STORE MANAGER KEEP TRACKING OF INVENTORY

A Project Report

Submitted to the university of Madras in partial fulfillment of the requirement for the award of Degree of

Bachelor Of Computer Applications

DOCUMENTATION SUBMISSION

ALPHA ARTS AND SCIENCE COLLEGE PORUR,

CHEENAI-600116

1.**INTRODUCTION:**

PROJECT TITLE : **STORE MANAGER KEEP TRACKING OF INVENTORY**

TEAM ID: NM2025TM40147

TEAM MEMBERS:

NM ID :	Name:
4A7BAD7DF0AF182ACCE4847A9A08FD5A	Prathesha R
7ADA9771BFAF125811C4C63BF12FD3DA	Deepika M
C46CEE82BCE4F8F83B84B5C94F100EE9	Evelin J
A56E8A1B4B9A1B6B9084588CEE5DFDAC	JENIFER P

2.**Project Overview:**

Purpose:

The project helps store managers to efficiently manage and track product inventory. It allows adding, updating, deleting, and viewing products. This reduces manual errors and provides quick access to stock details.

Features:

Add new products with details (name, quantity, price).

Update stock (increase/decrease quantity).

Delete products from inventory.

View product list with real-time stock levels.

Simple and user-friendly interface.

2.**Architecture:**

Frontend: HTML, CSS (for UI)

Backend: Node.js with Express.js

Database: SQL (MySQL / SQLite)

Flow:

User → Browser (HTML, CSS, JavaScript) → Node.js server (Express routes) → Database (CRUD

operations).

3.**Setup Instructions:**

Prerequisites:
Node.js installed
MySQL (or any SQL database) installed
Installation:
1. Clone the repository
git clone <your-repo-link>
cd store-manager
2. Install dependencies
npm install
3. Configure database connection inside config/db.js.
4. Start the server
node server.js

5.**Folder Structure:**

```
store-manager/
├── public/          # HTML, CSS, frontend assets
│   ├── index.html
│   ├── style.css
├── src/
│   ├── routes/
│   │   ├── products.js  # Product CRUD routes
│   ├── config/
│   │   ├── db.js       # Database connection
│   ├── app.js          # Express app
├── server.js           # Server bootstrap
├── package.json
```

6.**Running the Application:**

Frontend: Open index.html in browser

Backend:

node server.js

Runs on http://localhost:3000/

7. **Component Documentation:**

Key Components

1. Product List Page (inventory view)

Purpose: Displays all products with details (ID, name, quantity, price).

Functions: Fetches data from backend API and shows in a table.

Props/Inputs: Product data (fetched from database).

2. Add Product Form

Purpose: Allows the manager to add new products.

Functions: Collects product name, quantity, and price; sends data to backend.

Props/Inputs: Temporary form data (product details).

3. Update/Delete Actions

Purpose: Update stock quantity or remove products.

Functions: Buttons trigger API calls to update or delete product from DB.

Props/Inputs: Product ID (to identify the record).

4. Backend API (Node.js/Express)

Purpose: Handles CRUD operations (Create, Read, Update, Delete).

Functions: Connects with database and serves data to frontend.

Props/Inputs: Request body (form data), Request params (product ID).

Reusable Components

1. Form Component (HTML form)

Reused for adding and updating product details.

Configuration: Fields can be dynamically set (e.g., Name, Quantity, Price).

2. Table Component (HTML table)

Reused to display inventory data in list format.

Configuration: Columns (ID, Name, Quantity, Price) can be adjusted as needed.

3. API Utility Functions (JavaScript fetch calls)

Reused for all CRUD operations (GET, POST, PUT, DELETE).

Configuration: Endpoint URL and request body can be customized.

8. **State Management:**

Global State

The database (SQL) and Node.js server maintain the global state of the application.

Product details (name, quantity, price) are stored in the database and shared across all users.

Any change (add, update, delete) in one part of the system is reflected everywhere.

API endpoints in Node.js manage global state updates (CRUD operations).

Ensures data consistency across frontend and backend.

Local State

Local state is handled in the frontend (HTML forms & temporary variables in JavaScript).

Example: When a user fills the "Add Product" form, the values are stored temporarily before sending to the server.

Local state resets after form submission.

Used for temporary UI interactions (like showing success/error messages).

9.**User Interface:**

Add Product Page → Form to add items.

Inventory Page → Displays products in a table with update & delete options.

10.Styling:

CSS: Custom CSS for layout and design.

Simple and clean interface for easy usability.

11. Testing

Testing Strategy

1. Unit Testing

Tests individual backend functions (e.g., product addition, stock update).

Example: Check if addProduct() correctly inserts a record into the database.

Tools: Mocha/Chai (Node.js testing framework).

2. Integration Testing

Tests the interaction between backend (API routes) and database.

Example: Ensure that when the “Add Product” form is submitted, the new product is stored in SQL and retrieved correctly.

Tools: Supertest (for testing Express routes).

3. End-to-End (E2E) Testing

Tests the full workflow from frontend to backend.

Example: A user adds a product → system updates DB → product appears in the inventory table.

