

Technical Design Document

Angular Product Inventory Management

ARIJIT BASAK

March 15, 2025

Report

Prepared By: Arijit Basak

Date: March 15, 2025

Contents

1	Introduction	1
2	Purpose of this Document	1
3	Scope	1
3.1	Included in Scope	1
3.2	Not Included in Scope	1
4	System Requirements	1
4.1	Development System Specifications	1
5	Folder Structure	2
6	Application Design	2
6.1	Modules & Components	2
6.1.1	Feature Modules	2
7	Database Structure	3
8	State Management	3
9	Routing & Lazy Loading	3
10	Component Lifecycle Hooks	3
11	Error Handling & Logging	4
12	Testing Strategy	4
13	Performance Optimization	4
14	Deployment Strategy	4
15	User Interface Screenshots	5
15.1	Product List Page	5
15.2	Product List Page Upon Sign In	6
15.3	Sign In Page	6
15.4	Sign In Demo	7
15.5	Register Page	7
15.6	Register Page Demo	8
15.7	Add Product	8
15.8	Add Product Successfull	9
15.9	Search Bar	9
15.10	Edit Products	10
15.11	View Page	10
16	Application Endpoints	11

17 API Services	11
17.1 Example API Calls	11
18 Authentication Flow	11
19 Dependencies	11
20 Issues & Risks	12
20.1 Potential Issues	12
20.2 Future Improvements	12

1 Introduction

The **Angular Product Inventory Management** system is a web-based application that allows users to efficiently manage product listings. The system provides functionalities for **adding, updating, viewing, and deleting products**. The project is built using **Angular 15**, Bootstrap, and JSON Server for mock API handling.

2 Purpose of this Document

This **Technical Design Document (TDD)** provides a comprehensive overview of the project's **architecture, dependencies, file structure, and functionalities**. It serves as a reference for developers and stakeholders.

3 Scope

3.1 Included in Scope

- Environment Specification
- System Requirements
- Folder Structure
- Module & Component Breakdown
- API Services
- Authentication Flow
- Issues & Risks

3.2 Not Included in Scope

- Business Requirements
- Production Deployment Strategy
- Real Backend API Integration

4 System Requirements

4.1 Development System Specifications

- **Operating System:** Windows / Linux / macOS
- **Processor:** Intel Core i5 or higher
- **RAM:** 8GB minimum
- **Storage:** 500MB minimum

-
- **Software Requirements:**
 - Node.js (v14 or later)
 - Angular CLI (v15.0.0)
 - JSON Server (for mock API)
 - Jest (for unit testing)

5 Folder Structure

- **src/app/core/** – Contains global services like **DataService** and **AuthService**.
- **src/app/features/** – Feature modules:
 - **inventory/** (Product Management)
 - **auth/** (Authentication)
 - **about/** (Informational pages)
- **src/app/shared/** – Contains reusable components.
- **src/assets/** – Stores images, styles, and static files.
- **angular.json** – Angular project configuration.
- **package.json** – Dependency management.

6 Application Design

6.1 Modules & Components

The system follows a modular approach with **Lazy Loading** for better performance.

6.1.1 Feature Modules

- **Inventory Module (inventory/)**
 - **Product List** – Displays all products.
 - **Product Detail** – Shows details of a selected product.
 - **Add Product** – Adds a new product to the inventory.
 - **Update Product** – Modifies existing product details.
- **Authentication Module (auth/)**
 - Handles login/logout operations and access control.

7 Database Structure

The application uses **JSON Server** to mock a database. Below is the format of the 'db.json' file:

```
{
  "users": [
    { "id": 1, "name": "John Doe", "role": "admin" }
  ],
  "products": [
    { "id": 1, "name": "Laptop", "price": 1000, "category": "Electronics" }
  ]
}
```

Products contain an 'id', 'name', 'price', and 'category'. The API supports **CRUD** operations.

8 State Management

Angular uses **services** and **RxJS** to manage state efficiently. The 'DataService' uses a 'BehaviorSubject' for real-time updates:

```
private productSubject = new BehaviorSubject<Product []>([]);
public products$ = this.productSubject.asObservable();
```

This allows components to react to data changes instantly.

