Technical Design Document

Angular Product Inventory Management

ARIJIT BASAK

March 16, 2025

Report
Prepared By: Arijit Basak
Date: March 16, 2025

Contents

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 Functional Requirements

4.1 User Story US_01: Welcome Page

- Default landing page with navigation links: Products, About, Sign In, and Register.
- Product List page includes a search bar, action buttons (View/Edit/Delete), and an Add Product button that redirects unauthenticated users to Sign In.
- Persistent footer displaying "© 2024 Products Inventory" on all pages.

4.2 User Story US_02: Logged-in User Actions

- Authenticated users can perform CRUD operations (Create, Read, Update, Delete) on products.
- Navigation updates to show Products, About, and Logout links.
- Forms include real-time validation and mandatory field checks.

5 System Requirements

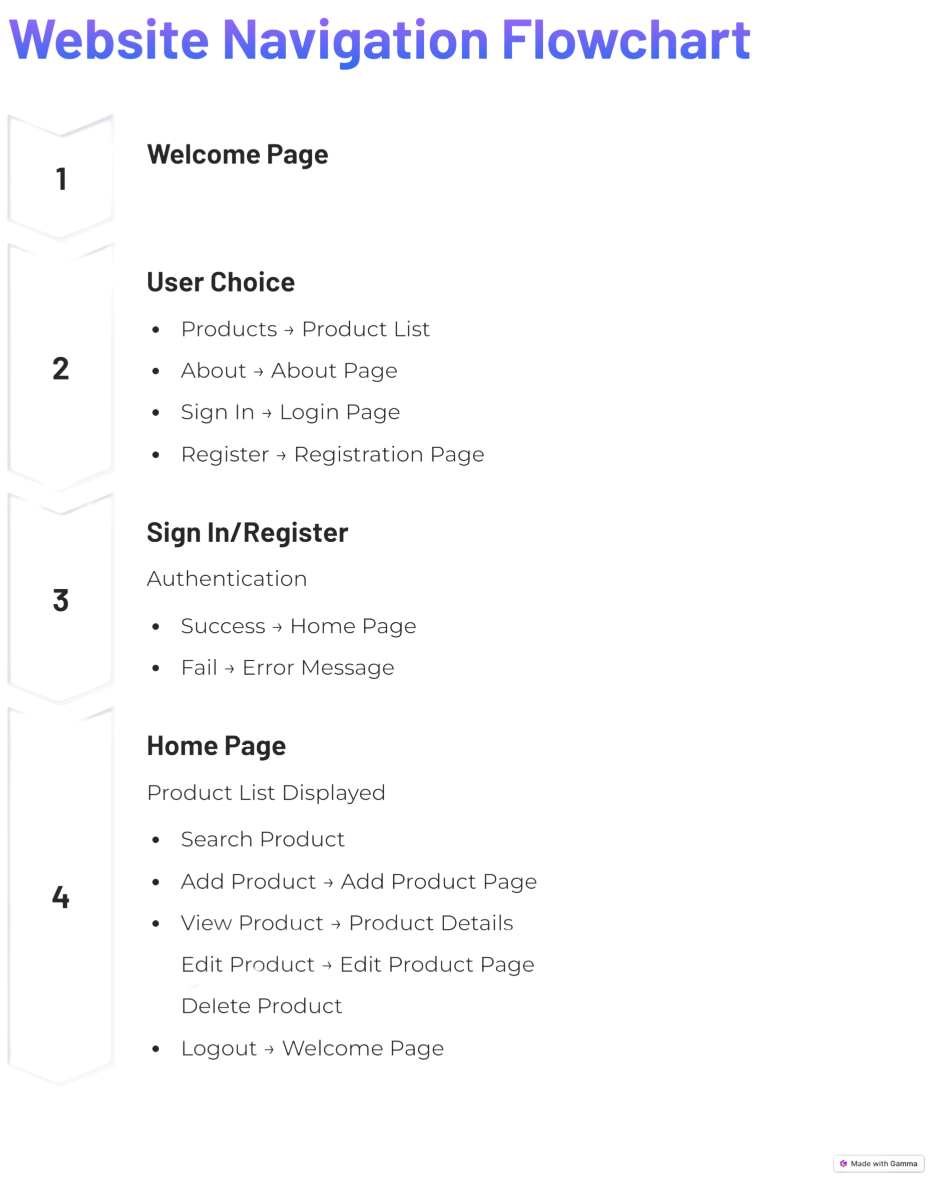
5.1 Development System Specifications

- **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)

6 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.

