第2次作業題目-作業-QZ2

學號：112111122
姓名：周廉恆
作業撰寫時間：180 (mins，包含程式撰寫時間)
最後撰寫文件日期：2026/01/02

1. HTTP Status Code 有哪些？怎麼分類？

Ans:

狀態碼的第一位數字代表了回應的類別：

- 1xx (資訊回應 - Informational): 伺服器收到請求，需要使用者繼續執行操作。

```
100 Continue：server已接收請求之標頭，要求客戶端繼續請求剩餘部分。
101 Switching Protocols ： 協議切換（如:從HTTP協議換至HTTPS）。
102 Processing : 請求多個檔案操作子請求。
103 Early Hints :  最終回應前返回的回應頭
```

- 2xx (成功 - Success): 操作被成功接收並處理。

```
200 OK: 請求成功。最為標準的回應，表示server處理了GET、POST 等請求。

201 Created: 請求成功而且資料也「建立」成功。（常用於 POST 請求創建新資料、INSERT 資料庫成功後回傳）。
```

- 3xx (重新導向 - Redirection): 需要進一步的操作以完成請求（像是跳轉網址）。

```
301 Moved Permanently：永久搬家 (SEO 權重會轉移)。
302 Found：暫時搬家 (例如：未登入者被跳轉到 /login 頁面)。
304 Not Modified：沒修改，用快取。常發生在靜態資源 (如 CSS/IMG)，瀏覽器直接讀快取，不需伺服器重傳 。
```

- 4xx (客戶端錯誤 - Client Error): 請求包含語法錯誤或無法完成（如網址打錯、無權限進入等）。

```
400 Bad Request：請求語法錯誤。例如：後端要 JSON，前端傳了字串；或參數格式不對。
401 Unauthorized：「未認證」。你還沒登入，不知道你是誰 (需要 Session/JWT) 。
403 Forbidden：「被禁止」。你有登入 (認證了)，但權限不足 (例如：C級人員想進A級總控制台管理)。
404 Not Found：「找不到」。網址打錯，或是靜態檔案沒放在正確的託管目錄 。
```

- 5xx (伺服器錯誤 - Server Error): 伺服器在處理請求的過程中發生了錯誤，相對於客戶端錯誤這裡是伺服器端有問題，如伺服器資源抓取失敗。

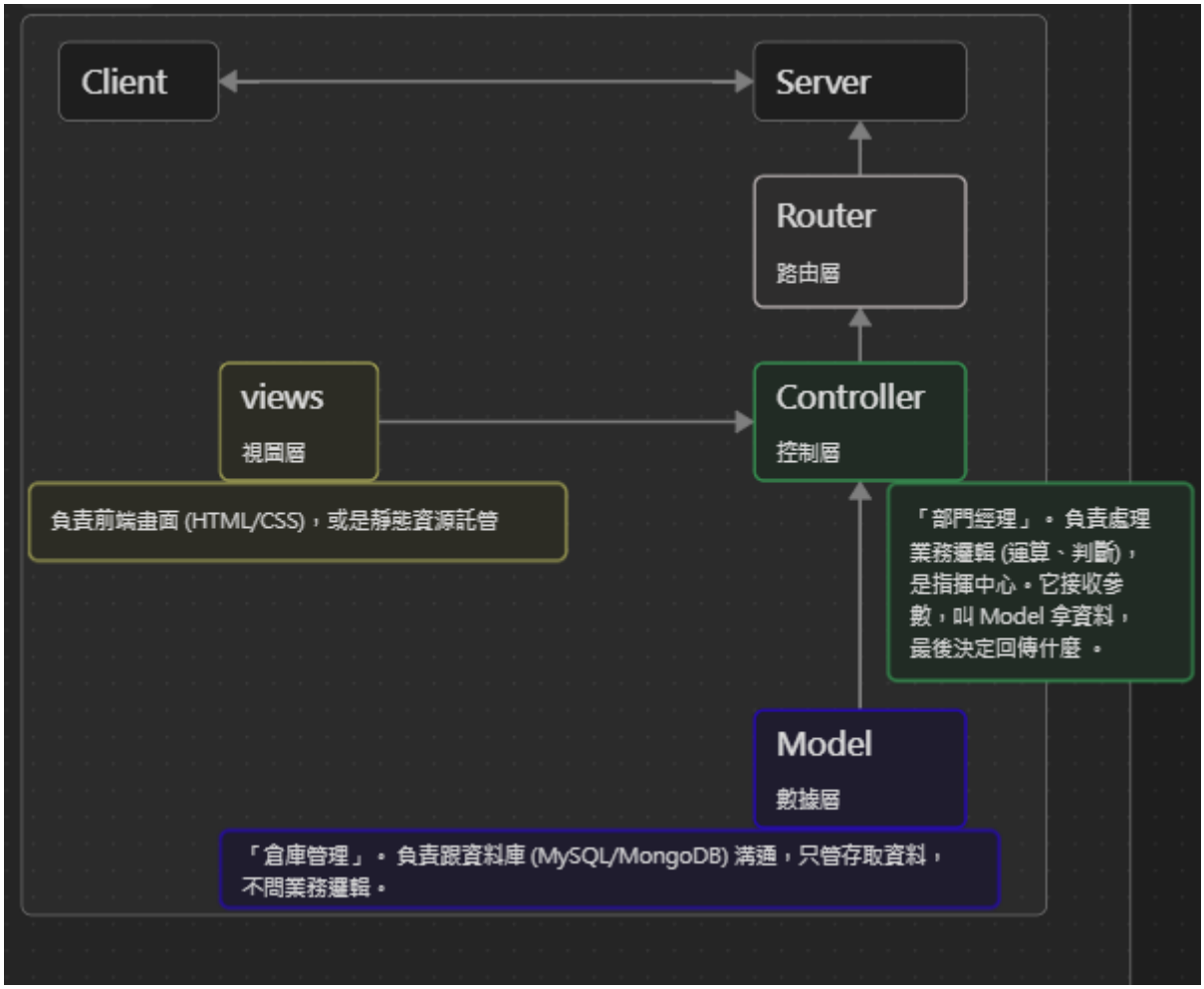
```
500 Internal Server Error：「伺服器內部錯誤」。程式碼崩潰、變數未定義、語法錯誤。在 Express 中通常由 app.use((err, req...)) 這種錯誤處理中介軟體捕捉 。
502 Bad Gateway：「網關錯誤」。通常是 Nginx 或雲端伺服器 (如 Google Cloud SQL) 連不到你的 Node.js 程式。
503 Service Unavailable：服務暫時無法使用 (伺服器過載或維護中)。
```

