LAB08 — Bookstore (Node.js + TypeScript + JSON File Storage)

Static HTML + Fetch API (No MySQL)

Overview

In this lab, you will build a simple Book Management web app. Users can add a new book from a form, and the web page will fetch and display the updated book list. Instead of using MySQL, the server will store data in a local JSON file.

Learning outcomes

- Set up an Express project using TypeScript
- Create REST endpoints (GET, POST) for CRUD basics
- Read/write a JSON file as a simple storage layer
- Use Fetch API on the client to display data dynamically
- Test endpoints using browser + API tester (optional)

Score (Total 26 points)

Part	**Points**
Part A — Project setup (TypeScript + Express)	6 pts
Part B — JSON file storage (read/write)	8 pts
Part C — REST endpoints (GET /books, POST /books/add)	8 pts
Part D — Frontend (static HTML + fetch)	4 pts

Starter code package (provided)

Download the starter resources_lab08 ZIP

What you need to complete:

- TODO 1–4 in src/services/bookFileDb.ts (read/write JSON file)
- TODO 5–10 in src/server.ts (middleware + routes)
  Write the API contract first and implement later
- TODO 11–12 in public/index.html (form action/method + input name)
- TODO 13–14 in public/app.js (fetch + render list)

Folder structure (required)

Your folder must look like this (important for grading):

```
se262_<studentID>_lab09/
  data/
    books.json
  public/
    index.html
    styles.css
    app.js
  src/
    server.ts
    services/
      bookFileDb.ts
  package.json
  tsconfig.json
```

Part A — Set up Node.js + TypeScript project (6 pts)

1) Initialize project

```
npm init -y
```

2) Install dependencies

```
npm i express
npm i -D typescript ts-node nodemon @types/node @types/express
```

3) Create tsconfig.json (minimum)

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "rootDir": "./src",
    "outDir": "./dist",
    "esModuleInterop": true,
    "strict": true
```

```
  }
}
```

4) Add scripts to package.json

```
{
  "scripts": {
    "dev": "nodemon --exec ts-node src/server.ts"
    "build": "tsc",
    "start": "node dist/server.js"
  }
}
```

Checkpoint: run the server

```
npm run dev
```

Part B — JSON file storage (8 pts)

1) Create data/books.json with this initial structure:

```
{
  "books": [
    { "bookNo": 1, "bookName": "Game of Thrones" },
    { "bookNo": 2, "bookName": "Clash of Kings" },
    { "bookNo": 3, "bookName": "The Name of the Wind" },
    { "bookNo": 4, "bookName": "The Wise Man's Fear" }
  ]
}
```

2) Create src/services/bookFileDb.ts and implement:

- readBooks(): Book[]
- addBook(bookName: string): Book

Path tip (works for ts-node and dist build):

```
import path from "path";
const dbPath = path.join(process.cwd(), "data", "books.json");
```

Rules:

- bookNo must be unique and incremental (next = max + 1)
- If bookName is empty, do not write to file (return error)
- Always read latest file before writing (avoid overwriting old data)

Hints: Check the implementation of the fileDb functions in the lecture slide.

Part C — Express endpoints (8 pts)

Create src/server.ts and implement these endpoints:

- GET / → send public/index.html
- GET /books → return JSON array of books
- POST /books/add → receive form field bookName and save to JSON file

Required middleware:

```
app.use(express.urlencoded({ extended: true })); // for HTML form
app.use(express.json()); // optional (API tester)
app.use(express.static(path.join(process.cwd(), "public")));
```

POST /books/add behavior:

- Read bookName from req.body.bookName
- If missing/empty → res.status(400).send(...)
- If success → res.redirect('/')

Hint: you can test POST /books/add API on Client API (Talend) first.

Part D — Frontend (Static HTML + Fetch) (4 pts)

In public/index.html:

- Create a form that submits to POST /books/add
- Create a container:
- Load public/app.js and use fetch('/books') to render the list

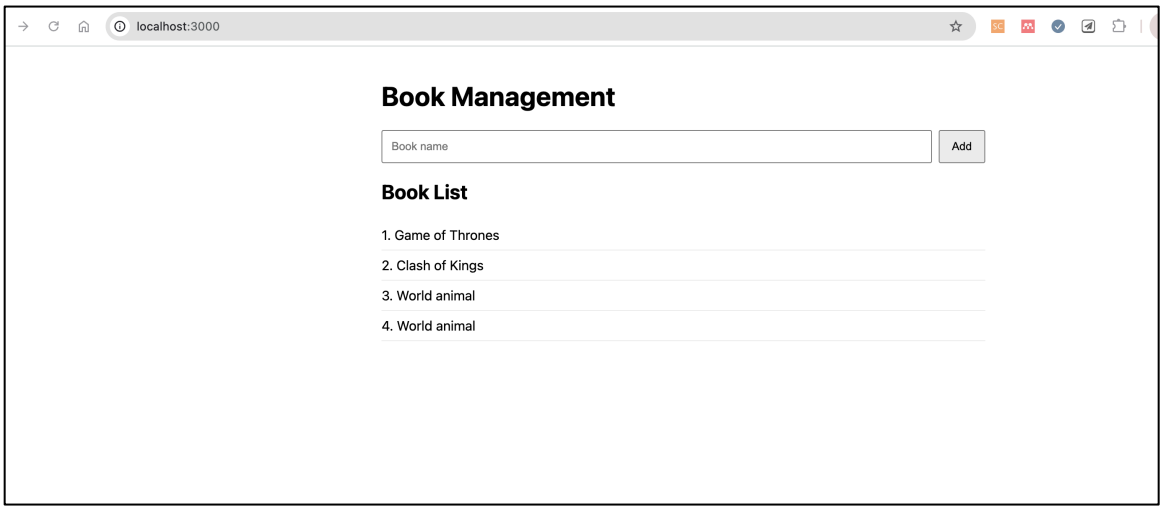
Example fetch (public/app.js):