7 Application Navigation Flow



Website Navigation Flowchart

1 **Welcome Page**

2 **User Choice**

- Products → Product List
- About → About Page
- Sign In → Login Page
- Register → Registration Page

3 **Sign In/Register**

Authentication

- Success → Home Page
- Fail → Error Message

4 **Home Page**

Product List Displayed

- Search Product
- Add Product → Add Product Page
- View Product → Product Details

  Edit Product → Edit Product Page

  Delete Product
- Logout → Welcome Page

Figure 1: Website Navigation Flowchart

The flowchart above illustrates the user journey, starting from the Welcome Page and navigating through authentication, product management, and logout functionalities.

8 Application Design

8.1 Modules & Components

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

8.1.1 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.

9 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**.

10 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.

11 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.

12 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;
  });
}
```

13 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.

14 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.

15 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.

16 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
```

17 Constraints

- All input fields (e.g., email, password, product details) require valid data and display error messages for invalid entries.
- Users must log in to perform inventory actions; unauthenticated users are redirected to the Sign In page.

18 User Interface Screenshots

Below are some key UI pages of the application.

18.1 Product List Page

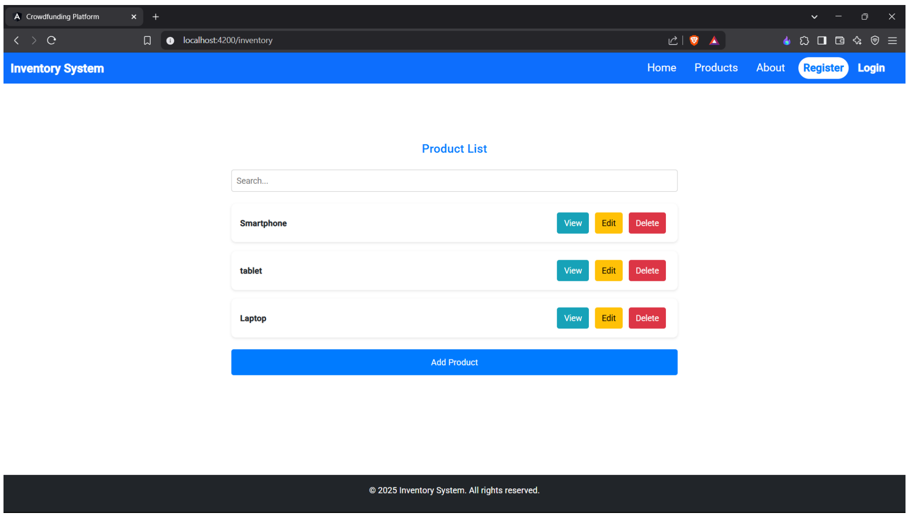


Figure 2: Product List Page

18.2 Product List Page Upon Sign In

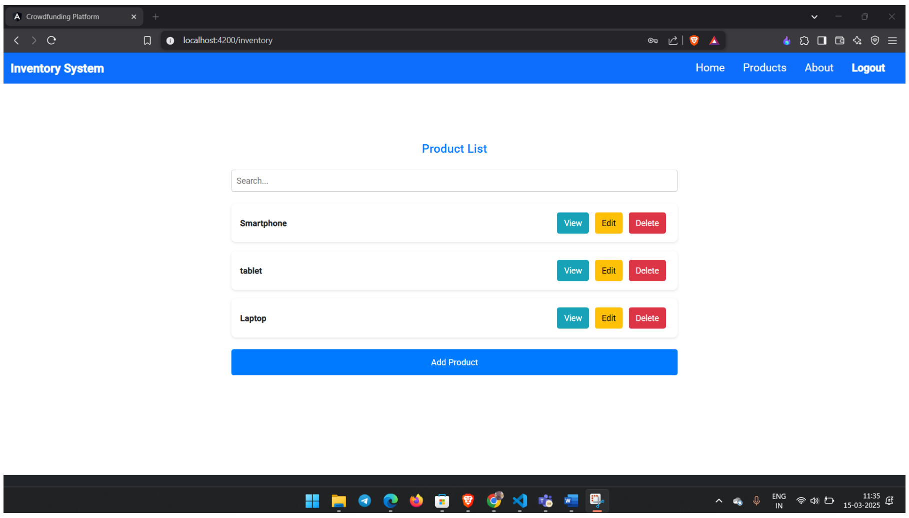


Figure 3: Product List Page After Sign In

18.3 Sign In Page

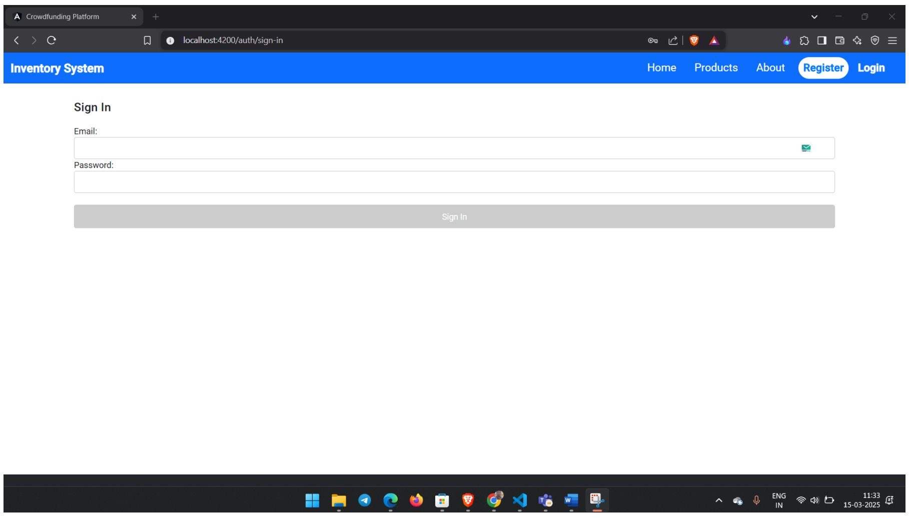
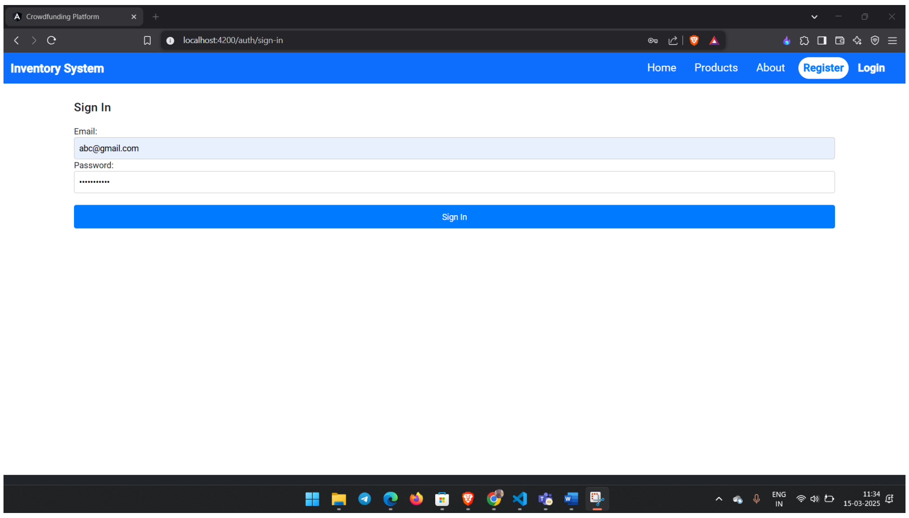


