第2次作業題目-作業-QZ2

學號：112111108
姓名：詹羽庭
作業撰寫時間：80 (mins，包含程式撰寫時間)
最後撰寫文件日期：2026/01/05

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 狀態碼主要分成 5 大類（依第一碼分類）：

☑ 1xx Informational（資訊）

- **100 Continue**：伺服器收到請求的一部分，客戶端可繼續送出剩餘內容
- **101 Switching Protocols**：切換通訊協定（例如升級到 WebSocket）

> 這類比較少在一般 CRUD 網站看到。

☑ 2xx Success（成功）

- **200 OK**：請求成功（常見於 GET）
- **201 Created**：新增成功（常見於 POST 成功建立資源）
- **204 No Content**：成功但不回傳內容（例如 DELETE 成功）

☑ 3xx Redirection（重新導向）

- **301 Moved Permanently**：永久轉址
- **302 Found**：暫時轉址
- **304 Not Modified**：快取可用（伺服器告訴你不用重新下載）

☑ 4xx Client Error（客戶端錯誤）

- **400 Bad Request**：請求格式錯（缺欄位/JSON 格式不對）
- **401 Unauthorized**：未登入/未授權（缺 token）
- **403 Forbidden**：有登入但沒權限
- **404 Not Found**：找不到路由或資源
- **409 Conflict**：衝突（例如重複投稿）
- **422 Unprocessable Entity**：格式正確但內容驗證不通過

☑ 5xx Server Error（伺服器錯誤）

- **500 Internal Server Error**：伺服器程式爆掉（try/catch 沒接住）

- **502 Bad Gateway**：常見於反向代理/網關錯誤
- **503 Service Unavailable**：服務暫停/維護/過載

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:常見做法可以用 **4～5 層**（越大專案越需要分層）：

1. **Route（路由層）**：負責 URL 對應到 controller
2. **Controller（控制器層）**：接 req/res，做參數整理、回應狀態碼
3. **Service（商業邏輯層）**：放「規則」：驗證、去重、流程控制
4. **Model / Repository（資料存取層）**：讀寫資料庫/檔案（例如 JSON）
5. **View / Frontend（前端呈現層）**：EJS / HTML / Fetch 呼叫 API 顯示資料

> 下面我用JSON 檔案當資料庫做一個小網站：
> 前端：輸入記事（title、content）並送出
> 後端：Express API 存到 `data/notes.json`，並提供列表顯示
> 分層：routes / controllers / services / models

實作：前後端網站（Express 分層）

A. 專案結構

```
my-notes-site/
├─ index.js
├─ package.json
├─ data/
│  └─ notes.json
├─ routes/
│  └─ notes.routes.js
├─ controllers/
│  └─ notes.controller.js
├─ services/
│  └─ notes.service.js
├─ models/
│  └─ notes.model.js
├─ views/
│  └─ index.ejs
└─ public/
├─ app.js
└─ style.css
```

B. 建立專案與安裝

1.先創建一個my-notes-site資料夾 用VSCode開啟資料夾 在資料夾內開啟終端機 並且在終端機分別輸入

```
npm init -y
npm i express ejs
npm i -D nodemon
```

1-1.將自動生成的json中加入scripts(可直接複製貼上)

```
{
  "name": "my-notes-site",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js"
  },
  "dependencies": {
    "ejs": "^3.1.10",
    "express": "^4.21.2"
  },
  "devDependencies": {
    "nodemon": "^3.1.9"
  }
}
```

