

# PRODUCT INVENTORY MANAGEMENT

IIHT

Time To Complete: 10 to 12 hr

## CONTENTS

---

|     |  |    |
|-----|--|----|
| 1   | Project Abstract                                       | 3  |
| 2   | Problem Statement                                      | 3  |
| 3   | Proposed Product Inventory Management System Wireframe | 4  |
| 3.1 | Welcome Page   | 4  |
| 3.2 | Screenshots  | 5  |
| 4   | Business-Requirement:                                  | 15 |
| 5   | Constraints  | 21 |
| 6   | Mandatory Assessment Guidelines                        | 22 |

## 1 PROJECT ABSTRACT

---

In the ever-evolving field of retail and business management, there is a growing necessity for digital platforms that facilitate efficient inventory tracking, enhance product management capabilities, and empower businesses to optimize their operations. With this vision in mind, the CEO of a product management startup assigns a team of developers with the task of creating a **Product Inventory Management System** using Angular.

Your task is to develop a digital solution that allows businesses to manage their product inventory efficiently by adding, updating, deleting, and listing products. The system should enable users to organize products by categories, update stock quantities, and manage product details such as descriptions, prices, and manufacturers. Additionally, the platform should allow administrators to securely log in, perform inventory actions, and track their inventory in real time.

## 2 PROBLEM STATEMENT

---

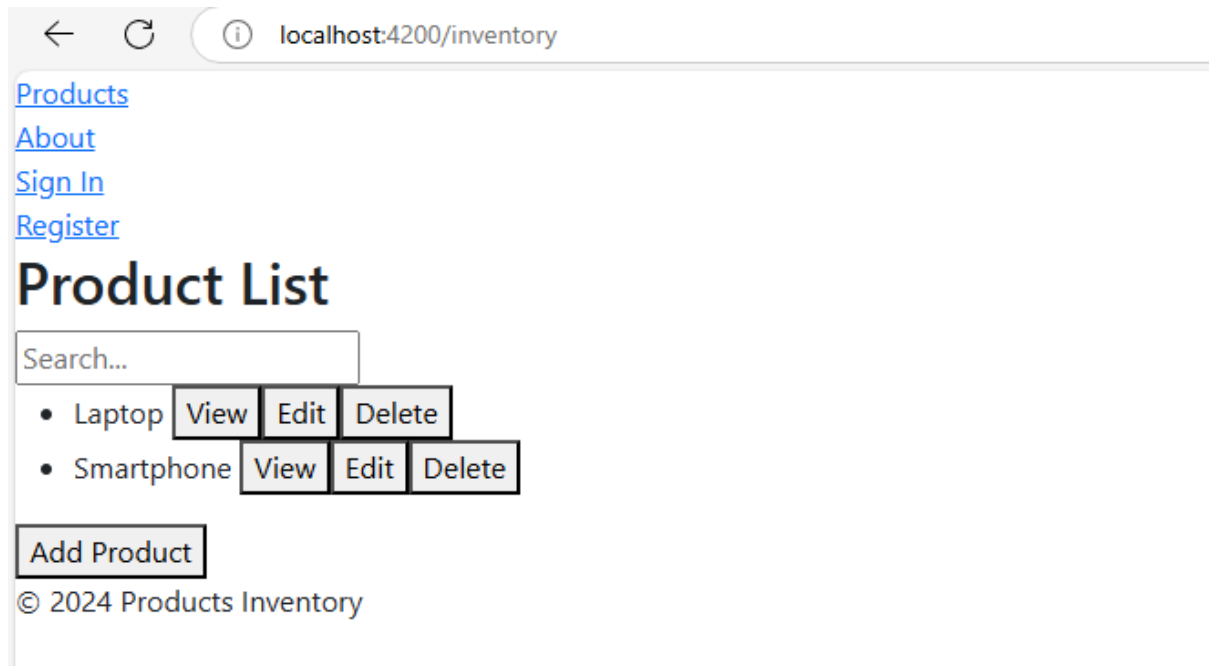
**“Product Inventory Management System”** is a Single Page Application (SPA) designed to enhance inventory management for businesses by simplifying the process of managing products and stock. This system enables businesses to efficiently add, update, delete, list, and search for product records. Users can also categorize products, update stock quantities, and view detailed product information such as descriptions, prices, and manufacturers.

### 3 PROPOSED PRODUCT INVENTORY MANAGEMENT SYSTEM WIREFRAME

---

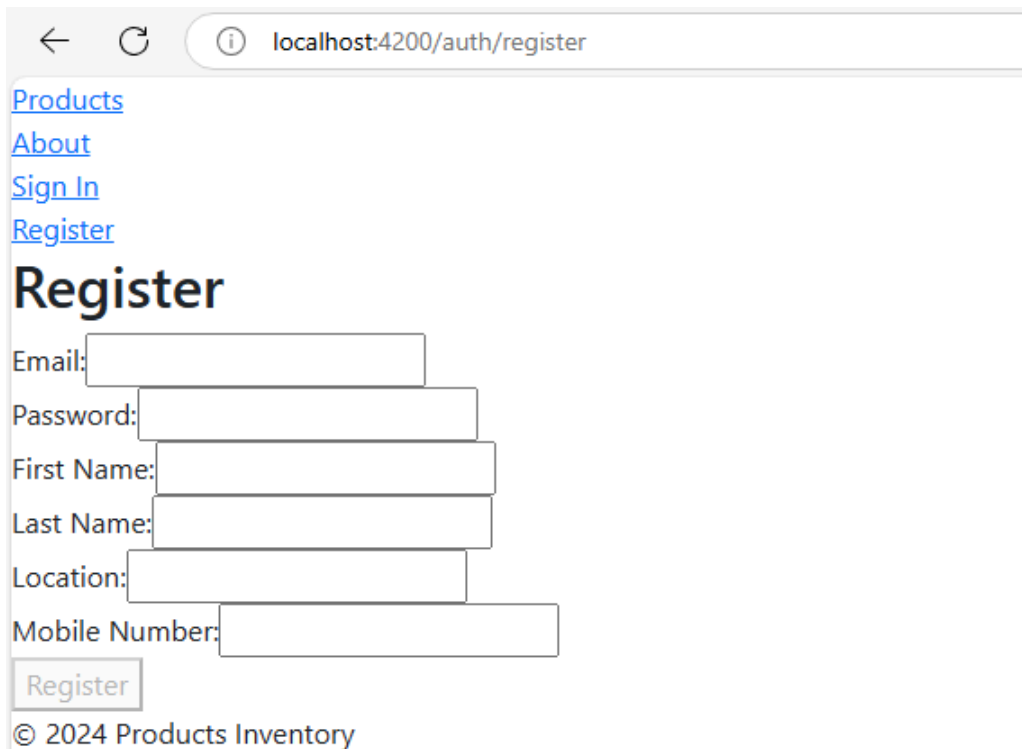
UI needs improvisation and modification as per given use case and to make test cases passed.