9 Routing & Lazy Loading

The application employs **lazy loading** to optimize performance. Routes are defined in 'app-routing.module.ts':

```
const routes: Routes = [
  { path: 'inventory', loadChildren: () => import('./features/inventory/inventory.module').then(m => m.InventoryModule) }
];
```

This prevents loading unnecessary modules until required.

10 Component Lifecycle Hooks

Angular provides lifecycle hooks to manage component states. The main hooks used include:

- **ngOnInit()** – Fetches product data on component initialization.
- **ngOnDestroy()** – Cleans up subscriptions to prevent memory leaks.

Example usage:

```
ngOnInit(): void {
  this.dataService.getProducts().subscribe(products => {
    this.products = products;
  });
}
```

11 Error Handling & Logging

API calls handle errors using 'catchError':

```
getProducts(): Observable<Product[]> {
  return this.http.get<Product[]>('/api/products').pipe(
    catchError(error => {
      console.error('Error fetching products', error);
      return throwError(() => new Error('Failed to load products'));
    })
  );
}
```

Errors are logged and displayed appropriately.

12 Testing Strategy

The project uses ****Jest**** for unit testing. Example of a test case for 'DataService':

```
describe('DataService', () => {
  it('should fetch products', () => {
    const service = new DataService();
    service.getProducts().subscribe(products => {
      expect(products.length).toBeGreaterThan(0);
    });
  });
});
```

Tests verify functionality and API interactions.

13 Performance Optimization

The application is optimized using:

- **Lazy Loading** – Loads feature modules only when needed.
- **OnPush Change Detection** – Reduces unnecessary UI updates.
- **TrackBy Function** – Optimizes list rendering.

14 Deployment Strategy

For production deployment, the application is built with:

```
ng build --prod
```

Before running the project, ensure the following:

1. Install **Node.js** and **Angular CLI**.

2. Install project dependencies:

```
npm install
```

3. Start the JSON Server:

```
npm run json-server
```

4. Run the Angular development server:

```
npm run start
```

15 User Interface Screenshots

Below are some key UI pages of the application.

