函數，最後取得處理結果並回傳。

Media Handler 處理所有 path 以 /media 開頭的請求。它按照 path 獲取檔案，並檢查檔案最後修改日期是否比 If-Modified-Since header 來得早。如果是，則回傳 status 304；如果不是，或沒有該 header，則回傳整個檔案。

File Handler 處理所有其他請求，方式與 Media Handler 相似，但獲取 `/frontend/build` 目錄下的檔案 (打包好的前端)。這種方式能讓前後端使用相同的 port，避免 CORS 問題。

特別發現

HTTPS 下訊息分步送達 — 我們觀察到在 HTTPS 下，當 request body 較大時，請求可能會被分成多部分送達。因此，server 在接收資料時必須藉 Content-Length header 判斷 body 內容是否已到齊。

Content-Length 與漢字 — 承上，在編碼時一個漢字被視為 3 個字元，而計算字串長度的函數會把一個漢字算作為 1 單位 (與字母相同)，因此須使用 Buffer.byteLength(body, 'utf8')) 來獲取 body 的長度，而非 body.length。

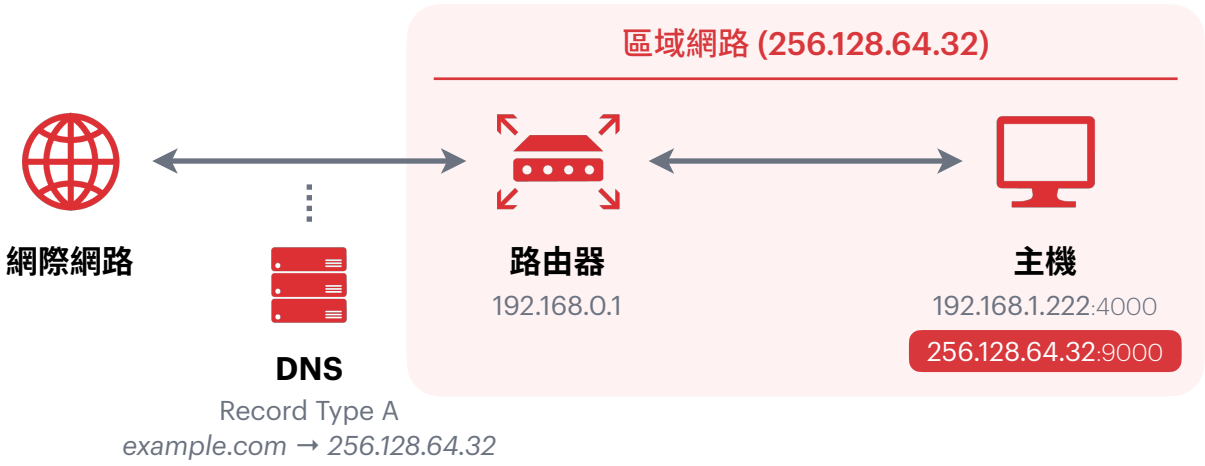
文字與檔案須分開寫入 socket — 回傳檔案時，必須把 header (字串) 及內容 (位元組流) 分兩次寫入 socket，不能把它們結合成單一字串回傳，否則會令檔案編碼出錯而無法開啟。

風格參考

在實作過程中，我們大量參考了 axios 及 Express 等套件/中介軟體的風格。這讓程式設計師不必關注底層的實作方式，並用跟這些套件差不多的方式編寫應用程式，使用上更加熟悉和自然，也符合網路服務框架裡的層級式結構。

部署狀況

網路配置


區域網路 (256.128.64.32)
網際網路
DNS
Record Type A
example.com → 256.128.64.32
路由器
192.168.0.1
主機
192.168.1.222:4000
256.128.64.32:9000