Full-Stack Development Bootcamp

Practical Assessment #3

Inventory Management System

Assignment Overview
Time Limit: 4 Hours
Assessment Type: Live Demo to Instructor
Total Marks: 100
Focus Areas: Database Relationships, Real-time Updates, Advanced Filtering

What is an Inventory Management System?

An Inventory Management System is an application that helps businesses track, organize, and manage their products and stock levels. Instead of manually tracking inventory in spreadsheets or notepads, an inventory manager lets you:

- **Track product information** with name, SKU, quantity, price, and supplier details
- **Organize products into categories** (like "Electronics", "Clothing", "Food")
- **Monitor stock levels** and get alerts when items are running low
- **Record stock movements** (adding or removing inventory)
- **Generate reports** on inventory value and stock status
- **Update product details** as prices and stock change

Real-world examples: Shopify Inventory, Square Inventory, and Cin7 are popular inventory management applications. Small retail stores, warehouses, and e-commerce businesses use these systems daily.

Technical Requirements

Tech Stack (Required)

Layer	**Technology**
Frontend	React with TypeScript or JavaScript
Backend	Express.js (Node) or Bun
Database	SQLite with Prisma ORM

Styling	Tailwind CSS (recommended) or plain CSS

Data Model

Products Table

Field	Type	Description
id	Int	Auto-generated primary key
name	String	Product name (e.g., "Laptop Dell XPS")
sku	String	Stock Keeping Unit - unique product code (e.g., "DELL-XPS-001")
category	String	Product category (e.g., "Electronics", "Accessories")
quantity	Int	Current stock quantity
minQuantity	Int	Minimum stock level before alert
price	Float	Unit price of the product
supplier	String	Supplier name
description	String?	Optional product description
createdAt	DateTime	Auto-generated timestamp
updatedAt	DateTime	Auto-generated update timestamp

Stock Movements Table (Optional but Recommended)

Field	Type	Description
id	Int	Auto-generated primary key
productId	Int	Foreign key to Products
movementType	String	"IN" (received) or "OUT" (sold/used)
quantity	Int	Amount added or removed
reason	String	Reason for movement (e.g., "Purchase Order", "Sale", "Damage")
notes	String?	Optional additional notes
createdAt	DateTime	Auto-generated timestamp

Required Features

Core CRUD Operations (40 marks)

1. **CREATE:** Add new products with name, SKU, category, quantity, minimum quantity, price, and supplier (10 marks)
2. **READ:** Display all products in a list/grid view with all relevant information (10 marks)
3. **UPDATE:** Edit existing product details and adjust stock quantities (10 marks)
4. **DELETE:** Remove products from the database (10 marks)

Additional Features (10 marks)

Stock Status Indicators: Display visual indicators (color-coded badges) showing:

- Green for "In Stock" (quantity > minQuantity)
- Yellow for "Low Stock" (quantity ≤ minQuantity)
- Red for "Out of Stock" (quantity = 0)

Component Architecture (15 marks)

Your React application must have at least these separate components:

1. **ProductCard.jsx/tsx** - Displays a single product with edit/delete buttons and stock status
2. **ProductList.jsx/tsx** - Maps through products and renders ProductCard for each
3. **AddProductForm.jsx/tsx** - Form/modal for creating and editing products
4. **CategoryFilter.jsx/tsx** - Dropdown or tabs to filter by category
5. **StockAdjustment.jsx/tsx** - Component to add/remove inventory from a product

Note: Props must be passed correctly between components. No component should be doing everything.

Required API Endpoints (20 marks)

1. **GET /api/products** - Fetch all products (with optional category filter query parameter)
2. **POST /api/products** - Create a new product
3. **PUT /api/products/:id** - Update an existing product
4. **DELETE /api/products/:id** - Delete a product
5. **PATCH /api/products/:id/adjust-stock** - Adjust inventory quantity for a product

CRITICAL REQUIREMENTS (15 marks)

These are non-negotiable based on your previous assessment:

- **NO commented-out API calls** - All fetch calls must be working
- **Frontend MUST connect to Backend** - Data must persist in database
- **Proper error handling** - try-catch blocks on ALL async functions
- **useEffect for initial data fetch** - Load products when app starts
- **No bugs in route parameters** - Use req.params.id correctly
- **Data validation** - Reject invalid quantities, negative prices, or duplicate SKUs

Demo Checklist

During your demo, you will be asked to show:

1. Add 3 products from different categories with different stock levels (one low stock, one out of stock)
2. Refresh the page - products should still be there (proves database connection)
3. Adjust stock for one product (increase and decrease)
4. Verify stock status indicators change color based on quantity
5. Edit one product's price and supplier information
6. Delete one product
7. Filter products by category
8. Show your component files (at least 5 separate components)
9. Show your server.js - explain your API endpoints and validation logic
10. Open browser DevTools - show Network tab during a stock adjustment API call

Grading Rubric

Criteria	**Marks**
Create Product (with validation)	10
Read/Display Products	10
Update Product Details	10
Delete Product	10
Stock Status Indicators	10
Component Architecture (5+ components)	15
API Endpoints (all 5 working)	20
Critical Requirements (error handling, validation, useEffect)	15
TOTAL	**100**

Tips for Success

- **Start with the backend** - Get your API endpoints working first using Postman or Thunder Client
- **Test database operations** - Use Prisma Studio (npx prisma studio) to verify data is saving
- **Build components incrementally** - Start with ProductCard, then ProductList, then forms
- **Implement validation early** - Catch invalid data before it reaches the database
- **Use meaningful test data** - Create products with realistic SKUs and prices

- **Keep DevTools open** - Watch for errors in Console and Network tabs during development
- **Plan your component hierarchy** - Sketch out how data flows between components before coding

Bonus Features (if time permits)

- Search functionality to find products by name or SKU
- Stock movement history showing when inventory was adjusted
- Low stock email/notification alerts
- Inventory value calculation (quantity × price)
- Export products to CSV
- Barcode/QR code generation for products

Good luck! Show us what you've learned over this past month.