第2次作業題目-作業-QZ2

學號：112111119
姓名：賴俊升
作業撰寫時間：60 (mins，包含程式撰寫時間)
最後撰寫文件日期：2025/12/31

本份文件包含以下主題：(至少需下面兩項，若是有多者可以自行新增)

- ☑ 說明內容
- ☑ 個人認為完成作業須具備觀念

說明程式與內容

1. HTTP Status Code 有哪些？怎麼分類？

Ans:

HTTP 狀態碼是說「剛才的請求發生了什麼事」的一種三位數代碼。

這些代碼共分為 五大類（1xx 到 5xx）。

1. 1xx：資訊回應

伺服器已收到請求，正在處理

```
100 Continue ： 伺服器已收到請求的開頭部分，請客戶端繼續發送剩餘部分。

101 Switching Protocols ： 協議切換（例如從 HTTP 切換到 WebSocket 時會看到）。

102 Processing (棄用) : 請求多個檔案操作子請求。
伺服器已接收但無回應可用

103 Early Hints :  最終回應前返回的回應頭
```

2. 2xx：成功 (Success)

請求已成功被伺服器接收

```
200 OK :  請求成功

201 Created :  請求成功且建立了一個新的資源

202 Accepted : 伺服器已接受，但尚未處理
```

203 Non-Authoritative Information : 以200 OK狀態碼為起源，但回應了原始回應的修改版本

204 No Content： 請求成功但伺服器不回傳任何內容

205 Reset Content : 伺服器成功處理了請求，但沒有返回任何內容

206 Partial Content : 伺服器已經成功處理了部分GET請求

207 Multi-Status : 之後的訊息為xml

208 Already Reported : DAV繫結的成員已經在（多狀態）回應之前的部分被列舉，且未被再次包含

226 IM Used : 伺服器已經滿足了對資源的請求，對實體請求的一個或多個實體操作的結果表示

3. 3xx：重新導向 (Redirection)

資源已經沒了，瀏覽器需要採取進一步動作（通常會自動跳轉）才能完成請求。

300 Multiple Choices : 使用者或瀏覽器能夠自行選擇一個首選的位址進行重新導向

301 Moved Permanently： 網址永久變更，會轉移到新網址。

302 Found (Moved Temporarily)： 網址暫時變更，未來可能會改回來。

303 See Other : 當回應於POST（或PUT / DELETE）接收到回應時，客戶端應該假定伺服器已經收到資料

304 Not Modified： 未修改（快取）。 資源自上次請求後沒變過，直接讀取瀏覽器的快取即可，不用重新下載。

305 Use Proxy : 被請求的資源必須通過指定的代理才能被訪問

306 Switch Proxy (不再使用): 後續請求應使用指定的代理

307 Temporary Redirect : 請求應該與另一個URI重複，但後續的請求應仍使用原始的URI

308 Permanent Redirect : 請求和所有將來的請求應該使用另一個URI重複

4. 4xx：客戶端錯誤 (Client Error)

請求包含語法錯誤或無法完成

400 Bad Request： 請求語法錯誤，伺服器看不懂。

401 Unauthorized： 未授權。

402 Payment Required : 未定義

403 Forbidden： 禁止訪問。 即使你登入了，你也沒權限看這個頁面

404 Not Found： 找不到網頁。 你請求的資源不存在

405 Method Not Allowed： 方法不被允許。例如此接口只接受 POST，你卻用了 GET。

406 Not Acceptable : 請求的資源的內容特性無法滿足請求頭中的條件，因而無法生成回應實體，該請求不可接受

407 Proxy Authentication Required : 客戶端必須在代理伺服器上進行身分驗證

408 Request Timeout : time out

409 Conflict : 請求衝突

410 Gone : 資源不可用

411 Length Required : 伺服器拒絕在沒有定義Content-Length頭的情況下接受請求

414 Request-URI Too Long : url超過伺服器解釋長度，應使用post請求

429 Too Many Requests： 請求次數過多。你點擊太快或發送太多請求，被伺服器暫時擋下來。

451 Unavailable For Legal Reasons : 應該國法律問題，而拒絕請求

5. 5xx：伺服器錯誤 (Server Error)