15.1 Product List Page

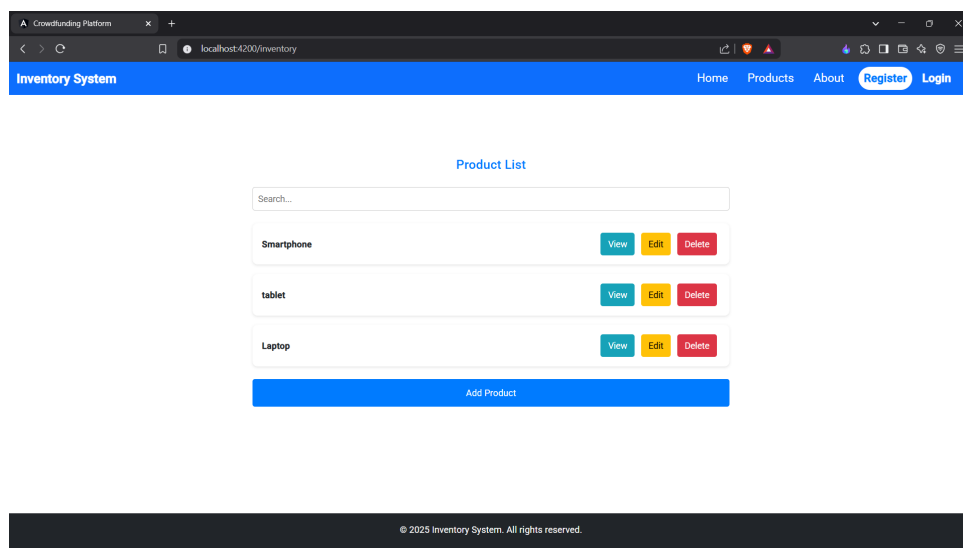


Figure 1: Product List Page

15.2 Product List Page Upon Sign In

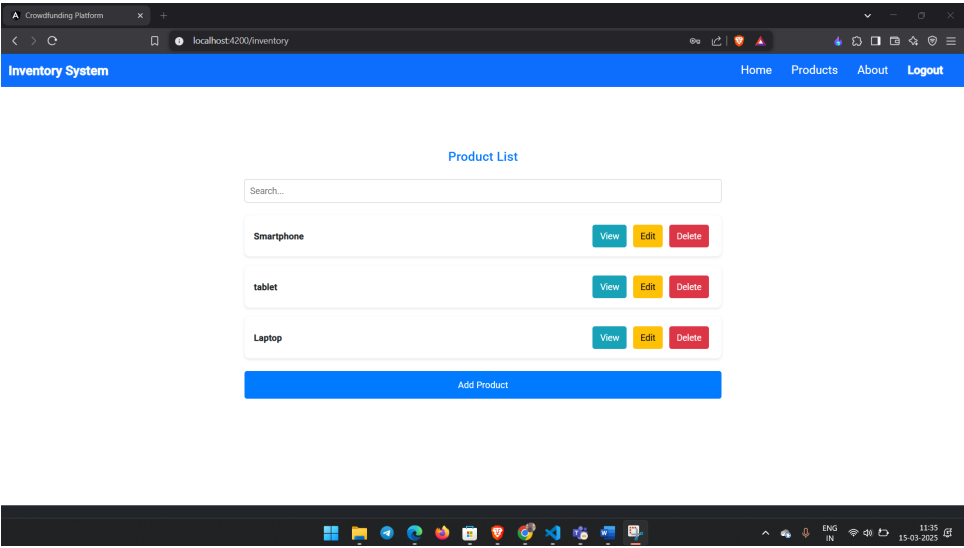


Figure 2: Product List Page After Sign In

15.3 Sign In Page