Tools: Manual testing in browser, or automated using Cypress.

Code Coverage

Code coverage ensures that all major functionalities (CRUD operations) are tested.

Tools like Istanbul (nyc) can be used with Mocha/Chai to measure how much of the code is tested.

Coverage includes:

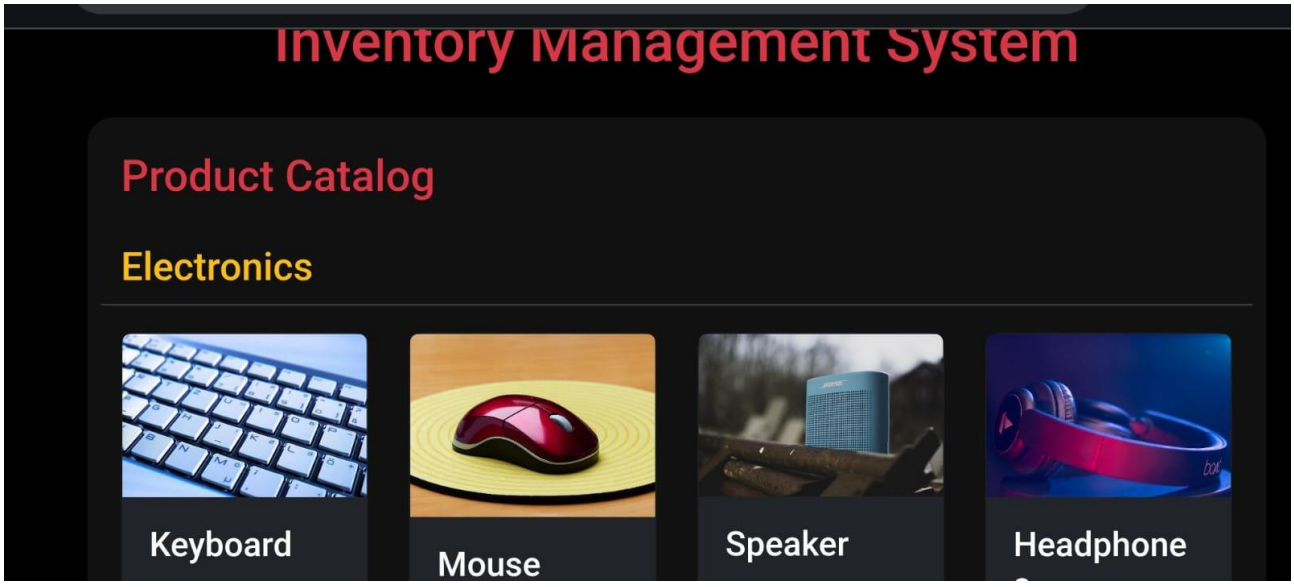
Routes tested (Add, Update, Delete, Fetch products).

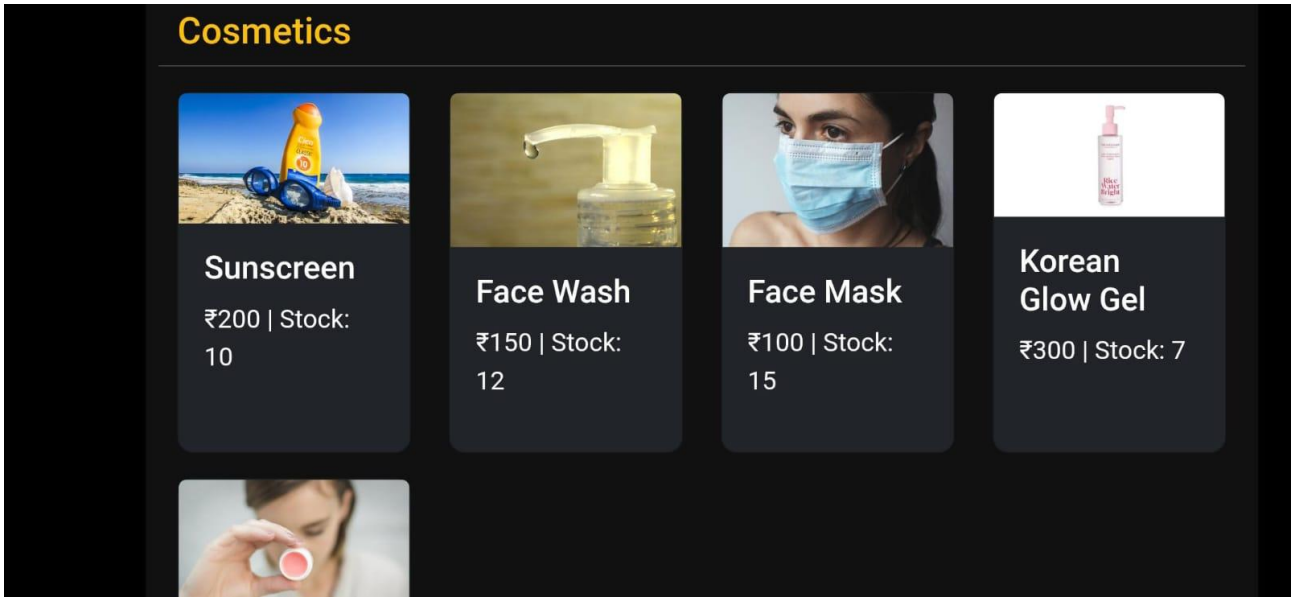
Database queries tested.