#### 3.1 WELCOME PAGE

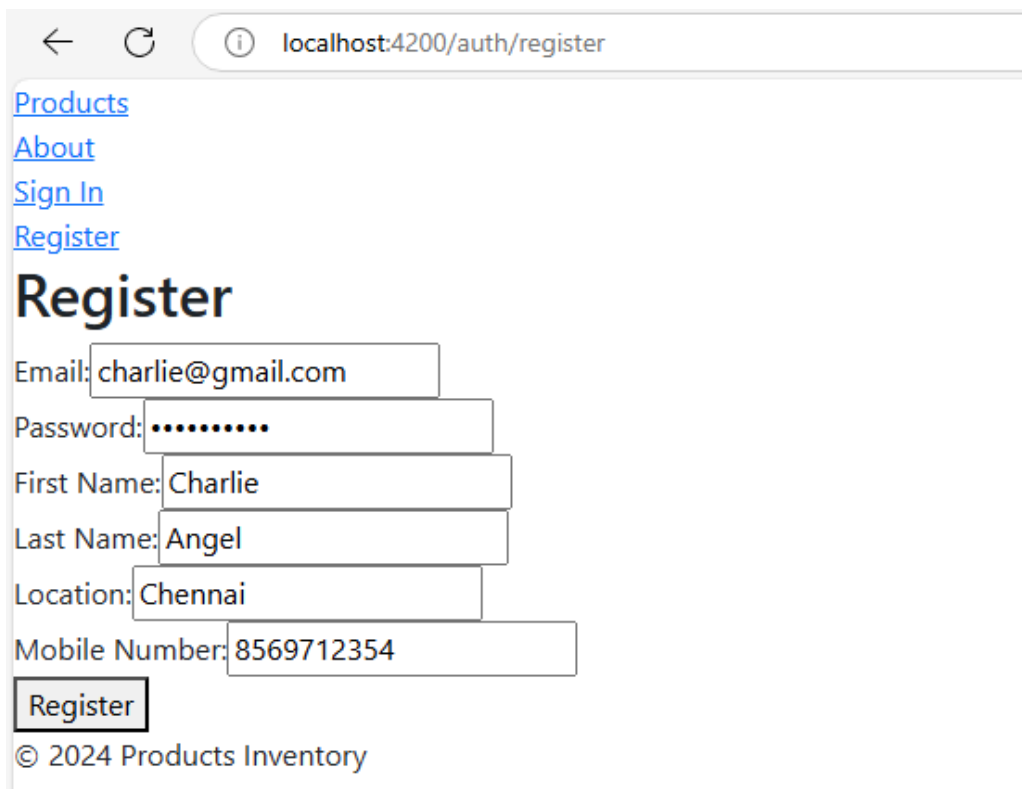


## 3.2 SCREENSHOTS

\*\*\*Register\*\*\*



A screenshot of a web browser showing the 'Register' page. The browser's address bar displays 'localhost:4200/auth/register'. On the left side, there is a vertical menu with links: 'Products', 'About', 'Sign In', and 'Register'. The main heading is 'Register'. Below it, there are six empty input fields for 'Email:', 'Password:', 'First Name:', 'Last Name:', 'Location:', and 'Mobile Number:'. A 'Register' button is positioned below the fields. At the bottom, the footer text reads '© 2024 Products Inventory'.



A second screenshot of the same 'Register' page, but with the form fields filled out. The 'Email:' field contains 'charlie@gmail.com', 'Password:' contains '.....', 'First Name:' contains 'Charlie', 'Last Name:' contains 'Angel', 'Location:' contains 'Chennai', and 'Mobile Number:' contains '8569712354'. The 'Register' button is now highlighted with a black border. The rest of the page, including the browser address bar and footer, remains the same.

← ↻ ⓘ localhost:4200/auth/register

[Products](#)  
[About](#)  
[Sign In](#)  
[Register](#)

## Register

Email:

Password:

First Name:

Last Name:

Location:

Mobile Number:

© 2024 Products Inventory

**localhost:4200 says**  
Registration successful!

\*\*\*Sign In\*\*\*

← ↻ ⓘ localhost:4200/auth/sign-in

[Products](#)  
[About](#)  
[Sign In](#)  
[Register](#)

## Sign In

Email:

Password:

© 2024 Products Inventory

\*\*\* Already added some users, kindly use their credentials in db.json file \*\*\*

← ↻ ⓘ localhost:4200/auth/sign-in

[Products](#)  
[About](#)  
[Sign In](#)  
[Register](#)

## Sign In

Email:

Password:

© 2024 Products Inventory

\*\*\*Signed In as a User\*\*\*

← ↻ ⓘ localhost:4200/inventory

[Products](#)  
[About](#)

