CASE STUDY 3: Inventory Management System with File Storage

Scenario:
A company wants an **inventory management system** that:

- Manages products with name, quantity, and price
- Saves inventory data to a file
- Calculates total inventory value
- Provides a simple HTTP API to view inventory and add products
- Demonstrates **fs module, async file operations, modules, and HTTP server**

Step-by-Step Implementation

Step 1: Project Structure

```
inventory-app/
 ├─ modules/
 │   ├─ product.js      // Product class and inventory
management
 │   ├─ fileStorage.js  // File read/write utilities
 │   ├─ billing.js      // Inventory value calculations
 │   └─ index.js        // Re-export modules
 └─ server.js           // HTTP server
 └─ data/
      └─ inventory.json // Stores inventory
```

Step 2: Product Module

- Create `Product` class with `name`, `quantity`, and `price`.
- Methods to:
    - Add a product
    - Update quantity or price
- Maintain an in-memory array of all products.

Concepts: Classes, arrays, validation.

Step 3: File Storage Module

- Methods to:
    - Save current inventory to a JSON file asynchronously
    - Load inventory from file on startup
- Demonstrates **non-blocking I/O using fs module**

Step 4: Billing Module

- Calculate total inventory value = sum of (price × quantity) for all products.
- Export functions to get total value and summary.

Concepts: Arrays, loops, data aggregation, module exports.

Step 5: Index Module

- Re-export all modules for easier imports in `server.js`.

Step 6: HTTP Server

- Import modules from index.
- Endpoints:
    - `/addProduct` → adds a product via POST request
    - `/inventory` → returns all products as JSON
    - `/inventoryValue` → returns total inventory value
- Use async callbacks to read/write file without blocking server.

Concepts: HTTP server, JSON responses, async file handling.

Step 7: Event Loop and Async Behavior

- File write operations don't block HTTP requests.
- Multiple users can update inventory concurrently.

Step 8: Debugging

- Use `console.log` to monitor file saves and API requests.
- Optional: Use `node --inspect server.js` to debug step-by-step.

Outcome

- Fully functional **inventory management system** with **persistent storage**, **async I/O**, and **HTTP API**.
- Demonstrates **Node.js core concepts, modules, fs, async, and HTTP server**.