2. 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。

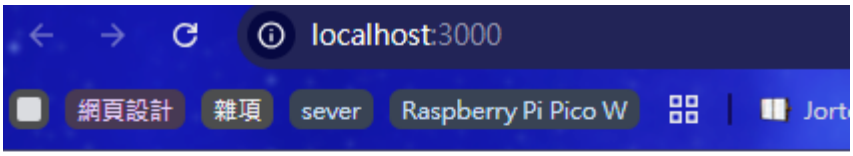
Ans:

首先這是一個基於 MVC (Model-View-Controller) 架構的**簡化版**解釋，也是 Express 開發中最常見的分層方式。

- 設計上通常分為 四層，畢竟express可以繼承MVC架構來使用。



※我將使用CommonJs來匯入express框架



MVC 搜尋用戶

麥克 搜尋

200 - 找到用戶 {"id":1,"name":"麥克","role":"Author"}

sever.js：

```
const express = require('express');//引入express框架
const app = express();

//啟動伺服器
const userRouter = require('./router');

// [視圖層] 託管靜態檔案 (public 資料夾)
app.use(express.static('public')); // [cite: 327]

// [路由層] 註冊路由模組
// 訪問 /api/user 時，會進入 userRouter
app.use('/api/user', userRouter);

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

router.js：

```
const express = require('express');//引入express框架
const router = express.Router();//抓取router工具
const userController = require('./modules/userController');

// 當有人訪問 GET / (這個路由模組的根目錄) 時，交給 controller 處理
router.get('/', userController.getUser);

module.exports = router;//模組化這個變數
```

userController.js：

```
const userModel = require('./userModule');

exports.getUser = function(req, res) {
    //概念為「往箱子裡放東西」，新增一個叫做getUser的功能；用法「exports.變數名」
    const name = req.query.name;
    const user = userModel.findUserByName(name);//使用findUserByName 的工具

    if (user) {
        res.json({ message: '找到用戶', data: user });
    } else {
        res.status(404).json({ message: '找不到用戶' });
    }
};
```

userModule.js：

```
// 模擬資料庫數據
const users = [
    { id: 1, name: '麥克', role: 'Author' },
    { id: 2, name: '史丹利', role: 'Reader' }
];

module.exports = {//這是 「直接把這個箱子換成別的東西」，替換物件
    // 提供一個方法來查找用戶
    findUserByName: function(name) {
        return users.find(u => u.name === name);
    }
};
```

./public/index.html

```
<!DOCTYPE html>
<html lang="zh-TW">
<head>
    <meta charset="UTF-8">
    <title>MVC 分層測試</title>
</head>
<body>
    <h1>MVC 搜尋用戶</h1>
    <input type="text" id="username" value="卡布奇諾">
    <button onclick="search()">搜尋</button>
    <p id="result"></p>

    <script>
        async function search() {
            const name = document.getElementById('username').value;
            // 呼叫後端 API
```

```
            const res = await fetch(`/api/user?name=${name}`);
            const data = await res.json();
            document.getElementById('result').innerText =
                `${res.status} - ${data.message} ${data.data ?
JSON.stringify(data.data) : ''}`;
        }
    </script>
</body>
</html>
```