```
async function loadBooks() {
  const res = await fetch("/books");
  const books = await res.json();

  const el = document.getElementById("book-list");
  el.innerHTML = books.map(b => `<div>${b.bookNo}.
${b.bookName}</div>`).join("");
}

window.addEventListener("DOMContentLoaded", loadBooks);
```

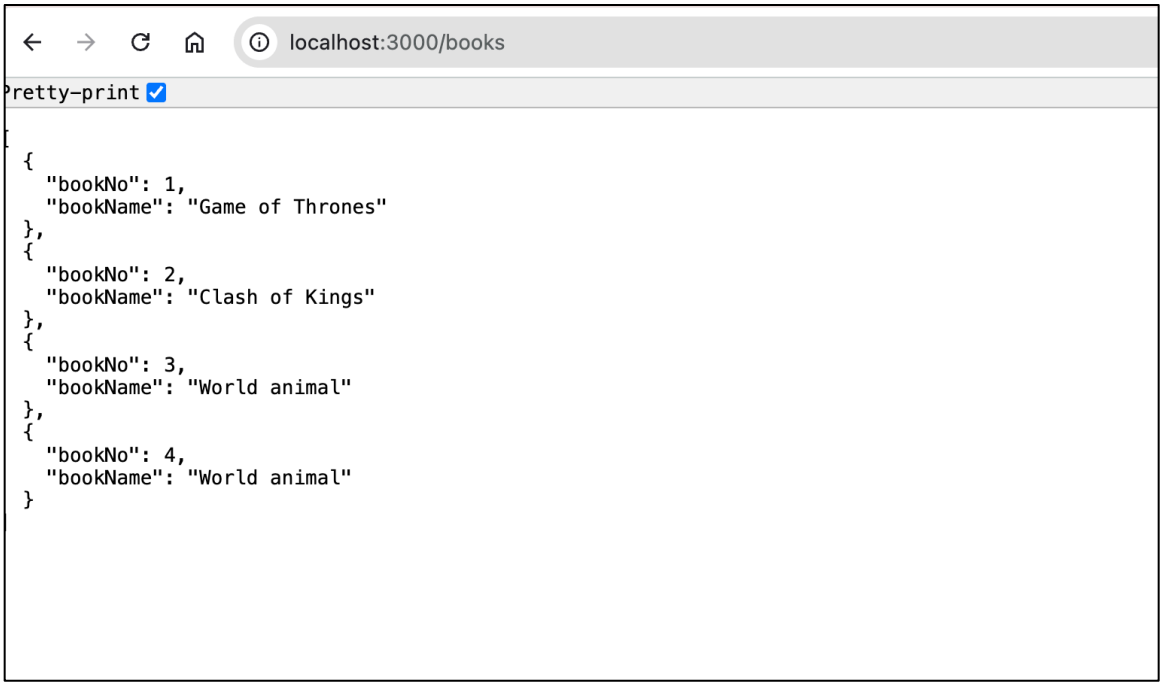
Checkpoint: after adding a new book, refresh the page and the new book should remain (stored in JSON file).

Testing checklist

- GET / shows the page



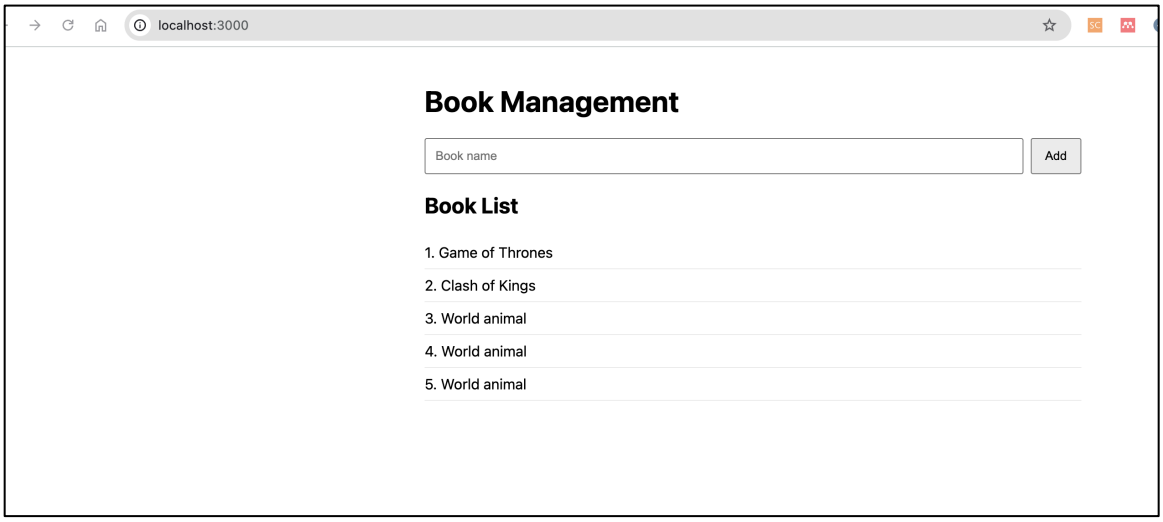
- GET /books returns JSON (array)



- POST /books/add adds a new book to data/books.json

- After restart server, data still exists

- UI displays updated list via fetch()



Challenge:

- Add functionality to delete book from the json file
- Add functionality to search books from the json file

Submission

Submit GitHub repository link containing your Lab 8 project.

Includes:

- data/books.json
- public/ (index.html, styles.css, app.js)
- src/ (server.ts, services/bookFileDb.ts)
- package.json, tsconfig.json

- Do NOT include node_modules in the zip.