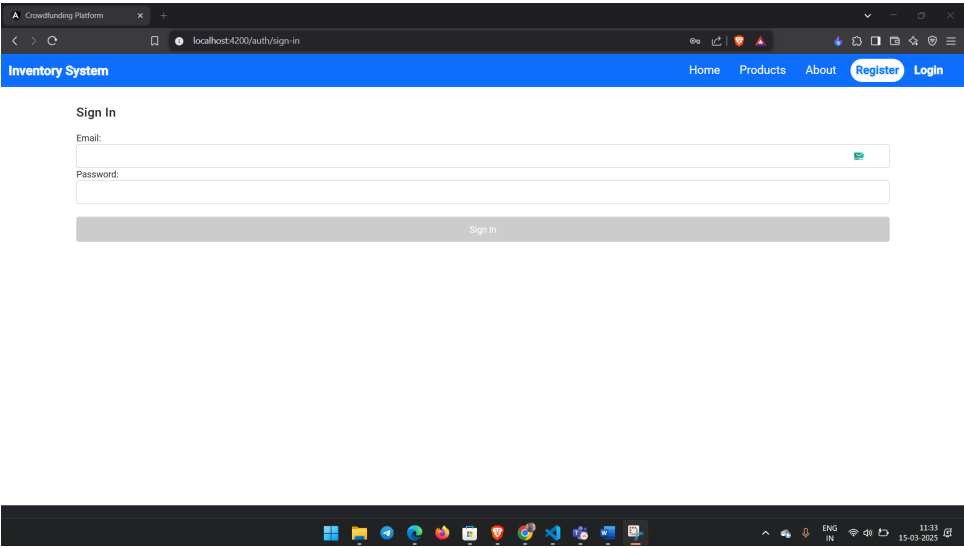


Figure 3: Sign In Page

15.4 Sign In Demo

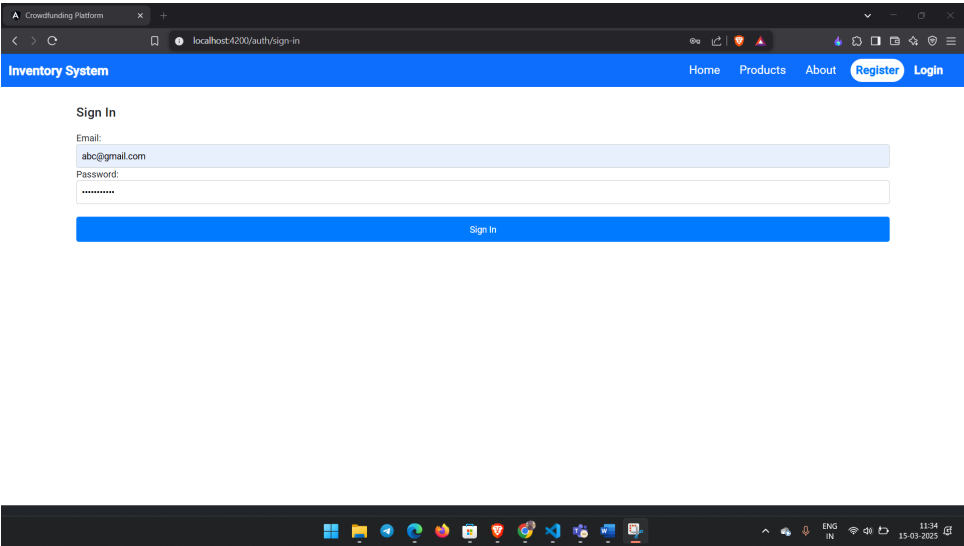


Figure 4: Sign In User Signed In

15.5 Register Page

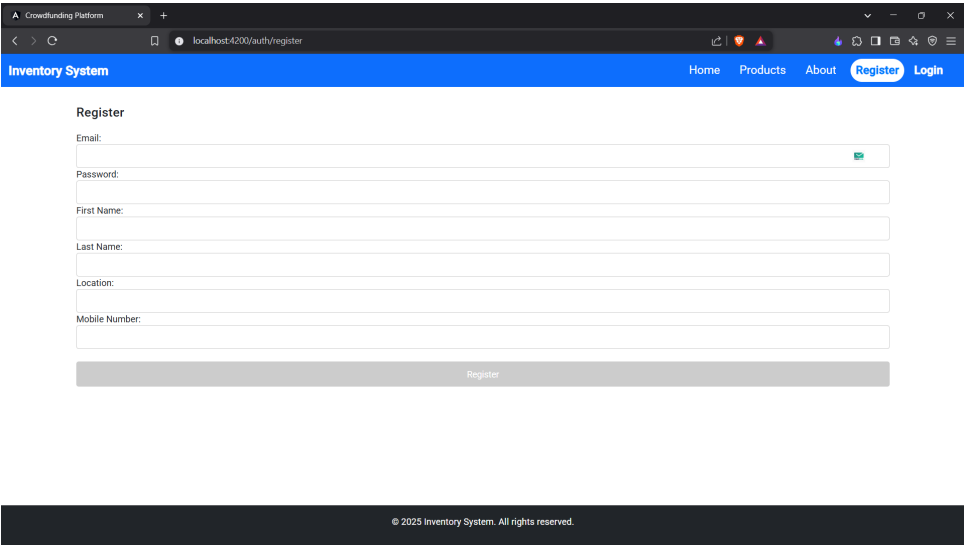