## Product List

- Laptop
- Smartphone

© 2024 Products Inventory

\*\*\*Add Product\*\*\*

← ↻ ⓘ localhost:4200/inventory/add-product

[Products](#)  
[About](#)

Logout

## Add Product

Name:

Description:

Manufacturer:

Price:

Quantity:

Add Product

© 2024 Products Inventory

\*\*\*Already added some products with their details\*\*\*

← ↻ ⓘ localhost:4200/inventory/add-product

[Products](#)  
[About](#)

Logout

## Add Product

Name:

Description:

Manufacturer:

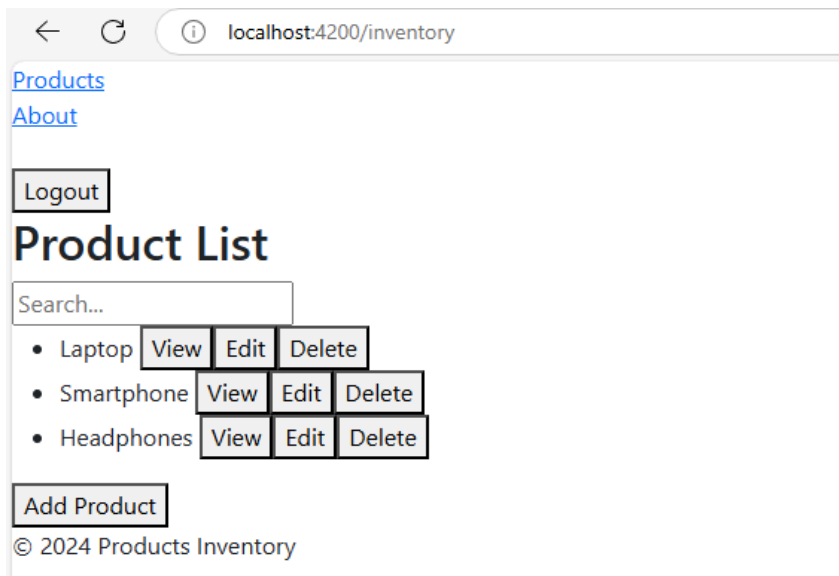
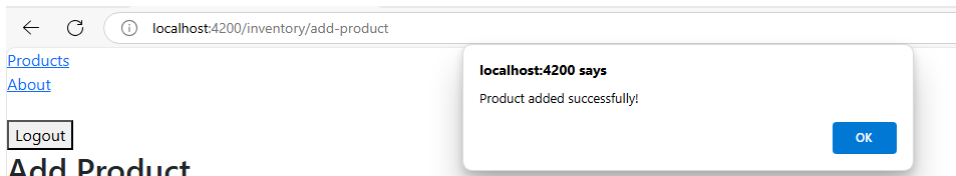
Price:

Quantity:

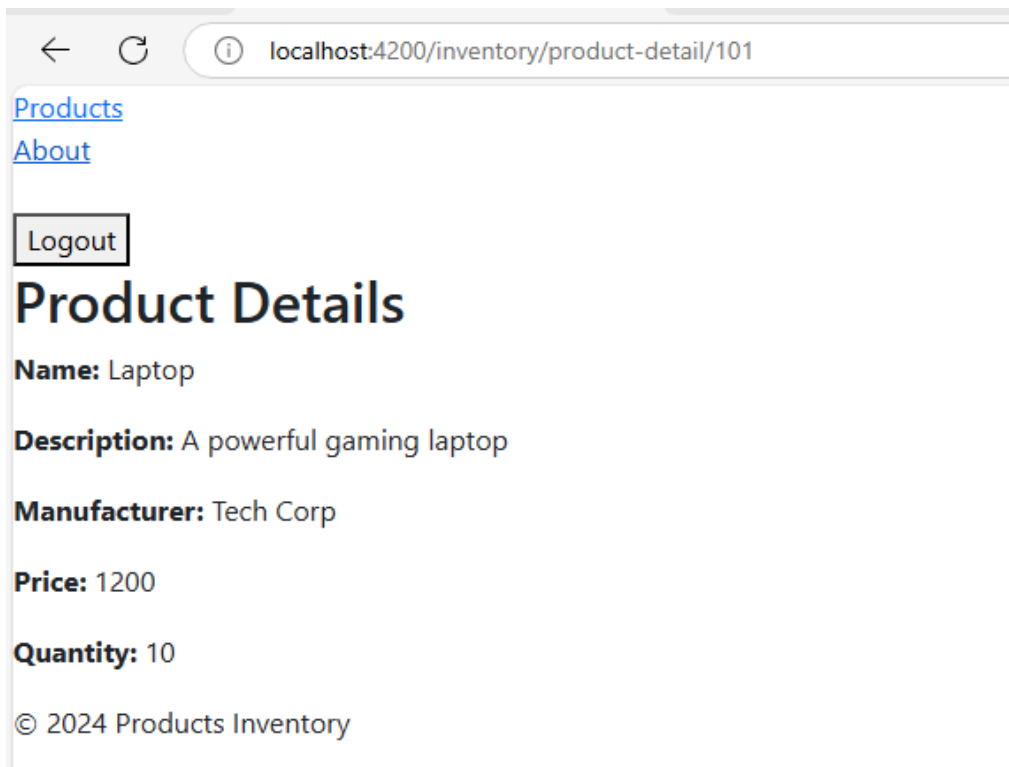
Add Product

© 2024 Products Inventory





**\*\*\*View Product\*\*\***



\*\*\*Edit Product\*\*\*

← ↻ ⓘ localhost:4200/inventory/update-product/101

[Products](#)  
[About](#)

Logout

## Update Product

Name:

Description:

Manufacturer:

Price:

Quantity:

Update Product

© 2024 Products Inventory

← ↻ ⓘ localhost:4200/inventory/update-product/101

[Products](#)  
[About](#)