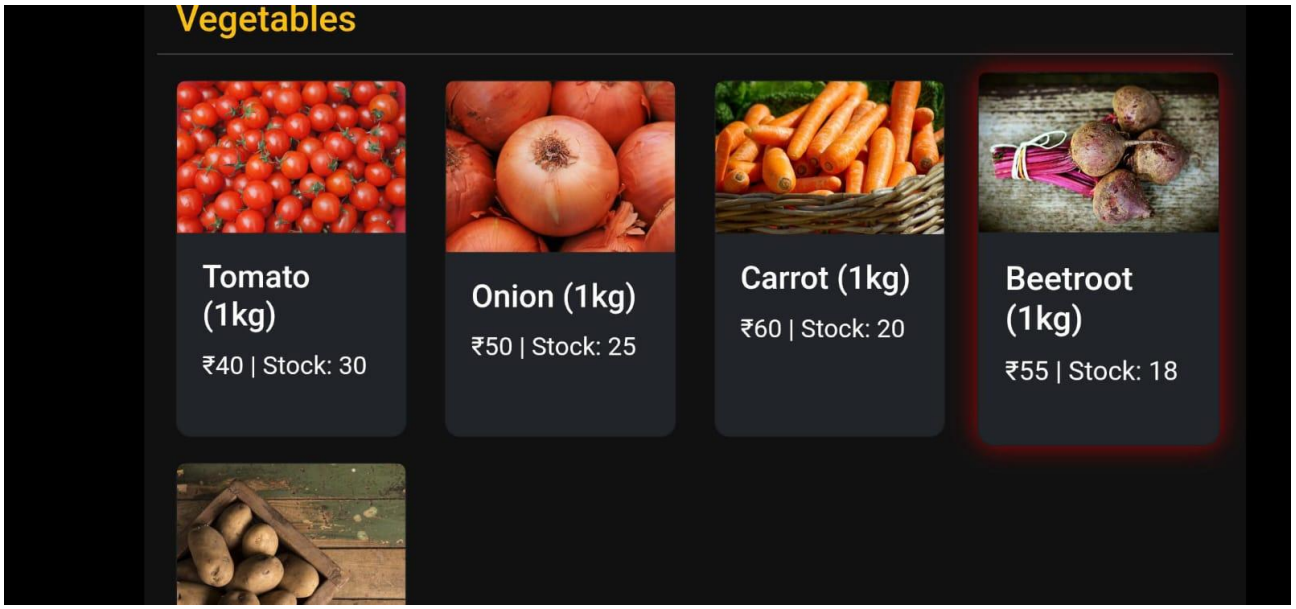
Edge cases (e.g., adding product with invalid data, deleting non-existing product).

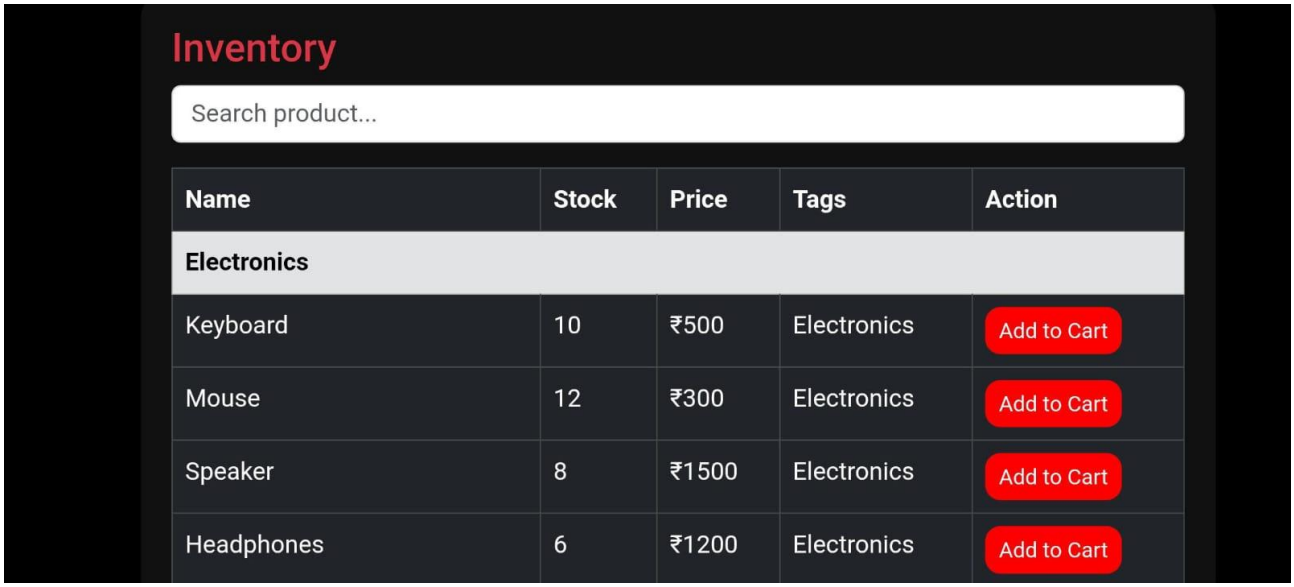
Goal: Achieve maximum coverage of backend logic to reduce bugs.