Figure 4: Sign In Page

18.4 Sign In Demo



Crowdfunding Platform

localhost:4200/auth/sign-in

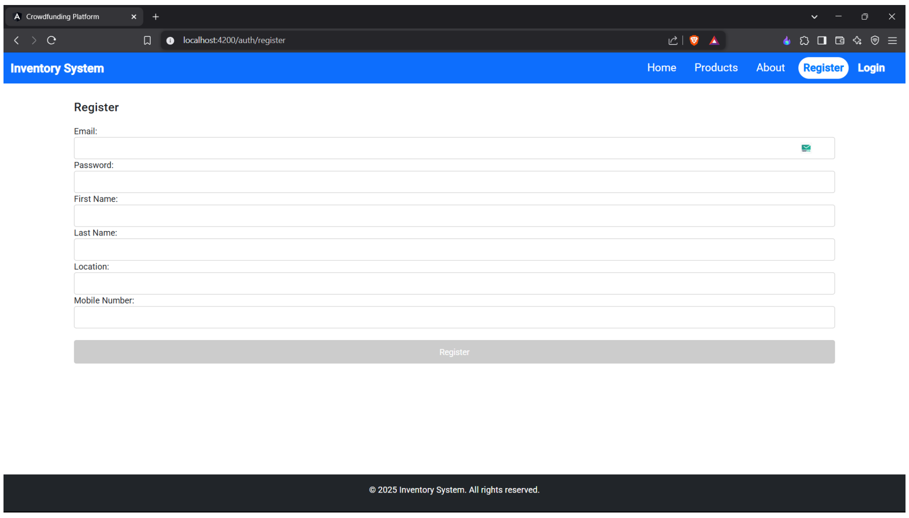
Sign In

Email:
abc@gmail.com

Password:
••••••••••••

Figure 5: Sign In User Signed In

18.5 Register Page



Crowdfunding Platform

localhost:4200/auth/register

Register

Email:

Password:

First Name:

Last Name:

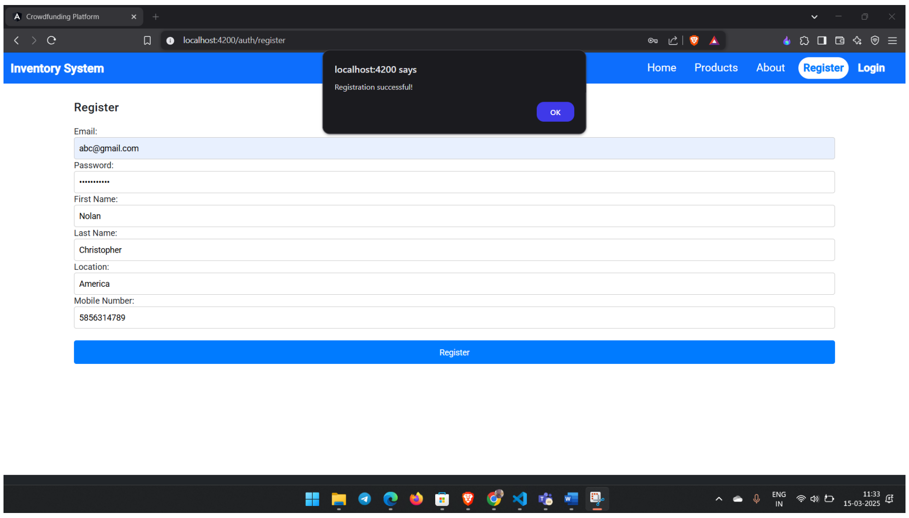
Location:

Mobile Number:

© 2025 Inventory System. All rights reserved.

Figure 6: Register Users in Mock Database

18.6 Register Page Demo


Crowdfunding Platform

localhost:4200/auth/register

localhost:4200 says

Registration successful!