Logout

## Update Product

Name:

Description:

Manufacturer:

Price:

Quantity:

Update Product

© 2024 Products Inventory

← ↻ ⓘ localhost:4200/inventory/update-product/101

[Products](#)  
[About](#)

Logout

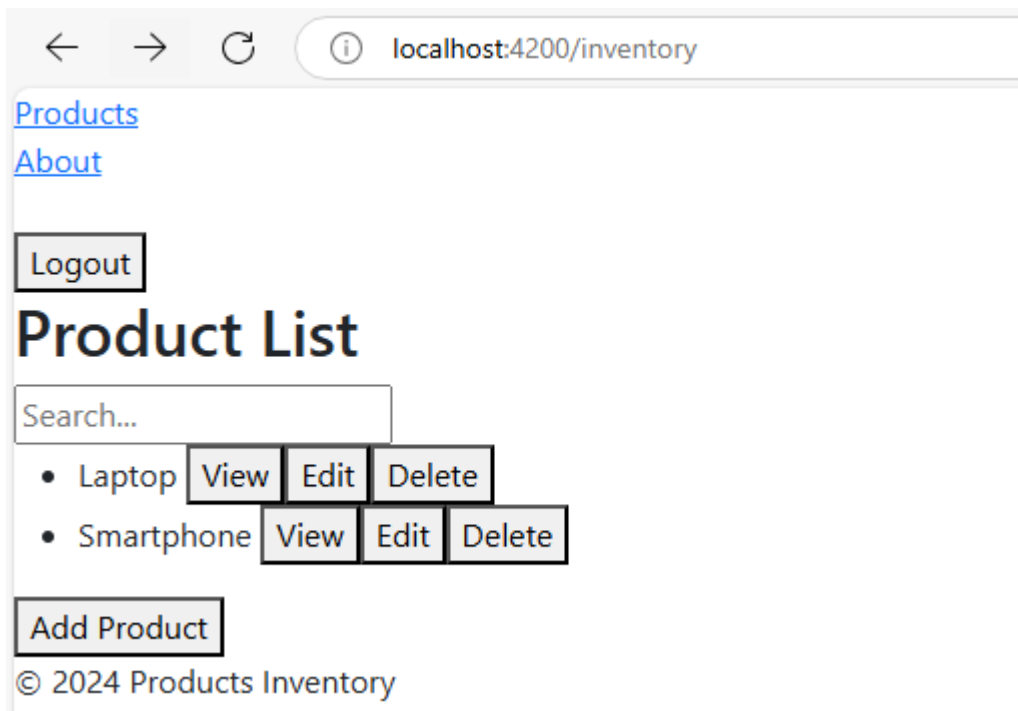
## Update Product

Name:

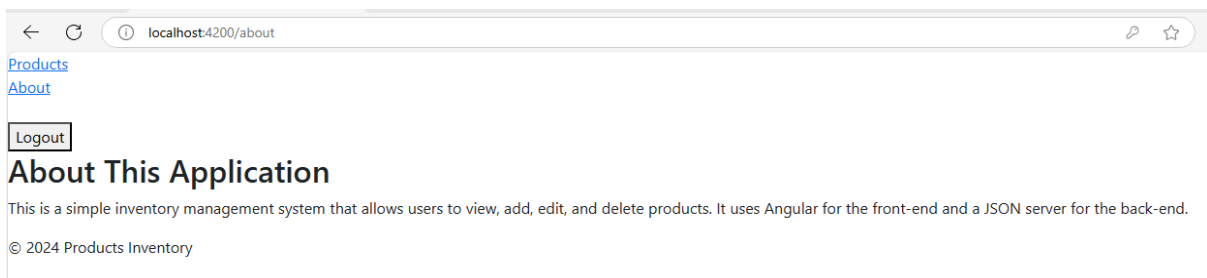
localhost:4200 says  
Product updated successfully!