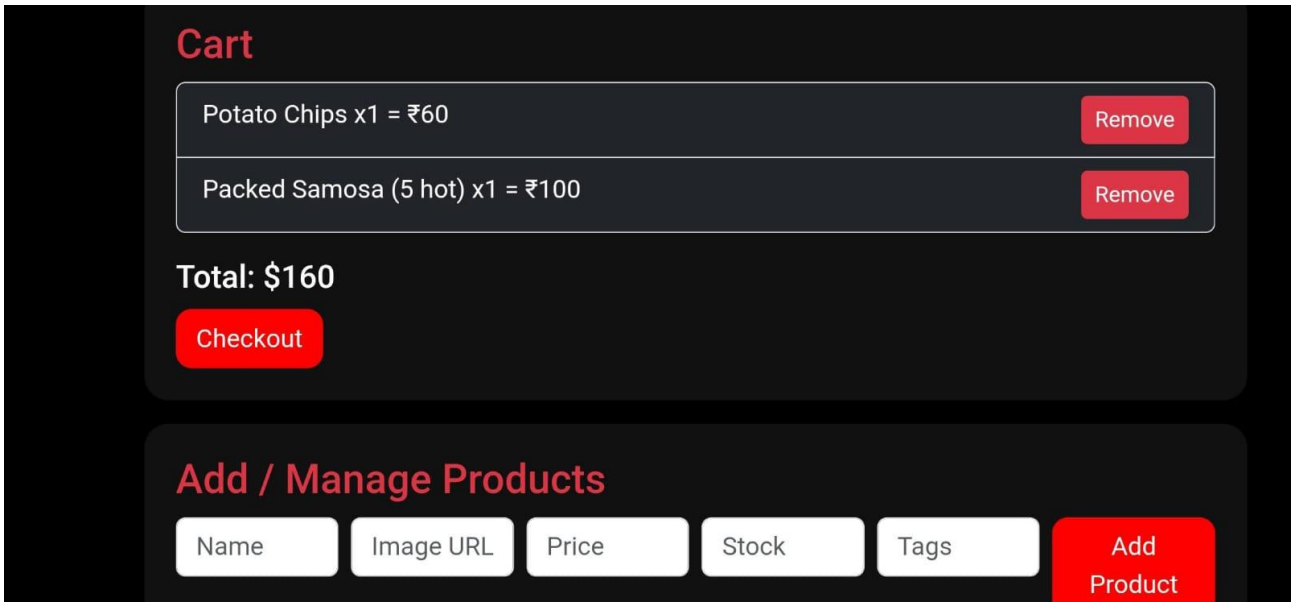
12.**SCREENSHOTS OR DEMO:**

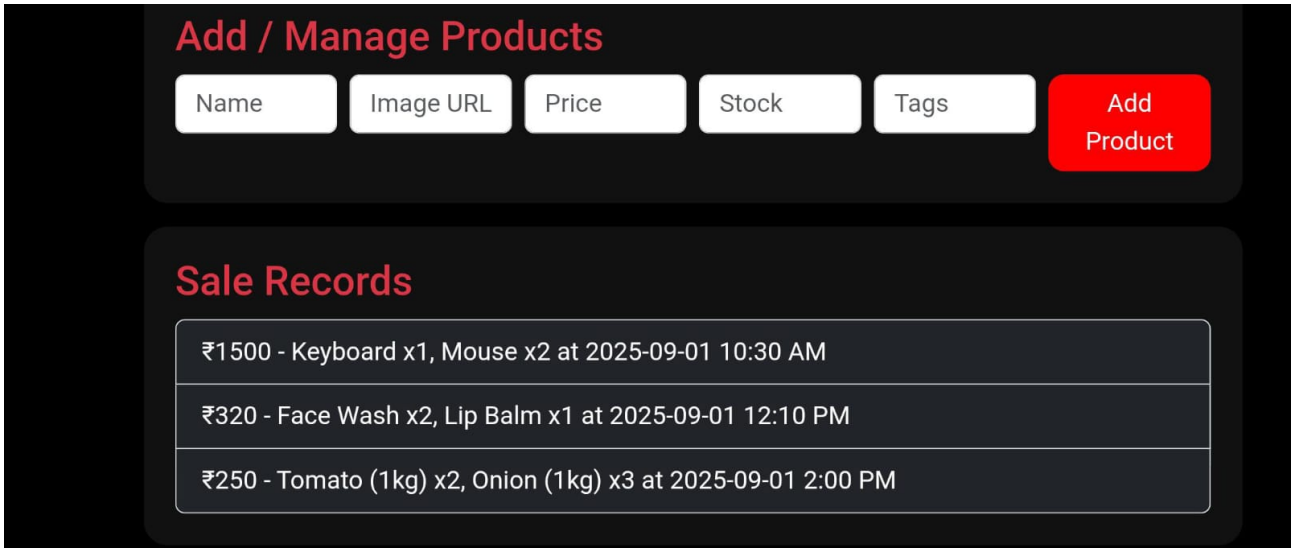



Cosmetics
Sunscreen
₹200 | Stock: 10
Face Wash
₹150 | Stock: 12
Face Mask
₹100 | Stock: 15
Korean Glow Gel
₹300 | Stock: 7