Register

Email: abc@gmail.com

Password: •••••••••••

First Name: Nolan

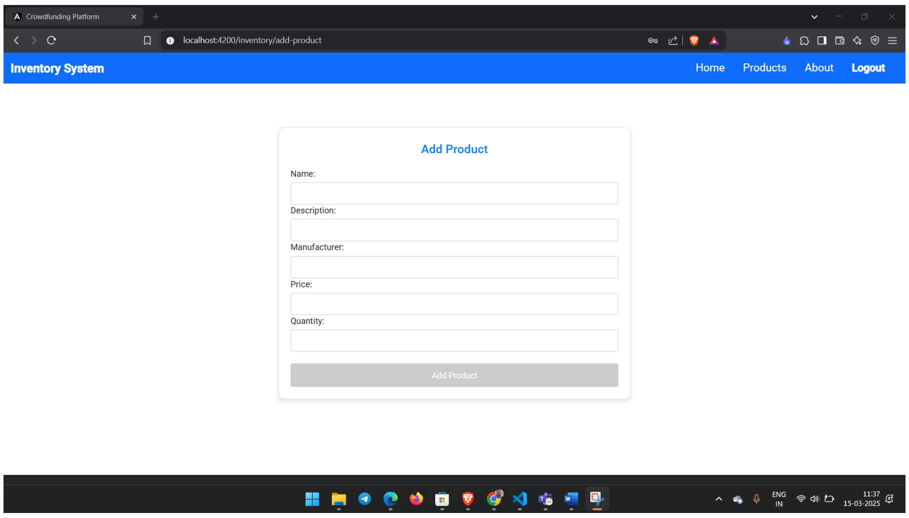
Last Name: Christopher

Location: America

Mobile Number: 5856314789

Figure 7: Register Users in Mock Database

18.7 Add Product


Crowdfunding Platform

localhost:4200/inventory/add-product

Add Product

Name:

Description:

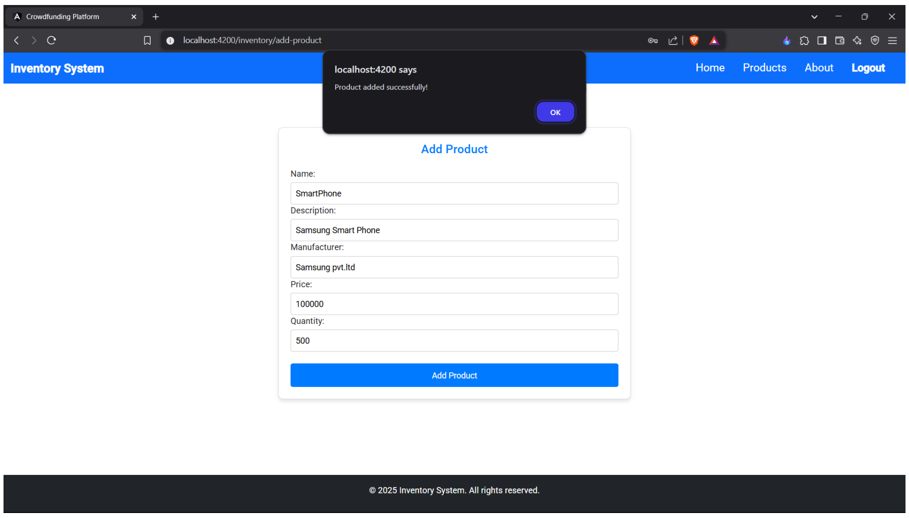
Manufacturer:

Price:

Quantity:

Figure 8: Add New Products

18.8 Add Product Successfull



Crowdfunding Platform

localhost:4200/inventory/add-product

Inventory System

Home Products About Logout

localhost:4200 says

Product added successfully!

OK

Add Product

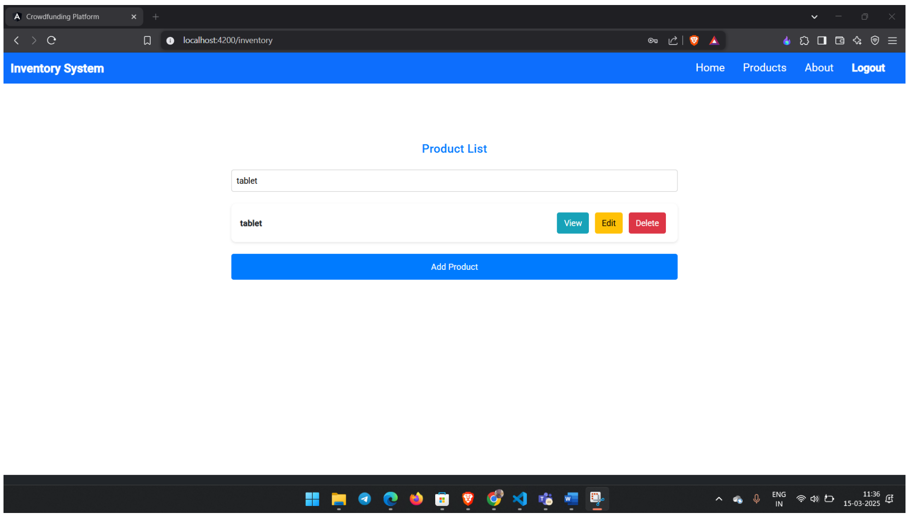
Name:	SmartPhone
Description:	Samsung Smart Phone
Manufacturer:	Samsung pvt.ltd
Price:	100000
Quantity:	500

Add Product

© 2025 Inventory System. All rights reserved.