OK

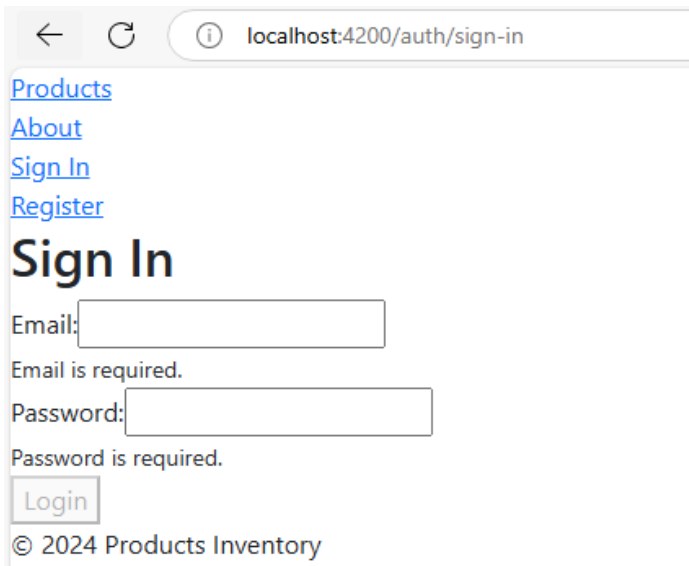
### \*\*\*Delete Product\*\*\*



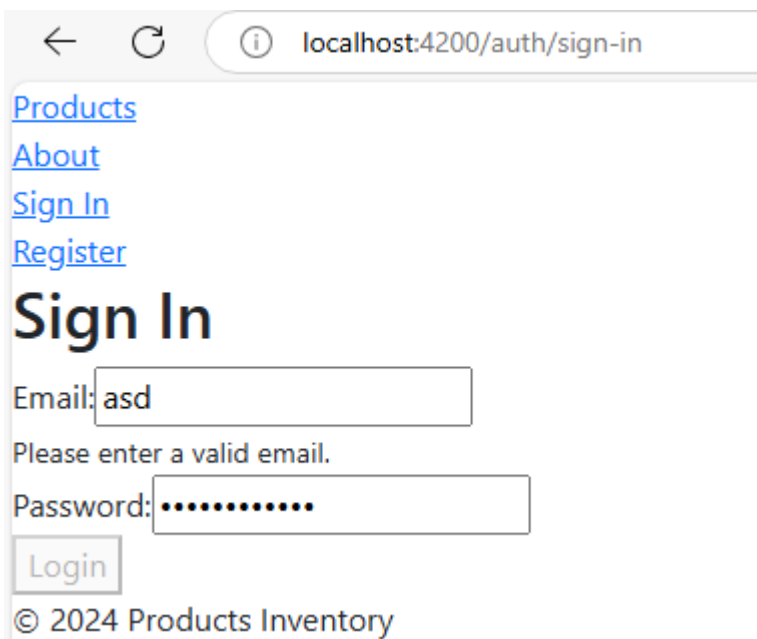
### \*\*\*About\*\*\*



\*\*\*Validations\*\*\*



A screenshot of a web browser at the URL `localhost:4200/auth/sign-in`. The page has a navigation menu with links for [Products](#), [About](#), [Sign In](#), and [Register](#). The main heading is "Sign In". Below it, there are two input fields: "Email:" and "Password:". The "Email:" field is empty, and below it is the message "Email is required.". The "Password:" field is also empty, and below it is the message "Password is required.". At the bottom of the form is a "Login" button. The footer text is "© 2024 Products Inventory".



A screenshot of the same web browser at the URL `localhost:4200/auth/sign-in`. The page content is identical to the first screenshot, but the input fields now contain text. The "Email:" field contains the text "asd", and below it is the message "Please enter a valid email.". The "Password:" field contains a series of dots ".....", and below it is the message "Password is required.". The "Login" button and footer text remain the same.

← ↻ ⓘ localhost:4200/auth/register

[Products](#)  
[About](#)  
[Sign In](#)  
[Register](#)

## Register

Email:   
Email is required.

Password:   
Password is required.

First Name:   
First name is required.

Last Name:   
Last name is required.

Location:   
Location is required.

Mobile Number:   
Mobile number is required.

© 2024 Products Inventory

← ↻ ⓘ localhost:4200/inventory/add-product

[Products](#)  
[About](#)

## Add Product

Name:   
Product name is required.

Description:   
Description is required.

Manufacturer:   
Manufacturer is required.

Price:   
Price is required.

Quantity:   
Quantity is required.

© 2024 Products Inventory

[←](#) [↻](#) [localhost:4200/auth/register](#)

[Products](#)  
[About](#)  
[Sign In](#)  
[Register](#)

## Register

Email:

Please enter a valid email.

Password:

Password must be at least 6 characters long.

First Name:

Last Name:

Location:

Mobile Number:

Enter a valid 10-digit mobile number.

© 2024 Products Inventory

## 4 BUSINESS-REQUIREMENT:

As an application developer, develop the Product Inventory Management System (Single Page App) with below guidelines:

| User Story # | User Story Name | User Story   |
|--------------|-----------------|--|
| US_01        | Welcome Page    | <p>As a user I should be able to visit the welcome page as the default page.</p> <h3>Angular – Product Inventory Management</h3> <ul style="list-style-type: none"><li>◆ App Component (<b>app.component.ts</b>, <b>app.component.html</b>)</li></ul> <p><b>Purpose:</b><br/>Defines the main layout of the application.</p> <p><b>Structure to implement:</b></p> <ul style="list-style-type: none"><li>● Use a <b>&lt;app-header&gt;</b> element for the top nav bar.</li><li>● Use a <b>&lt;main&gt;</b> tag containing a <b>&lt;router-outlet&gt;</b> for rendering routed components.</li><li>● Use a <b>&lt;app-footer&gt;</b> at the bottom.</li></ul> <p><b>Responsibilities:</b></p> <ul style="list-style-type: none"><li>● Acts as a wrapper layout for the entire app.</li></ul> <p><b>Title Property:</b></p> <ul style="list-style-type: none"><li>● Add a <b>title</b> property with the value: "Product Inventory System"</li></ul> <ul style="list-style-type: none"><li>◆ Core Module</li></ul> <p><b>auth.guard.ts</b></p> <p><b>Purpose:</b><br/>Protects specific routes by checking if the user is logged in.</p> <p><b>Responsibilities:</b></p> <ul style="list-style-type: none"><li>● Use Angular's <b>CanActivate</b> interface</li><li>● Inject <b>AuthService</b> and <b>Router</b></li><li>● In <b>canActivate()</b>:<ul style="list-style-type: none"><li>○ If the user is logged in → allow access</li><li>○ If not logged in → redirect to <b>/auth/sign-in</b> and deny access</li></ul></li></ul> <p><b>auth.service.ts</b></p> |