伺服器未能完成明顯有效的請求

500 Internal Server Error： 內部伺服器錯誤。

502 Bad Gateway： 作為網關或代理的伺服器，從上游伺服器收到無效回應

503 Service Unavailable： 伺服器過載或正在維護中，暫時無法處理請求。

504 Gateway Timeout： 網關超時。上游伺服器回應太慢。

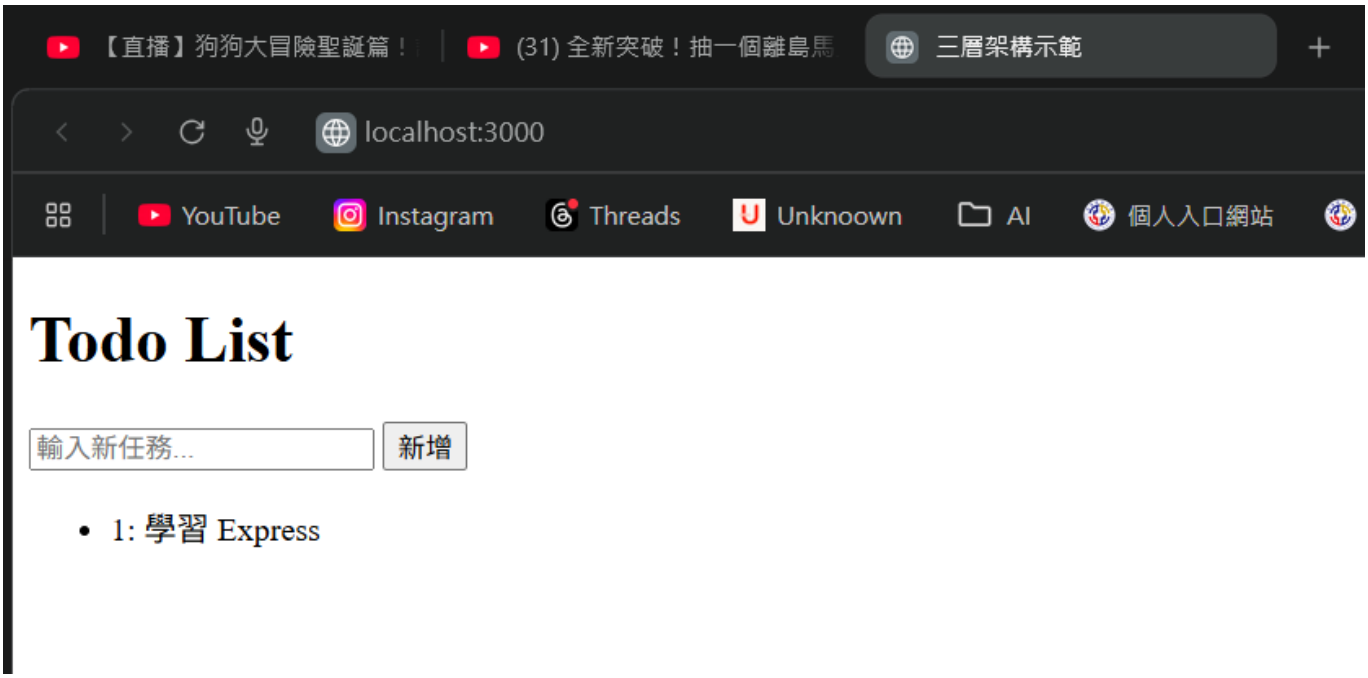
505 HTTP Version Not Supported : 版本問題

508 Loop Detected 無窮迴圈

511 Network Authentication Required 客戶端需要權限

1. 在 Express 中，設計基本上可以分成幾層？請依上述回答實作一個後端與前端的網站(需有程式碼)，並且將結果和執行畫面顯示於該份md，並逐步說明。

Ans:



三層架構說明 我們通常由外而內將程式分為這三層：

控制層 負責第一線面對客戶（瀏覽器）。

當接收 HTTP 請求 (req)、解析參數、呼叫下一層處理業務，最後決定回傳什麼狀態碼 (200 OK, 400 Bad Request) 與資料 (res)。

服務層 處理所有的邏輯。

這是程式最純粹的地方。它接收資料，進行運算、判斷

資料存取層

專門負責跟資料庫（Database）溝通。

執行 SQL 語句或 MongoDB 指令（如 find, insert）。

```
<!DOCTYPE html>
<html lang="zh-TW">
<head>
    <meta charset="UTF-8">
    <title>三層架構示範</title>
</head>
<body>
    <h1>Todo List</h1>

    <input type="text" id="taskInput" placeholder="輸入新任務...">
    <button onclick="addTodo()">新增</button>
    <p id="errorMsg" style="color: red;"></p>

    <ul id="list"></ul>

    <script>
```

```
        // 1. 讀取列表
        async function loadTodos() {
            const res = await fetch('/api/todos');
            const data = await res.json();
            const list = document.getElementById('list');
            list.innerHTML = data.map(t => `<li>${t.id}: ${t.task}
</li>`).join('');
        }

        // 2. 新增任務
        async function addTodo() {
            const input = document.getElementById('taskInput');
            const errorMsg = document.getElementById('errorMsg');
            errorMsg.innerText = ""; // 清空錯誤

            const res = await fetch('/api/todos', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ task: input.value })
            });

            const result = await res.json();

            if (res.status === 201) {
                input.value = ""; // 清空輸入框
                loadTodos();      // 重新整理列表
            } else {
                errorMsg.innerText = `錯誤: ${result.error} (代碼:
${res.status})`;
            }
        }

        loadTodos(); // 頁面載入時執行
    </script>
</body>
</html>
```

```
// controller.js
const todoService = require('./service');

exports.getTodos = async (req, res) => {
    try {
        const data = await todoService.getTodoList();
        res.status(200).json(data);
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
};

exports.createTodo = async (req, res) => {
    try {
```

```
        const { task } = req.body;
        const newTodo = await todoService.addTodo(task);
        res.status(201).json(newTodo); // 201 Created
    } catch (err) {
        if (err.message === "任務名稱不能為空！") {
            res.status(400).json({ error: err.message }); // 400 Bad Request
        } else {
            res.status(500).json({ error: "伺服器錯誤" });
        }
    }
};
```

```
// model.js
// 模擬資料庫
const todos = [
    { id: 1, task: "學習 Express", done: false }
];

class TodoModel {
    getAll() {
        return Promise.resolve(todos);
    }

    create(taskName) {
        return new Promise((resolve) => {
            const newTodo = {
                id: todos.length + 1,
                task: taskName,
                done: false
            };
            todos.push(newTodo);
            resolve(newTodo);
        });
    }
}
module.exports = new TodoModel();
```

```
// service.js
const todoModel = require('./model');

class TodoService {
    async getTodoList() {
        return await todoModel.getAll();
    }

    async addTodo(taskName) {
        // 業務邏輯檢查
        if (!taskName || taskName.trim() === "") {
            throw new Error("任務名稱不能為空！");
```

```
        }
        return await todoModel.create(taskName);
    }
}
module.exports = new TodoService();
```

```
// server.js
const express = require('express');
const app = express();
const path = require('path');
const controller = require('./controller');

app.use(express.json()); // 解析 JSON

// 路由設定
app.get('/api/todos', controller.getTodos);
app.post('/api/todos', controller.createTodo);

// 提供前端頁面
app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, 'index.html'));
});

app.listen(3000, () => {
    console.log('Server running at http://localhost:3000');
});
```