Figure 9: Shows All Successfully Updated Product

18.9 Search Bar



Crowdfunding Platform

localhost:4200/inventory

Inventory System

Home Products About Logout

Product List

tablet

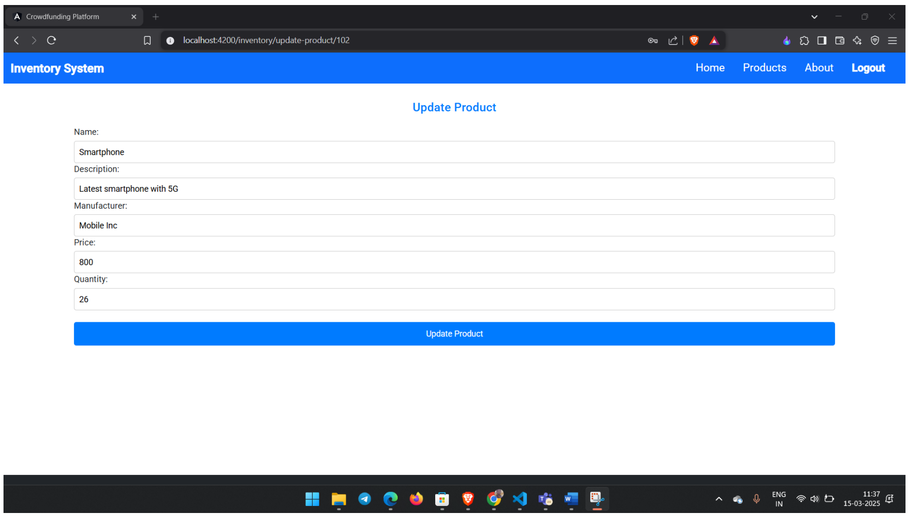
tablet View Edit Delete

Add Product

ENG IN 11:36 15-03-2025

Figure 10: Search Products

18.10 Edit Products

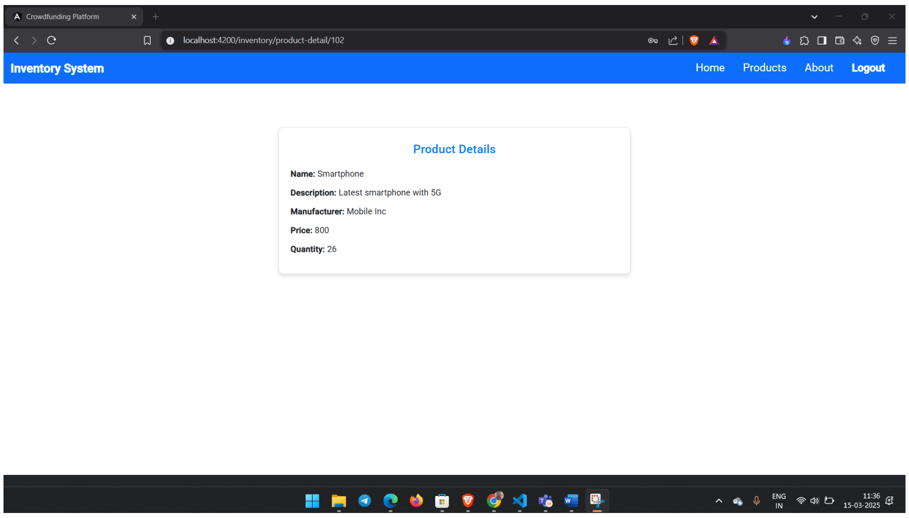


Update Product

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

Figure 11: Edit Product Page

18.11 View Page



Product Details

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

Figure 12: View Product Details Page

19 Application Endpoints

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

20 API Services

20.1 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);
});
```

21 Authentication Flow

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

22 Dependencies

The system uses:

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

23 Issues & Risks

23.1 Potential Issues

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

23.2 Future Improvements

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