Vegetables
Tomato (1kg)
₹40 | Stock: 30
Onion (1kg)
₹50 | Stock: 25
Carrot (1kg)
₹60 | Stock: 20
Beetroot (1kg)
₹55 | Stock: 18


Inventory
Search product...
Name
Stock
Price
Tags
Action
Electronics
Keyboard
10
₹500
Electronics
Add to Cart
Mouse
12
₹300
Electronics
Add to Cart
Speaker
8
₹1500
Electronics
Add to Cart
Headphones
6
₹1200
Electronics
Add to Cart


Cart
Potato Chips x1 = ₹60
Remove
Packed Samosa (5 hot) x1 = ₹100
Remove
Total: $160
Checkout
Add / Manage Products
Name
Image URL
Price
Stock
Tags
Add Product



13.Known Issues:

No authentication system yet (anyone can access). **Project Overview:**

Purpose:

The project helps store managers to efficiently manage and track product inventory. It allows adding, updating, deleting, and viewing products. This reduces manual errors and provides quick access to stock details.

Features:

Add new products with details (name, quantity, price).

Update stock (increase/decrease quantity).

Delete products from inventory.

View product list with real-time stock levels.

Simple and user-friendly interface.

2.Architecture:

Frontend: HTML, CSS (for UI)

Backend: Node.js with Express.js

Database: SQL (MySQL / SQLite)

Flow:

User → Browser (HTML, CSS, JavaScript) → Node.js server (Express routes) → Database (CRUD operations).

3.Setup Instructions:

Prerequisites:

Node.js installed

MySQL (or any SQL database) installed

Installation:

1. Clone the repository

git clone <your-repo-link>

cd store-manager

2. Install dependencies

npm install

3. Configure database connection inside config/db.js.

4. Start the server

node server.js

5.Folder Structure:

```
store-manager/
├── public/          # HTML, CSS, frontend assets
│   ├── index.html
│   ├── style.css
├── src/
│   ├── routes/
│   │   ├── products.js  # Product CRUD routes
│   ├── config/
│   │   ├── db.js       # Database connection
│   ├── app.js         # Express app
├── server.js          # Server bootstrap
```

├── package.json

6.Running the Application:

Frontend: Open index.html in browser

Backend:

node server.js

Runs on http://localhost:3000/

7. Component Documentation:

Key Components

1. Product List Page (inventory view)

Purpose: Displays all products with details (ID, name, quantity, price).

Functions: Fetches data from backend API and shows in a table.

Props/Inputs: Product data (fetched from database).

2. Add Product Form

Purpose: Allows the manager to add new products.

Functions: Collects product name, quantity, and price; sends data to backend.

Props/Inputs: Temporary form data (product details).

3. Update/Delete Actions

Purpose: Update stock quantity or remove products.

Functions: Buttons trigger API calls to update or delete product from DB.

Props/Inputs: Product ID (to identify the record).

4. Backend API (Node.js/Express)

Purpose: Handles CRUD operations (Create, Read, Update, Delete).

Functions: Connects with database and serves data to frontend.

Props/Inputs: Request body (form data), Request params (product ID).

Reusable Components

1. Form Component (HTML form)

Reused for adding and updating product details.

Configuration: Fields can be dynamically set (e.g., Name, Quantity, Price).

2. Table Component (HTML table)

Reused to display inventory data in list format.

Configuration: Columns (ID, Name, Quantity, Price) can be adjusted as needed.

3. API Utility Functions (JavaScript fetch calls)

Reused for all CRUD operations (GET, POST, PUT, DELETE).

Configuration: Endpoint URL and request body can be customized.

8. State Management:

Global State

The database (SQL) and Node.js server maintain the global state of the application.

Product details (name, quantity, price) are stored in the database and shared across all users.

Any change (add, update, delete) in one part of the system is reflected everywhere.

API endpoints in Node.js manage global state updates (CRUD operations).

Ensures data consistency across frontend and backend.

Local State

Local state is handled in the frontend (HTML forms & temporary variables in JavaScript).

Example: When a user fills the "Add Product" form, the values are stored temporarily before sending to the server.

Local state resets after form submission.

Used for temporary UI interactions (like showing success/error messages).

9.User Interface:

Add Product Page → Form to add items.

Inventory Page → Displays products in a table with update & delete options.

10.Styling:

CSS: Custom CSS for layout and design.

Simple and clean interface for easy usability.

11. Testing

Testing Strategy

1. Unit Testing

Tests individual backend functions (e.g., product addition, stock update).