|  |  |  |
|--|--|--|
|  |  | <p><b>Purpose:</b><br/>Handles login, logout, registration, and session checks.</p> <p><b>Functions &amp; Responsibilities:</b></p> <p><b>login(email, password):</b></p> <ul style="list-style-type: none"> <li>• Sends a GET request to retrieve all users</li> <li>• Finds a user with matching email and password</li> <li>• If found: <ul style="list-style-type: none"> <li>◦ Stores the user object in localStorage under the key <code>"currentUser"</code></li> <li>◦ Returns true</li> </ul> </li> <li>• If no match: <ul style="list-style-type: none"> <li>◦ Returns false</li> </ul> </li> </ul> <p><b>logout():</b></p> <ul style="list-style-type: none"> <li>• Removes <code>"currentUser"</code> from localStorage</li> </ul> <p><b>isLoggedIn():</b></p> <ul style="list-style-type: none"> <li>• Returns true if a user is stored in localStorage</li> <li>• Used by guards and components to control access</li> </ul> <p><b>register(user)</b></p> <ul style="list-style-type: none"> <li>• Sends a POST request to <code>/users</code></li> <li>• Adds a new user entry in the backend</li> </ul> <p><b>data.service.ts</b></p> <p><b>Purpose:</b><br/>Provides functions to manage products and users from the backend API.</p> <p><b>Functions &amp; Responsibilities:</b></p> <p><b>getUsers()</b></p> <ul style="list-style-type: none"> <li>• Sends a GET request to <code>/users</code></li> <li>• Returns a list of registered users</li> </ul> <p><b>getProducts()</b></p> <ul style="list-style-type: none"> <li>• Sends a GET request to <code>/products</code></li> <li>• Returns the product list</li> </ul> <p><b>getProductById(id)</b></p> <ul style="list-style-type: none"> <li>• Sends a GET request to <code>/products/{id}</code></li> <li>• Returns the details of a single product by ID</li> </ul> <p><b>addProduct(product)</b></p> <ul style="list-style-type: none"> <li>• Sends a POST request to <code>/products</code></li> <li>• Adds a new product to the inventory</li> </ul> <p><b>updateProduct(product)</b></p> <ul style="list-style-type: none"> <li>• Sends a PUT request to <code>/products/{id}</code></li> <li>• Updates the specified product by ID</li> </ul> <p><b>deleteProduct(id)</b></p> |
|--|--|--|



|  |  |  |
|--|--|--|
|  |  | <ul style="list-style-type: none"> <li>• Sends a DELETE request to <code>/products/{id}</code></li> <li>• Removes the specified product by ID</li> </ul> <p><b>core.module.ts</b></p> <p><b>Purpose:</b><br/>Ensures that core services are only registered once and shared throughout the app.</p> <p><b>Responsibilities:</b></p> <ul style="list-style-type: none"> <li>• Import <code>CommonModule</code></li> <li>• Prevent duplicate imports using a constructor check: <ul style="list-style-type: none"> <li>◦ If <code>CoreModule</code> is already loaded, throw an error: <code>'CoreModule is already loaded. Import it in the AppModule only.'</code></li> </ul> </li> </ul> <h2>About &amp; Authentication Module (Register &amp; Sign In)</h2> <h3>AboutComponent</h3> <p><b>Purpose:</b></p> <ul style="list-style-type: none"> <li>• Static informational component about the app.</li> </ul> <p><b>Implementation Details:</b></p> <ul style="list-style-type: none"> <li>• Add a heading using <code>&lt;h2&gt;</code>.</li> <li>• This component is declared in <code>AboutModule</code>, making it lazily loadable via routes (e.g., <code>/about</code>).</li> </ul> <h2>Authentication Module Overview</h2> <h3>AuthModule</h3> <ul style="list-style-type: none"> <li>• Contains routes for <code>register</code> and <code>sign-in</code>.</li> <li>• Uses <code>FormsModule</code> for template-driven forms.</li> <li>• Routes are managed via <code>AuthRoutingModule</code>.</li> </ul> <h3>AuthRoutingModule</h3> <p><b>Purpose:</b><br/>Defines child routes for auth-related components.</p> <p><b>Structure:</b></p> <ul style="list-style-type: none"> <li>• <code>/auth/register</code> → loads <code>RegisterComponent</code></li> </ul> |
|--|--|--|

- `/auth/sign-in` → loads `SignInComponent`

## RegisterComponent

### Purpose:

Provides a **user registration form** with full **client-side validation** before creating a user.

### Structure Breakdown:

- Use a top-level `<form>` with `ngForm`
- Use `<input>` elements bound via `ngModel` for:

- Email
- Password
- First Name
- Last Name
- Location
- Mobile Number

### Validations:

- **Email**
  - Required
  - Must be in valid email format (`email` validator)
  - Shows:
    - *"Email is required"*
    - *"Please enter a valid email"*
- **Password**
  - Required
  - Minimum 6 characters (`minlength="6"`)
  - Shows:
    - *"Password is required"*
    - *"Password must be at least 6 characters long"*
- **First/Last Name**
  - Required
  - Shows *"First name is required"*, etc.
- **Location**
  - Required
- **Mobile Number**
  - Required
  - Must match pattern `^[0-9]{10}$`
  - Shows:
    - *"Mobile number is required"*
    - *"Enter a valid 10-digit mobile number"*

### Submission Logic:

- Use `(ngSubmit)="onRegister(form)"` with a local reference `#registerForm="ngForm"`.

- In `onRegister()`:
  - If `form.valid`:
    - Call `authService.register(form.value)` and subscribe to success
    - Show `alert('Registration successful!')`
  - Else:
    - Show `alert('Please fill out all required fields.')`

## SignInComponent

### Purpose:

Allows existing users to log in using email and password.

### HTML Structure:

#### Display a heading:

- Use an `<h2>` element with the title: “Sign In”

#### Add a form with validation using `ngForm`:

- Bind the form using `#loginForm="ngForm"`
- Handle form submission with `(ngSubmit)="onLogin(loginForm)"`

#### Show error messages:

- Display a paragraph block if `errorMessage` exists.
- Message appears above the form inputs in a `div` with class `error-message`.

#### Create an Email field section:

- Label: “Email:”
- Input:
  - `type="email"`
  - Bound to `ngModel`
  - Marked as `required` and `email` validated
- Error messages:
  - If touched and invalid:
    - “Email is required.”
    - “Please enter a valid email.”