Figure 5: Register Users in Mock Database

15.6 Register Page Demo

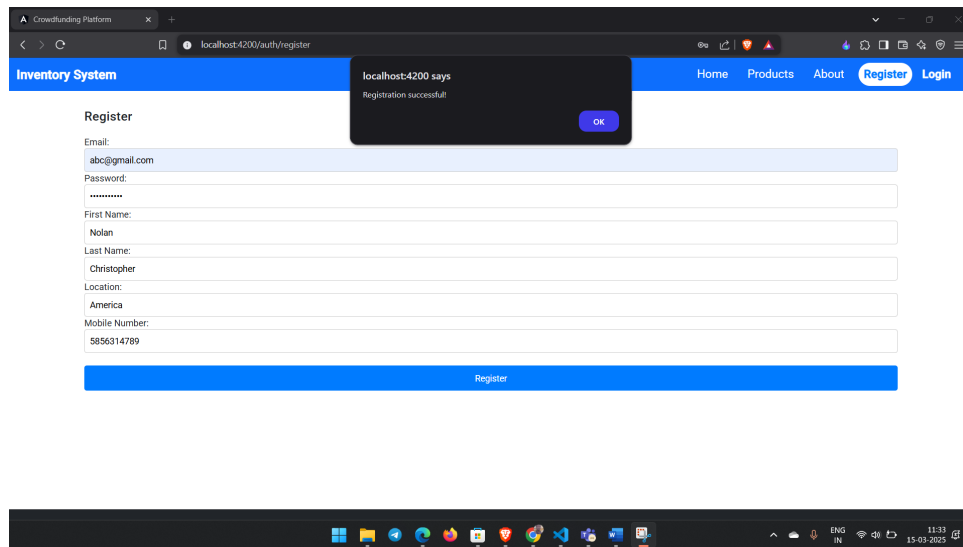


Figure 6: Register Users in Mock Database

15.7 Add Product

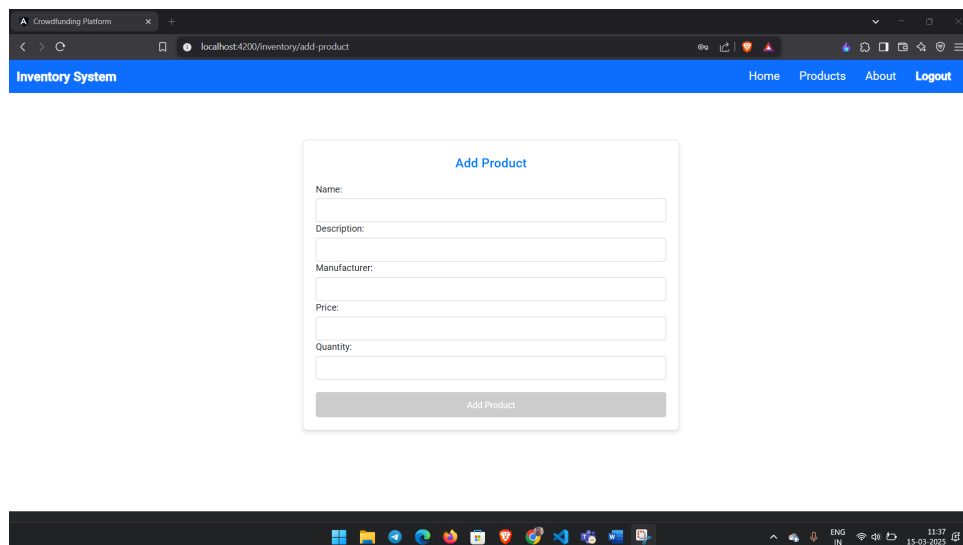


Figure 7: Add New Products