Example: Check if addProduct() correctly inserts a record into the database.

Tools: Mocha/Chai (Node.js testing framework).

2. Integration Testing

Tests the interaction between backend (API routes) and database.

Example: Ensure that when the "Add Product" form is submitted, the new product is stored in SQL and retrieved correctly.

Tools: Supertest (for testing Express routes).

3. End-to-End (E2E) Testing

Tests the full workflow from frontend to backend.

Example: A user adds a product → system updates DB → product appears in the inventory table.

Tools: Manual testing in browser, or automated using Cypress.

Code Coverage

Code coverage ensures that all major functionalities (CRUD operations) are tested.

Tools like Istanbul (nyc) can be used with Mocha/Chai to measure how much of the code is tested.

Coverage includes:

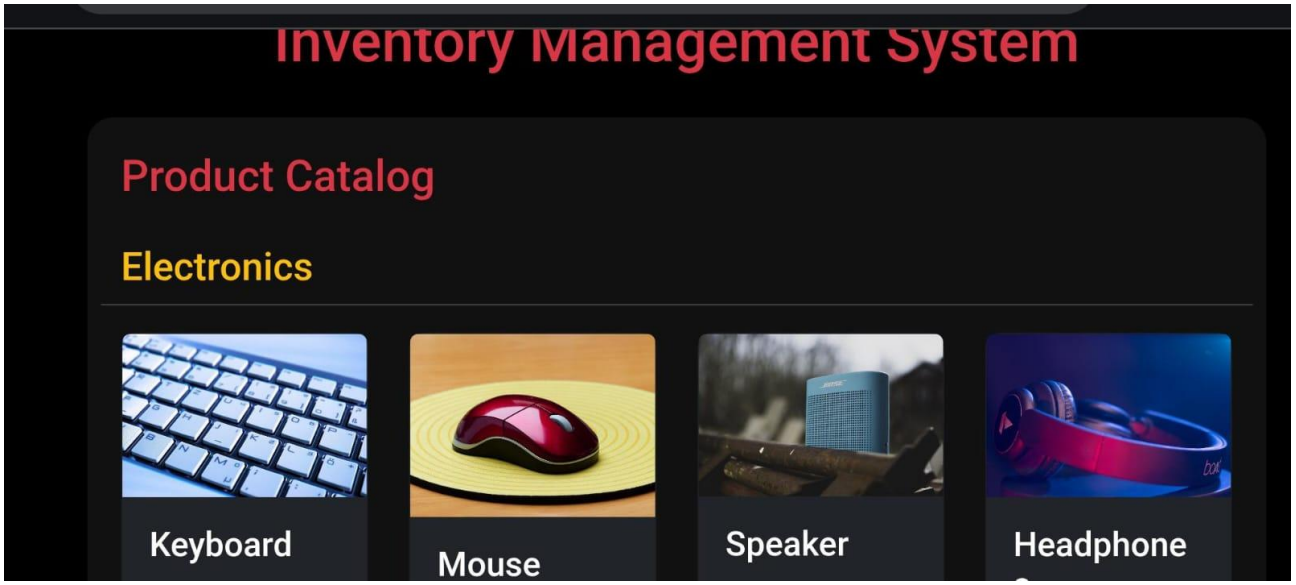
Routes tested (Add, Update, Delete, Fetch products).

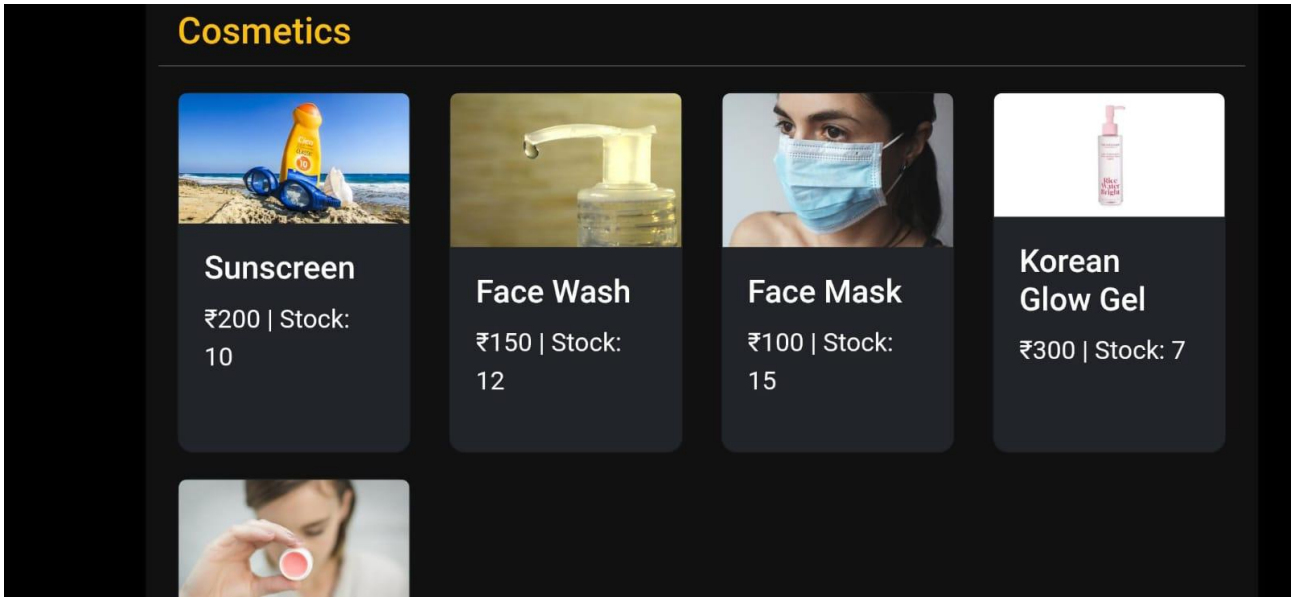
Database queries tested.