2.建立一個index.js 並輸入下方程式碼

```
// 伺服器入口：負責基本設定、掛載路由、啟動 server

const express = require("express");
const path = require("path");

const notesRoutes = require("./routes/notes.routes");

const app = express();
const PORT = 3000;

// 1) View Engine（前端頁面用 EJS）
app.set("view engine", "ejs");
app.set("views", path.join(__dirname, "views"));

// 2) Middleware：解析 JSON / 表單
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// 3) 靜態資源（public 放 CSS / JS）
app.use(express.static(path.join(__dirname, "public")));

// 4) 首頁：回傳前端頁面（View）
app.get("/", (req, res) => {
  res.render("index");
```

```
});

// 5) API 路由（分層：Route -> Controller -> Service -> Model）
app.use("/api/notes", notesRoutes);

// 6) 404（找不到路由）
app.use((req, res) => {
  res.status(404).json({ message: "Not Found" });
});

// 7) 啟動
app.listen(PORT, () => {
  console.log(`Server running: http://localhost:${PORT}`);
});
```

3.建立一個routes資料夾，並在裡面創建一個notes.routes.js 並輸入下列程式碼

```
// routes/notes.routes.js
// 路由層：只負責「路徑」對應到 controller，不放商業邏輯

const express = require("express");
const router = express.Router();

const notesController = require("../controllers/notes.controller");

// GET /api/notes -> 取得全部 notes
router.get("/", notesController.getAll);

// POST /api/notes -> 新增 note
router.post("/", notesController.create);

module.exports = router;
```

4.建立一個controllers資料夾，並在裡面創建一個notes.controller.js 並輸入下列程式碼

```
// controllers/notes.controller.js
// 控制器層：接收 req/res，負責狀態碼、回應格式、參數整理

const notesService = require("../services/notes.service");

// 取得全部
function getAll(req, res) {
  const notes = notesService.getAllNotes();
  return res.status(200).json(notes); // 200 OK
}

// 新增
function create(req, res) {
  try {
```

```
    const { title, content } = req.body;

    const result = notesService.createNote({ title, content });

    // service 會回傳 { ok, status, data/message }
    if (!result.ok) {
      return res.status(result.status).json({ message: result.message });
    }

    return res.status(201).json(result.data); // 201 Created
  } catch (err) {
    return res.status(500).json({ message: "Internal Server Error" }); // 500
  }
}

module.exports = {
  getAll,
  create
};
```

5.創建services資料夾，並創一個notes.service.js檔 程式如下

```
// 商業邏輯層：放規則（驗證、去重、流程）
// controller 不要寫規則，model 不要寫規則

const notesModel = require("../models/notes.model");

// 取得全部
function getAllNotes() {
  return notesModel.readAll();
}

// 新增 note（含驗證 + 去重）
function createNote({ title, content }) {
  // 1) 基本驗證
  if (!title || !content) {
    return { ok: false, status: 400, message: "title 和 content 都必填" }; // 400
Bad Request
  }

  // 2) 去重規則：title 不能重複（示範 409 Conflict）
  const all = notesModel.readAll();
  const exists = all.some((n) => n.title.trim() === title.trim());
  if (exists) {
    return { ok: false, status: 409, message: "title 已存在（重複）" }; // 409
Conflict
  }

  // 3) 建立資料
  const newNote = {
    id: Date.now(),
    title: title.trim(),
```

```
    content: content.trim(),
    createdAt: new Date().toISOString()
  };

  notesModel.insert(newNote);

  return { ok: true, status: 201, data: newNote };
}

module.exports = {
  getAllNotes,
  createNote
};
```

6.創建一個models資料夾，並且創notes.model.js檔 輸入以下程式碼

```
// 資料存取層：只負責「讀寫資料」
// 這裡用 JSON 檔案當資料庫（demo 用）

const fs = require("fs");
const path = require("path");

const DATA_DIR = path.join(__dirname, "..", "data");
const DATA_PATH = path.join(DATA_DIR, "notes.json");

// 確保資料夾 & 檔案存在
if (!fs.existsSync(DATA_DIR)) fs.mkdirSync(DATA_DIR);
if (!fs.existsSync(DATA_PATH)) fs.writeFileSync(DATA_PATH, "[]", "utf8");

// 讀取全部
function readAll() {
  const raw = fs.readFileSync(DATA_PATH, "utf8");
  return JSON.parse(raw);
}

// 寫入全部（覆蓋）
function writeAll(data) {
  fs.writeFileSync(DATA_PATH, JSON.stringify(data, null, 2), "utf8");
}

// 插入一筆
function insert(note) {
  const all = readAll();
  all.push(note);
  writeAll(all);
}

module.exports = {
  readAll,
  insert
};
```