#### Create a Password field section:

- Label: “Password:”
- Input:
  - `type="password"`
  - Bound to `ngModel`

|  |  |   |
|--|--|---|
|  |  | <ul style="list-style-type: none"> <li>○ Marked as <b>required</b></li> <li>● Error message: <ul style="list-style-type: none"> <li>○ If touched and invalid: <ul style="list-style-type: none"> <li>■ "Password is required."</li> </ul> </li> </ul> </li> </ul> <p><b>Add a Login button:</b></p> <ul style="list-style-type: none"> <li>● Button type: <b>submit</b></li> <li>● Disabled if <b>loginForm.invalid</b> is true</li> <li>● Triggers login when clicked</li> </ul> <p><b>Login Flow:</b></p> <ul style="list-style-type: none"> <li>● On valid form submission: <ul style="list-style-type: none"> <li>○ Calls <b>authService.login(email, password)</b></li> <li>○ If success → Navigate to <b>/inventory</b></li> <li>○ If failed → Shows <b>errorMessage</b></li> </ul> </li> </ul> <p><b>AuthService Integration</b></p> <p><b>Internally Used By:</b></p> <ul style="list-style-type: none"> <li>● <b>RegisterComponent</b> → calls <b>.register(user)</b></li> <li>● <b>SignInComponent</b> → calls <b>.login(email, password)</b></li> </ul> <p><b>Authentication Handling:</b></p> <ul style="list-style-type: none"> <li>● Stores the logged-in user in <b>localStorage</b> under <b>"currentUser"</b> key.</li> <li>● Authenticated users are allowed through <b>AuthGuard</b> on protected routes.</li> </ul> <p><b>Product Inventory Management – UI &amp; Behavior Structure</b></p> <p><b>InventoryModule</b></p> <p><b>Purpose:</b></p> <ul style="list-style-type: none"> <li>● Main module encapsulating product management features.</li> </ul> <p><b>Declares:</b></p> <ul style="list-style-type: none"> <li>● <b>ProductListComponent</b></li> <li>● <b>ProductDetailComponent</b></li> </ul> |
|--|--|---|

- `AddProductComponent`
- `UpdateProductComponent`

#### Imports:

- `FormsModule` → Template-driven form (add)
- `ReactiveFormsModule` → Reactive form (update)
- `SharedModule` → Reusable UI
- `InventoryRoutingModule`

## AddProductComponent

#### Purpose:

- Used to add a new product into the inventory.

#### HTML Structure:

- Heading: `Add Product` in `<h2>`
- Form using `ngForm` with the following fields and **error messages**:
  - **Name**
    - `required, minLength=3`
    - Errors:
      - “Product name is required.”
      - “Name must be at least 3 characters long.”
  - **Description**
    - `required`
    - Error:
      - “Description is required.”
  - **Manufacturer**
    - `required`
    - Error:
      - “Manufacturer is required.”
  - **Price**
    - `required, min=0`
    - Errors:
      - “Price is required.”

|  |  |  |
|--|--|--|
|  |  | <ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li>■ “Price cannot be negative.”</li> </ul> </li> <li>○ <b>Quantity</b> <ul style="list-style-type: none"> <li>■ <code>required, min=1</code></li> <li>■ Errors: <ul style="list-style-type: none"> <li>■ “Quantity is required.”</li> <li>■ “Quantity must be at least 1.”</li> </ul> </li> </ul> </li> <li>● Submit Button: <code>Add Product</code> <ul style="list-style-type: none"> <li>○ Disabled when form is invalid</li> </ul> </li> </ul> <p><b>Functions &amp; Responsibilities:</b></p> <ul style="list-style-type: none"> <li>● <code>onAdd(form: NgForm)</code> <ul style="list-style-type: none"> <li>○ Validates form.</li> <li>○ Adds product via <code>dataService.addProduct(...)</code>.</li> <li>○ Generates a unique ID using <code>Date.now()</code>.</li> <li>○ Navigates back to <code>/inventory</code> with a success alert: <b>Product added successfully!</b></li> <li>○ If form is invalid, alerts (<b>Please fill in all required fields.</b>) user to complete fields.</li> </ul> </li> </ul> <h2>UpdateProductComponent</h2> <p><b>Purpose:</b></p> <ul style="list-style-type: none"> <li>● Allows editing/updating an existing product using reactive forms.</li> </ul> <p><b>HTML Structure:</b></p> <ul style="list-style-type: none"> <li>● Heading: <code>Update Product</code> in <code>&lt;h2&gt;</code></li> <li>● Form using <code>ReactiveFormsModule ([formGroup])</code> with validation messages under each field: <ul style="list-style-type: none"> <li>➔ <b>Name</b> <ul style="list-style-type: none"> <li>◆ <code>required, minLength=3</code></li> <li>◆ Errors: <ul style="list-style-type: none"> <li>● “Product name is required.”</li> </ul> </li> </ul> </li> </ul> </li> </ul> |
|--|--|--|

|  |  |   |
|--|--|---|
|  |  | <ul style="list-style-type: none"> <li>• “Name must be at least 3 characters long.”</li> </ul> <p>→ <b>Description</b></p> <ul style="list-style-type: none"> <li>◆ <b>required</b></li> <li>◆ Error: <ul style="list-style-type: none"> <li>• “Description is required.”</li> </ul> </li> </ul> <p>→ <b>Manufacturer</b></p> <ul style="list-style-type: none"> <li>◆ <b>required</b></li> <li>◆ Error: <ul style="list-style-type: none"> <li>• “Manufacturer is required.”</li> </ul> </li> </ul> <p>→ <b>Price</b></p> <ul style="list-style-type: none"> <li>◆ <b>required,min=0</b></li> <li>◆ Errors: <ul style="list-style-type: none"> <li>• “Price is required.”</li> <li>• “Price cannot be negative.”</li> </ul> </li> </ul> <p>→ <b>Quantity</b></p> <ul style="list-style-type: none"> <li>◆ <b>required,min=1</b></li> <li>◆ Errors: <ul style="list-style-type: none"> <li>• “Quantity is required.”</li> <li>• “Quantity must be at least 1.”</li> </ul> </li> </ul> <ul style="list-style-type: none"> <li>• Submit Button: <b>Update Product</b> <ul style="list-style-type: none"> <li>→ Disabled when form is invalid</li> </ul> </li> </ul> <p><b>Functions &amp; Responsibilities:</b></p> <ul style="list-style-type: none"> <li>• <b>ngOnInit()</b> <ul style="list-style-type: none"> <li>○ Gets product ID from route.</li> <li>○ Fetches product by ID and pre-fills the form.</li> <li>○ If product not found → alerts: “Product not found.”</li> <li>○ On fetch error → logs error and alerts: <ul style="list-style-type: none"> <li>→ Console: <b>'Error loading product:', error</b></li> <li>→ Alert: “An error occurred while loading the product.”</li> </ul> </li> </ul> </li> </ul> |
|--|--|---|