Edge cases (e.g., adding product with invalid data, deleting non-existing product).

Goal: Achieve maximum coverage of backend logic to reduce bugs.

12.SCREENSHOTS OR DEMO:


Inventory Management System
Product Catalog
Electronics
Keyboard
Mouse
Speaker
Headphone


Cosmetics
Sunscreen
₹200 | Stock: 10
Face Wash
₹150 | Stock: 12
Face Mask
₹100 | Stock: 15
Korean Glow Gel
₹300 | Stock: 7

Vegetables

Tomato (1kg)

₹40 | Stock: 30

Onion (1kg)

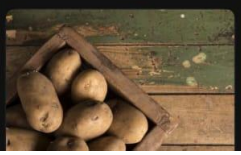
₹50 | Stock: 25

Carrot (1kg)

₹60 | Stock: 20

Beetroot (1kg)

₹55 | Stock: 18



Snacks

Biscuits (1 pack)

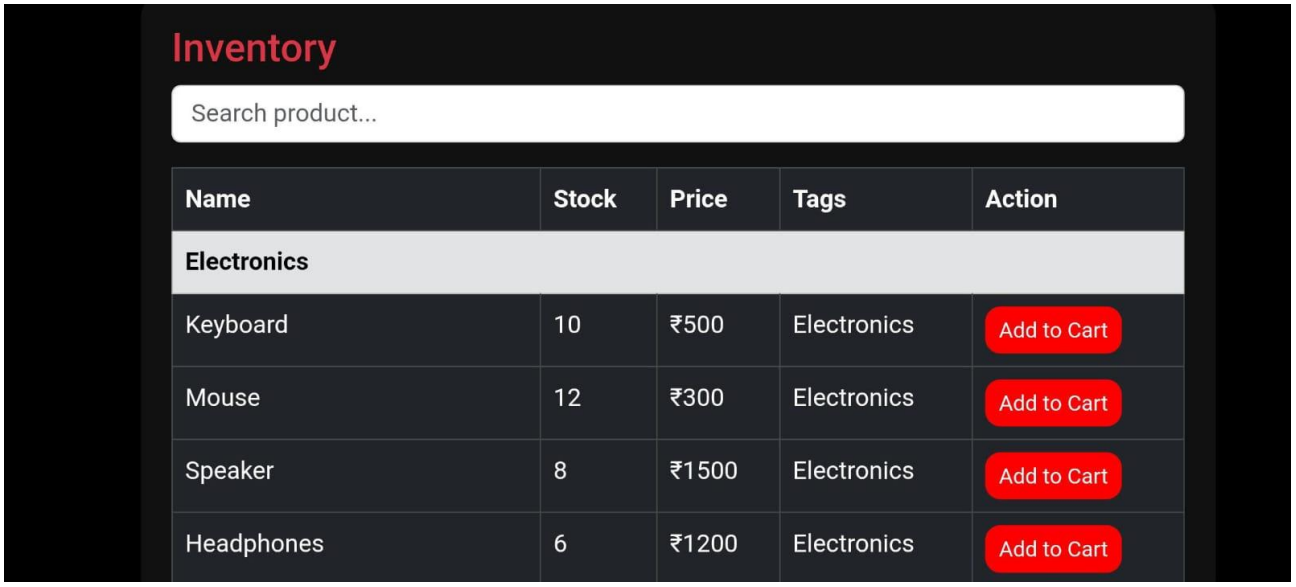
₹30 | Stock: 50

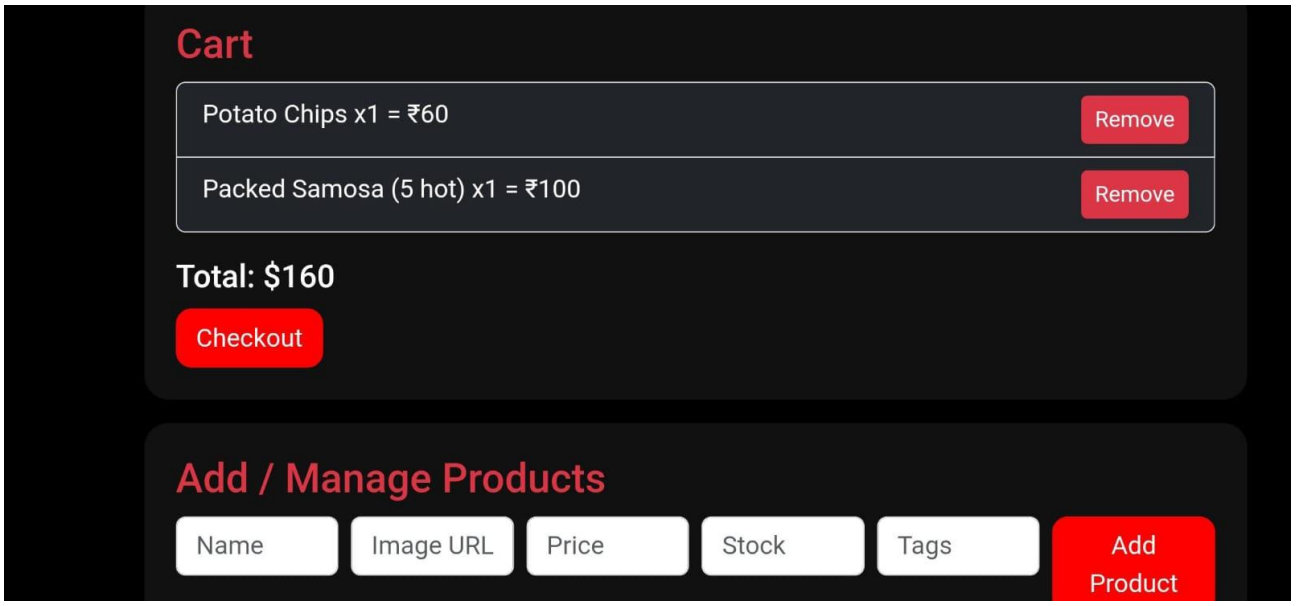
Packed Samosa (5 hot)

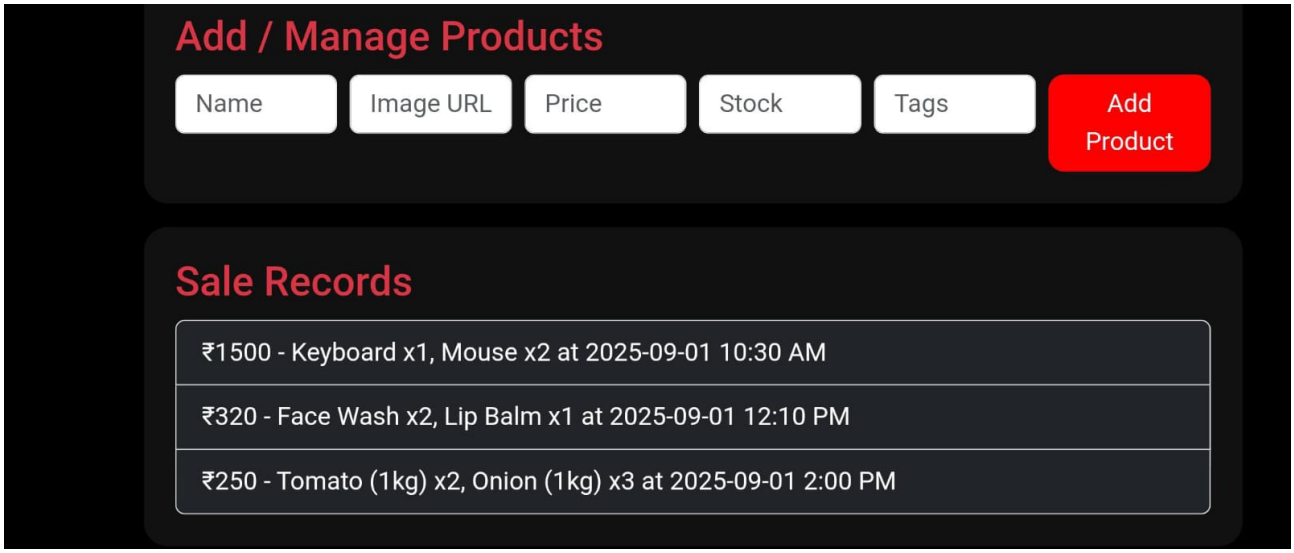
₹100 | Stock: 15

Potato Chips

₹60 | Stock: 40


Inventory
Search product...
Name
Stock
Price
Tags
Action
Electronics
Keyboard
10
₹500
Electronics
Add to Cart
Mouse
12
₹300
Electronics
Add to Cart
Speaker
8
₹1500
Electronics
Add to Cart
Headphones
6
₹1200
Electronics
Add to Cart


Cart
Potato Chips x1 = ₹60
Remove
Packed Samosa (5 hot) x1 = ₹100
Remove
Total: $160
Checkout
Add / Manage Products
Name
Image URL
Price
Stock
Tags
Add Product



13.Known Issues:

No authentication system yet (anyone can access).

No advanced reports (only basic stock list).

14.Future Enhancements:

Add authentication (login for managers).

Generate sales & stock reports.

Add barcode scanning.

Implement React frontend for better UI