LAB 04 — TypeScript, JSON, Modules, Express (Basics)

Overview

This lab contains 3 problems:
1) Create your own module and use it in an Express server
2) Practice TypeScript types + JSON load/update/save
3) Build a small Todo web app with Express + JSON storage

Learning Outcomes

- Create and use modules with export/import.
- Define TypeScript types and write typed functions.
- Read and write JSON data in Node.js.
- Run an Express server and handle GET/POST with HTML form data.

Requirements

- Node.js 18+ (recommended)
- VS Code
- Terminal (macOS / Windows)

Project Setup (Do Once)

1) Create a folder and initialize npm:

```
mkdir lab04
cd lab04
npm init -y
```

2) Install packages:

```
npm install express
npm install -D typescript ts-node @types/node @types/express
```

3) Create a minimal tsconfig.json (you may start from tsc --init and then edit):

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "CommonJS",
    "types": ["node"],
    "moduleResolution": "Node",
```

```
    "verbatimModuleSyntax": false,
    "rootDir": "./src",
    "outDir": "./dist",
    "strict": true,
    "esModuleInterop": true
  }
}
```

4) Update package.json (ES Modules + dev script):

```
{
  "scripts": {
    "dev": "ts-node src/server.ts"
  }
}
```

- Checkpoint: After you create src/server.ts, running npm run dev should start your server.

Problem 1 — Mini Practice: Create a Module and Use It in server.ts (2 points)

Goal: Create your own module and import it inside an Express route.

Steps

1. Create folders/files:

```
src/
  server.ts
  utils/
    format.ts
```

2. In src/utils/format.ts, export a function:

```
export function formatTitle(title: string): string {
  return title.trim().toUpperCase();
}
```

3. In src/server.ts, import the module and create a /hello route:

```
import express from "express";
import { formatTitle } from "./utils/format";

const app = express();

app.get("/hello", (req, res) => {
  const name = String(req.query.name ?? "student");
  res.send(`<h1>Hello ${formatTitle(name)}</h1>`);
});

app.listen(3000, () => {
```

```
  console.log("Open: http://localhost:3000/hello?name=toey");
});
```

- Run:

```
npm run dev
```

Checkpoint

- Open http://localhost:3000/hello?name=toey and confirm the page shows HELLO TOEY.

Problem 2 — Practice TypeScript + JSON (Load → Update → Save) (3 points)

Goal: Work with typed JSON data using service functions (no Express required).

Files to Create

```
src/
  models/
    student.ts
  services/
    studentService.ts
  data/
    students.json
  test-students.ts
```

Steps

1) src/models/student.ts

```
export type Student = {
  id: number;
  name: string;
  major: string;
};
```

2) src/data/students.json (start with 2 students):

```
[
  { "id": 1, "name": "Aom", "major": "SE" },
  { "id": 2, "name": "Ben", "major": "DS" }
]
```

3) src/services/studentService.ts (implement 3 functions):

```
import fs from "fs";
import path from "path";
import { Student } from "../models/student";

const dataPath = path.join(process.cwd(), "src", "data", "students.json");
```

```
export function loadStudents(): Student[] {
  const text = fs.readFileSync(dataPath, "utf-8");
  return JSON.parse(text) as Student[];
}

export function saveStudents(students: Student[]): void {
  fs.writeFileSync(dataPath, JSON.stringify(students, null, 2), "utf-8");
}

export function addStudent(
  students: Student[],
  name: string,
  major: string
): Student[] {
  const nextId =
    students.length === 0 ? 1 : Math.max(...students.map(s => s.id)) + 1;

  const newStudent: Student = { id: nextId, name, major };
  return [...students, newStudent];
}
```

4) src/test-students.ts (test your functions):

```
import { loadStudents, saveStudents, addStudent } from "./services/studentService";

const students = loadStudents();
const updated = addStudent(students, "Mina", "UX");
saveStudents(updated);

console.log("Updated students:", updated);
```

- Run:

```
npx ts-node src/test-students.ts
```

Checkpoint

- students.json is updated and includes Mina.
- Optional (async practice): Rewrite load/save using fs.promises and async/await.

Problem 3 — Todo Project with Express (HTML + POST form) (5 points)

Goal: Build a small Todo web app that stores todos in a JSON file.

Required Structure

```
src/
  server.ts
  models/
    todo.ts
  services/
```

```
    todoService.ts
  data/
    todos.json
```

Steps

1) src/models/todo.ts

```
export type Todo = {
  id: number;
  title: string;
  done: boolean;
};
```

2) src/data/todos.json

```
[
  { "id": 1, "title": "Buy milk", "done": false },
  { "id": 2, "title": "Read book", "done": true }
]
```

3) src/services/todoService.ts (same pattern as Problem 2):

- loadTodos(): Todo[]
- saveTodos(todos: Todo[]): void
- addTodo(todos: Todo[], title: string): Todo[]

4) src/server.ts (GET / shows list + form, POST /todos adds todo):

```
import express from "express";
import { loadTodos, saveTodos, addTodo } from "./services/todoService";

const app = express();
app.use(express.urlencoded({ extended: true })); // HTML form body (Week 5)

app.get("/", (req, res) => {
  const todos = loadTodos();
  const items = todos
    .map(t => `<li>${t.done ? "✅" : "⬜"} ${t.title}</li>`)
    .join("");

  res.send(`
    <h1>Todo List</h1>
    <ul>${items}</ul>

    <form method="POST" action="/todos">
      <input name="title" placeholder="New todo" />
      <button type="submit">Add</button>
    </form>
  `);
});
```

```
app.post("/todos", (req, res) => {
  const title = String(req.body.title ?? "").trim();
  if (!title) return res.status(400).send("Title is required");

  const todos = loadTodos();
  const updated = addTodo(todos, title);
  saveTodos(updated);

  res.redirect("/");
});

app.listen(3000, () => {
  console.log("Open: http://localhost:3000");
});
```

- Run:

```
npm run dev
```

Checkpoint

- GET / shows the todo list.
- Submitting the form adds a new todo and the page refreshes.
- todos.json is updated after submit.

Submission

Submit ONE GitHub repository link containing all three problems.

- Problem 1: module + /hello route working.
- Problem 2: studentService functions + test script, students.json updated.
- Problem 3: Todo app with Express + JSON storage (GET / and POST /todos).