15.8 Add Product Successfull

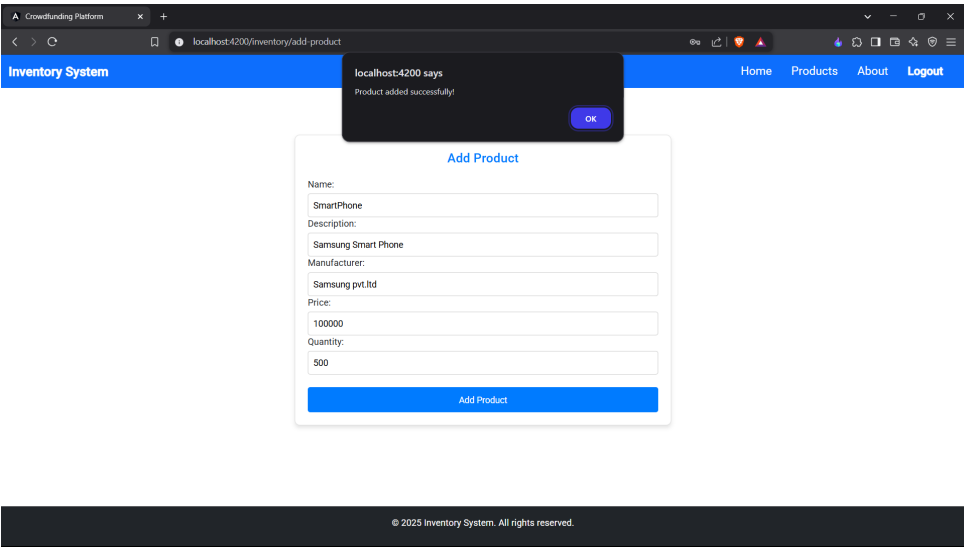


Figure 8: Shows All Successfully Updated Product

15.9 Search Bar

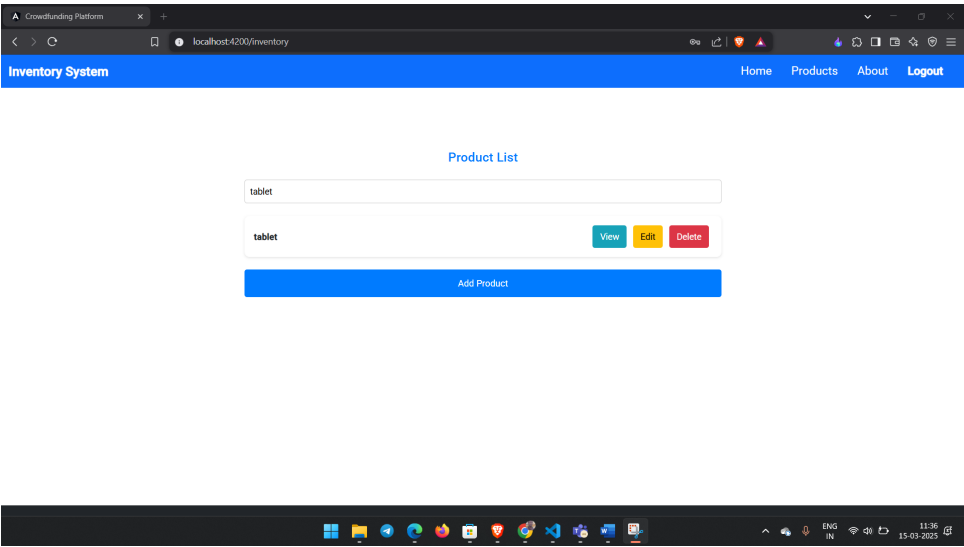
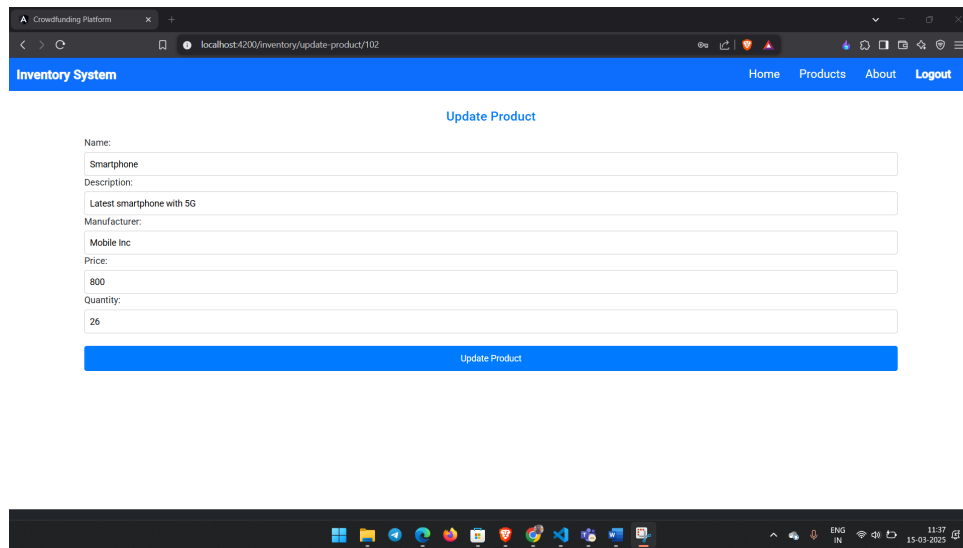