- `onUpdate()`
  - Validates form.
  - Updates product via `dataService.updateProduct(...)`.
  - On success → **navigates back to product list with alert:** “Product updated successfully!”
  - On update failure → logs error and alerts:
    - console: `'Error updating product:', error`
    - Alert: “An error occurred while updating the product.”
  - If form is invalid → shows alert: “Please correct the errors in the form.”

## ProductDetailComponent

### Purpose:

- Displays detailed information of a single product.

### HTML Structure:

- Heading in h2: `Product Details`
- Displays:
  - `Name, Description, Manufacturer, Price, Quantity` using `{{ product?.field }}`

### Functions & Responsibilities:

- `ngOnInit()`
  - Extracts product ID from route.
  - Fetches all products and finds the one with the matching ID.
  - Binds the found product to `this.product`.



## ProductListComponent

### Purpose:

- Lists all products with options to view, edit, or delete.
- Integrates a search filter.

### HTML Structure:

- Heading in h2: `Product List`
- Includes `<app-search-filter>` to filter products.
- `<ul>` with `*ngFor` loop to render each product with:
  - `View`, `Edit`, and `Delete` buttons.
- `Add Product` button → routes to `/add-product`.

### Functions & Responsibilities:

- `ngOnInit()`
  - Fetches all products via `dataService.getProducts(...)`.
  - Stores products in both `products` and `filteredProducts` for filtering.
- `onSearch(query: string)`
  - Filters the `products` list by product name using the entered query.
- `viewProduct(id: number)`
  - Navigates to `product-detail/:id` page.
- `editProduct(id: number)`
  - Navigates to `update-product/:id` page.
- `deleteProduct(id: number)`
  - Checks if the user is logged in using `authService.isLoggedIn()`:
    - If not logged in → alerts: "You must be logged in to delete a product!"
    - Redirects to sign-in page.

- If logged in → deletes product via `dataService.deleteProduct(...)` and updates UI.

## SharedModule

### Purpose:

- Provides reusable components (`Header`, `Footer`, `SearchFilter`) across the app.
- **Exports** them for use in other modules like `InventoryModule` and `AppModule`.

## HeaderComponent

### HTML Structure:

- Navigation links:
  - `/inventory` – Products
  - `/about` – About
  - Conditional links:
    - If not logged in → Show “Sign In” and “Register”
    - If logged in → Show “Logout” button

### Functions & Responsibilities:

#### `isLoggedIn()`

- Uses `authService.isLoggedIn()` to determine current user status.
- Controls what links are visible (e.g., show login/register or logout).

#### `logout()`

- Calls `authService.logout()` to remove user session.
- No navigation is triggered automatically — simply logs the user out.

## FooterComponent

### HTML Structure:

- Simple footer:  
"© 2024 Products Inventory"

## SearchFilterComponent

### HTML Structure:

- A single input field:
  - Placeholder: "Search..."
  - On input change → triggers `onSearch()` method.

### Functions & Responsibilities:

#### @Output() search

- Emits the search string entered by the user to the parent component.

#### onSearch(event)

- Gets the value from the input field.
- Emits trimmed value using `search.emit(query)`.

Used in **ProductListComponent** to filter the displayed list of products.

## product.model.ts

### Interface Definition:

Defines the structure of a product:

- `id, name, description, manufacturer, manufacturingDate, price, quantity`.
- Used in product CRUD forms and views to enforce structure and type safety.

|  |  |   |
|--|--|---|
|  |  | <b>user.model.ts</b><br><br><b>Interface Definition:</b><br><br>Defines the structure of a user: <ul style="list-style-type: none"> <li>• <code>id, email, password, firstName, lastName, location, mobileNumber.</code></li> <li>• Used in authentication and registration forms.</li> </ul><br><b>** Kindly refer to the screenshots for any clarifications. **</b> |
|  |  |   |

## 5 CONSTRAINTS

---

1. You should be able to press the "TAB" key and "SHIFT + TAB" to navigate from top field to bottom field and vice-versa.
2. All required fields must be completed with valid data.
3. When logging into the system, all fields must be filled.
4. When adding or updating a product, all fields are mandatory to be filled.
5. The system should validate invalid entries and display appropriate error messages.

## 6 MANDATORY ASSESSMENT GUIDELINES

---

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.
3. This editor Auto Saves the code.
4. These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
5. This is a web-based application, to run the application on a browser, use the

internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.

**Note: The application will not run in the local browser**

6. You can follow series of command to setup Angular environment once you are in your project-name folder:
  - a. `npm install` -> Will install all dependencies -> takes 10 to 15 min.
  - b. `npm run start` -> To compile and deploy the project in browser. You can press the <Ctrl> key while clicking on localhost:4200 to open the project in the browser -> takes 2 to 3 min.
  - c. `npm run json:server` -> to deploy fake rest api created with json-server -> takes 10 to 15 seconds
  - d. `npm run test` -> to run all test cases. **It is mandatory to run this command before submission of workspace** -> takes 5 to 6 min.
7. Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on **"Submit Assessment"** after you are done with code.