**1. Requirement Understanding**

**Business Requirements:**

The system allows users to:

* **View products** (public and authenticated users)
* **Add, update, and delete products** (only authenticated users)
* **User Authentication** (Sign In / Register / Logout)
* **Navigation between different pages**
* **Validation for input fields**
* **Data persistence using JSON Server**

**Technologies Used:**

* **Frontend:** Angular 19, TypeScript
* **State Management:** NgRx
* **Database:** JSON Server (db.json)
* **UI Framework:** Bootstrap
* **Routing & Guards:** Angular Router & Auth Guards

**2. Design Flow Explanation with Project Flow Diagram**

**Project Flow:**

1. **User visits Welcome Page**
2. Clicks **Sign In / Register** → Logs in → Redirects to Product List
3. If **not logged in**, Add Product redirects to Sign In
4. If **logged in**, users can:
   * View product details
   * Add, edit, or delete products
   * Logout (returns to Welcome Page)

**Project Flow Diagram:**

[Welcome Page] → [Sign In / Register] → [Product List] → [View / Add / Edit / Delete Products] → [Logout]

**3. Demo for All Code**

**App Initialization (main.ts)**

import { bootstrapApplication } from '@angular/platform-browser';

import { AppComponent } from './app/app.component';

import { appConfig } from './app/app.config';

bootstrapApplication(AppComponent, appConfig).catch(err => console.error(err));

**Routing Configuration (app.routes.ts)**

import { Routes } from '@angular/router';

import { SignInComponent } from './sign-in.component';

import { ProductListComponent } from './product-list.component';

import { ViewProductComponent } from './view-product.component';

import { AuthGuard } from './services/auth.guard';

export const appRoutes: Routes = [

{ path: '', redirectTo: 'products', pathMatch: 'full' },

{ path: 'sign-in', component: SignInComponent },

{ path: 'products', component: ProductListComponent, canActivate: [AuthGuard] },

{ path: 'view-product/:id', component: ViewProductComponent, canActivate: [AuthGuard] }

];

**4. Demo for Functionalities & Concepts Used**

**Authentication & Navigation (auth.guard.ts)**

import { Injectable } from '@angular/core';

import { CanActivate, Router } from '@angular/router';

@Injectable({ providedIn: 'root' })

export class AuthGuard implements CanActivate {

constructor(private router: Router) {}

canActivate(): boolean {

if (localStorage.getItem('isLoggedIn') !== 'true') {

this.router.navigate(['/sign-in']);

return false;

}

return true;

}

}

**Product List Component (product-list.component.ts)**

import { Component } from '@angular/core';

import { Router } from '@angular/router';

@Component({ selector: 'app-product-list', templateUrl: './product-list.component.html' })

export class ProductListComponent {

constructor(private router: Router) {}

viewProduct(productId: number) {

if (localStorage.getItem('isLoggedIn') === 'true') {

this.router.navigate(['/view-product', productId]);

} else {

alert('You must be logged in to view product details.');

this.router.navigate(['/sign-in']);

}

}

}

**Login Handling (sign-in.component.ts)**

import { Component } from '@angular/core';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { Router } from '@angular/router';

@Component({ selector: 'app-sign-in', templateUrl: './sign-in.component.html' })

export class SignInComponent {

loginForm: FormGroup;

constructor(private fb: FormBuilder, private router: Router) {

this.loginForm = this.fb.group({

email: ['', [Validators.required, Validators.email]],

password: ['', Validators.required]

});

}

onSubmit() {

if (this.loginForm.valid) {

localStorage.setItem('isLoggedIn', 'true');

this.router.navigate(['/products']);

}

}

}

**5. Maintain Code Structure**

**Recommended Folder Structure**

/src/app

├── pages/

│ ├── product-list/

│ ├── sign-in/

│ ├── register/

│ ├── view-product/

├── services/

│ ├── auth.service.ts

│ ├── product.service.ts

├── guards/

│ ├── auth.guard.ts

├── db.json

│

├── app.routes.ts

├── app.component.ts

**6. Data Validation & Exception Handling**

**Form Validation (Login Form Example)**

<form [formGroup]="loginForm" (ngSubmit)="onSubmit()">

<input type="email" formControlName="email" placeholder="Enter email">

<div \*ngIf="loginForm.controls['email'].invalid && loginForm.controls['email'].touched">

Valid email is required.

</div>

<input type="password" formControlName="password" placeholder="Enter password">

<div \*ngIf="loginForm.controls['password'].invalid && loginForm.controls['password'].touched">

Password is required.

</div>

<button type="submit" [disabled]="loginForm.invalid">Login</button>

</form>

**Exception Handling (auth.service.ts)**

login(email: string, password: string): Observable<User> {

return this.http.post<User>(`${this.apiUrl}/login`, { email, password }).pipe(

catchError((error: HttpErrorResponse) => {

console.error('Login failed', error);

return throwError(() => new Error('Invalid credentials'));

})

);

}

* ✅ **Validates user input before submission**
* ✅ **Displays user-friendly error messages**
* ✅ **Handles API errors gracefully**

**Conclusion**

The **Product Inventory Management System** is a robust Angular 19 application designed with **standalone components** and **state management** to ensure a seamless user experience. By integrating authentication, role-based access control, and data validation, the system maintains **data integrity and security.**