Figure 9: Search Products

15.10 Edit Products



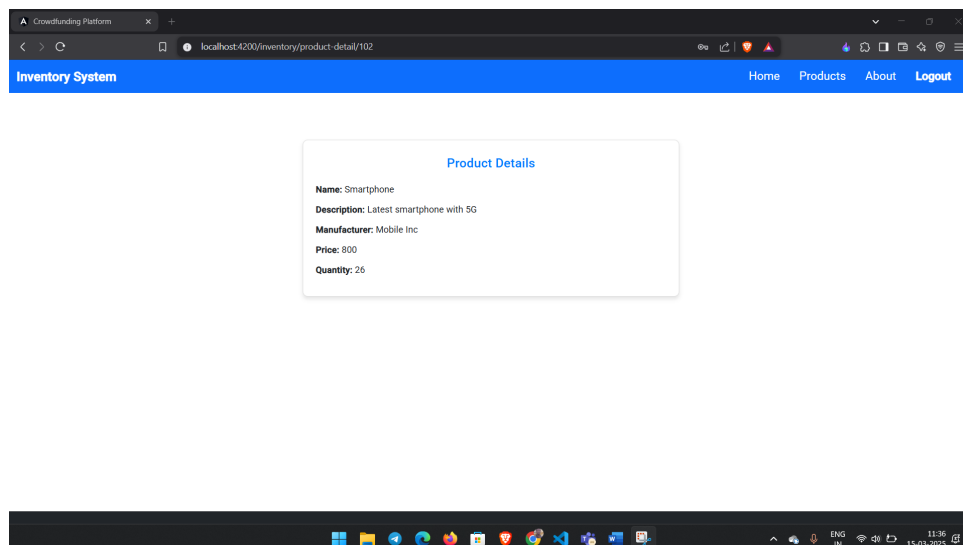
The screenshot shows a web browser window with the URL `localhost:4200/inventory/update-product/102`. The page has a blue header with the text "Inventory System" and navigation links for "Home", "Products", "About", and "Logout". The main content area is titled "Update Product" and contains a form with the following fields:

- Name:
- Description:
- Manufacturer:
- Price:
- Quantity:

At the bottom of the form is a blue button labeled "Update Product".

Figure 10: Edit Product Page

15.11 View Page



The screenshot shows a web browser window with the URL `localhost:4200/inventory/product-detail/102`. The page has a blue header with the text "Inventory System" and navigation links for "Home", "Products", "About", and "Logout". The main content area is titled "Product Details" and contains a card with the following information:

- Name:** Smartphone
- Description:** Latest smartphone with 5G
- Manufacturer:** Mobile Inc
- Price:** 800
- Quantity:** 26

Figure 11: View Product Details Page

16 Application Endpoints

- Angular Frontend: <http://localhost:4200/>
- JSON Server API:
 - Users Endpoint: <http://localhost:3000/users>
 - Products Endpoint: <http://localhost:3000/products>

17 API Services

17.1 Example API Calls

Get All Products:

```
this.dataService.getProducts().subscribe((data) => {  
  this.products = data;  
});
```

Add a Product:

```
this.dataService.addProduct(product).subscribe((response) => {  
  console.log('Product added:', response);  
});
```

18 Authentication Flow

- The **AuthService** manages user authentication.
- **Login Required** for deleting a product.
- **Redirection to Login Page** if an unauthenticated user tries to delete.

19 Dependencies

The system uses:

- **Angular 15** (Core Framework)
- **RxJS** (Reactive Programming)
- **Bootstrap 5.3.3** (Styling)
- **Jest** (Testing)
- **JSON Server** (Mock API)

20 Issues & Risks

20.1 Potential Issues

1. **Security** – No actual authentication backend. We are using Mock API.
2. **Performance** – Large product lists may affect frontend performance.

20.2 Future Improvements

- Integrate with a real database (e.g., Firebase, MongoDB, SQL).
- Implement role-based access control.
- Improve UI/UX with better design elements.