7.創一個views資料夾，並在資料夾內創一個index.ejs來放前端的東西 程式碼如下

```
<!DOCTYPE html>
<html lang="zh-Hant">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>My Notes Site</title>
  <link rel="stylesheet" href="/style.css" />
</head>
<body>
  <main class="wrap">
    <h1>記事小站（Express 分層示範）</h1>

    <section class="card">
      <h2>新增記事</h2>
      <form id="noteForm">
        <label>標題</label>
        <input name="title" id="title" placeholder="例如：今天學到 HTTP Status
Code" />

        <label>內容</label>
        <textarea name="content" id="content" placeholder="輸入內容...">
</textarea>

        <button type="submit">送出</button>
      </form>
      <p id="msg"></p>
    </section>

    <section class="card">
      <h2>記事列表</h2>
      <div id="list"></div>
    </section>
  </main>

  <script src="/app.js"></script>
</body>
</html>
```

8.創一個public資料夾存放靜態資料，在資料夾內創一個app.js 程式碼如下

```
// 前端：用 fetch 呼叫後端 API 並更新畫面

const form = document.querySelector("#noteForm");
const list = document.querySelector("#list");
const msg = document.querySelector("#msg");

async function loadNotes() {
```

```
  const res = await fetch("/api/notes");
  const notes = await res.json();

  list.innerHTML = notes
    .map(
      (n) => `
      <article class="note">
        <h3>${escapeHtml(n.title)}</h3>
        <p>${escapeHtml(n.content)}</p>
        <small>${new Date(n.createdAt).toLocaleString()}</small>
      </article>
    `
    )
    .join("");
}

// 簡單防 XSS（demo 用）
function escapeHtml(str = "") {
  return str
    .replaceAll("&", "&amp;")
    .replaceAll("<", "&lt;")
    .replaceAll(">", "&gt;")
    .replaceAll('"', "&quot;")
    .replaceAll("'", "&#039;");
}

form.addEventListener("submit", async (e) => {
  e.preventDefault();

  const title = document.querySelector("#title").value;
  const content = document.querySelector("#content").value;

  msg.textContent = "";

  const res = await fetch("/api/notes", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ title, content })
  });

  const data = await res.json();

  if (!res.ok) {
    // 例如 400 / 409
    msg.textContent = `❌ ${data.message}`;
    return;
  }

  msg.textContent = "☑ 新增成功！";
  form.reset();
  loadNotes();
});
```

```
// 初次載入
loadNotes();
```

9.在public資料夾內在創建一個style.css檔案 並鍵入下方程式(簡易設計)

```
body {
  font-family: system-ui, -apple-system, "Segoe UI", Roboto, "Noto Sans TC", sans-
serif;
  margin: 0;
  background: #f6f7fb;
}

.wrap {
  max-width: 860px;
  margin: 32px auto;
  padding: 0 16px;
}

.card {
  background: #fff;
  border-radius: 12px;
  padding: 16px;
  margin: 16px 0;
  box-shadow: 0 6px 20px rgba(0,0,0,0.06);
}

input, textarea, button {
  width: 100%;
  padding: 10px;
  margin: 8px 0 12px;
  border-radius: 10px;
  border: 1px solid #ddd;
  box-sizing: border-box;
}

textarea { min-height: 90px; }

button {
  cursor: pointer;
  border: none;
  background: #111;
  color: white;
  font-weight: 700;
}

.note {
  border: 1px solid #eee;
  border-radius: 10px;
  padding: 12px;
  margin: 10px 0;
  background: #fafafa;
}
```