Міністерство освіти і науки України  
Харківський національний університет радіоелектроніки

Кафедра «Програмної інженерії»

Лабораторна робота № 3  
«РОЗРОБКА БІЗНЕС-ЛОГІКИ ТА ФУНКЦІЙ АДМІНІСТРУВАННЯ СЕРВЕРНОЇ ЧАСТИНИ ПРОГРАМНОЇ СИСТЕМИ.»

Роботу виконав:

студент 3 курсу

факультету

«Комп’ютерні науки»

групи ПЗПІ-22-6

Іванов Д.С.

Перевірив:  
ст. викладач кат. ПІ  
Сокорчук І.П.

Харків 2025

| № | Дата | Версія звіту | Опис змін та виправлень |
| --- | --- | --- | --- |
| 1 | 25.08.2025 | 0.1 | Створено початковий варіант звіту |
| 2 | 26.08.2025 | 0.2 | Розширено опис архітектури, додано UML-діаграми та фрагменти коду |
| 3 | 26.08.2025 | 1.0 | Фіналізація та підготовка до завантаження |

Метою лабораторної роботи є розробка та тестування клієнтської частини програмної системи FoodCare Monitor System.

Для цього необхідно:

1. Створити структуру клієнтської частини проєкту, визначивши ключові технології та підходи.
2. Розробити інтерфейси користувача для взаємодії з REST API, реалізованим на попередній лабораторній роботі.
3. Описати механізми управління станом додатку, маршрутизації та обробки даних у реальному часі.
4. Створити звіт з детальним описом виконаної роботи, висновками та додатками згідно зі стандартами ДСТУ.
5. Підготувати відеозапис демонстрації функціональних можливостей клієнта з хронологічним описом.
6. Завантажити звіт у PDF-форматі та відео на YouTube.
7. Розмістити програмний код та звіт у GitHub-репозиторії.

3. ОПИС ВИКОНАНОЇ РОБОТИ

3.1. Архітектура клієнтської частини Клієнтська частина програмної системи FoodCare Monitor System була розроблена на базі фреймворку Angular з використанням мови TypeScript. Застосунок реалізовано як односторінковий додаток (SPA), що забезпечує швидкий та динамічний користувацький досвід. Для взаємодії з серверною частиною (бекендом) використовується протокол HTTP та RESTful API.

Для ефективного управління станом застосунку та роботи з асинхронними даними, такими як потоки показників від сенсорів, використано бібліотеку RxJS та паттерн Reactive State Management. Це дозволило реалізувати механізм автоматичного оновлення даних на інтерфейсі без необхідності перезавантаження сторінки.

Система автентифікації та авторизації побудована на основі JWT-токенів, які зберігаються у локальному сховищі браузера. Маршрутизація між сторінками реалізована за допомогою вбудованого модуля RouterModule Angular, що забезпечує розмежування доступу до різних частин застосунку залежно від ролі користувача.

3.2. Ключові функціональні модулі Були розроблені наступні модулі:

* Модуль автентифікації (AuthModule): відповідає за реєстрацію, вхід та вихід користувачів.
* Модуль складів (WarehouseModule): містить компоненти для перегляду та керування складами.
* Модуль продуктів (ProductModule): забезпечує CRUD-операції з інформацією про продукти та партії.
* Модуль моніторингу (MonitoringModule): відображає дані, отримані від IoT-сенсорів у реальному часі, та історію показників.
* Адміністративна панель (AdminModule): надає адміністраторам інтерфейс для управління користувачами та їх ролями.

3.3. Реалізація компонентів та логіки Кожен компонент розроблено згідно з принципами модульності та інкапсуляції, що дозволяє повторно використовувати елементи інтерфейсу та полегшує подальше масштабування системи. Наприклад, компонент ProductListComponent відображає список продуктів, а AlertsComponent — список сповіщень про порушення умов зберігання.

Управління даними здійснюється через спеціалізовані сервіси (ProductService, SensorService), які інкапсулюють логіку взаємодії з API. Це дозволяє розділити представлення (компоненти) від бізнес-логіки.

4. ВИСНОВКИ

Під час виконання лабораторної роботи було успішно розроблено клієнтську частину програмної системи FoodCare Monitor System.

Основні результати:

* Розроблено надійну клієнтську архітектуру на базі Angular та TypeScript, що забезпечує високу продуктивність та зручність використання.
* Реалізовано ключовий функціонал для роботи з усіма сутностями системи, включаючи управління складами, продуктами, сенсорами та користувачами.
* Налаштовано механізм динамічного оновлення даних від сенсорів за допомогою RxJS.
* Підготовлено звітну документацію та необхідні UML-діаграми, що візуально відображають структуру та функціональність клієнтської частини.

ДОДАТОК А

Відеозапис та хронологічний опис

Хронологічний опис відеозапису:

* 0:00 - Вступ: огляд теми та цілей роботи.
* 0:35 - Демонстрація процесу реєстрації нового користувача та входу в систему.
* 1:40 - Огляд головної панелі: список складських приміщень.
* 2:20 - Сценарій "Керування складом": створення та редагування інформації про склад.
* 3:30 - Перехід до сторінки складу: перегляд підключених сенсорів та їхніх показників.
* 4:50 - Демонстрація модуля продуктів: додавання нового продукту та редагування існуючого.
* 5:55 - Сценарій "Реєстрація партії": додавання нової партії продуктів до складу.
* 7:15 - Функції адміністратора: управління користувачами та їх ролями через адмін-панель.
* 8:30 - Висновки: підсумки виконаної роботи та перспективи проєкту.

Посилання на GitHub-репозиторій: [https://github.com/NureIvanovDmytro/apz-pzpi-22-6-ivanov-dmytro/tree/main/Lab3](https://www.google.com/search?q=https://github.com/NureIvanovDmytro/apz-pzpi-22-6-ivanov-dmytro/tree/main/Lab3)

ДОДАТОК Б

Програмний код

В.1 Приклад компонента для відображення списку продуктів

TypeScript

import { Component, OnInit, inject } from '@angular/core';

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';

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

import { ToastrService } from 'ngx-toastr';

import { Product } from '../../\_models/product';

import { ProductService } from '../../\_services/product.service';

@Component({

selector: 'app-product-list',

standalone: true,

imports: [CommonModule, FormsModule],

templateUrl: './product-list.component.html',

styleUrl: './product-list.component.css'

})

export class ProductListComponent implements OnInit {

products: Product[] = [];

isLoading = true;

private productService = inject(ProductService);

private toastr = inject(ToastrService);

private router = inject(Router);

ngOnInit(): void {

this.loadProducts();

}

loadProducts(): void {

this.isLoading = true;

this.productService.getProducts().subscribe({

next: (products) => {

this.products = products;

this.isLoading = false;

},

error: (err) => {

this.toastr.error('Failed to load products.');

this.isLoading = false;

}

});

}

deleteProduct(id: number): void {

if (confirm('Are you sure you want to delete this product?')) {

this.productService.deleteProduct(id).subscribe({

next: () => {

this.toastr.success('Product deleted successfully.');

this.loadProducts();

},

error: () => {

this.toastr.error('Failed to delete product.');

}

});

}

}

editProduct(id: number): void {

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

}

}

В.2 Приклад компонента для відображення сповіщень

TypeScript

import { Component, OnInit, OnDestroy, inject } from '@angular/core';

import { CommonModule } from '@angular/common';

import { Subscription } from 'rxjs';

import { Alert } from '../../\_models/alert';

import { AlertService } from '../../\_services/alert.service';

@Component({

selector: 'app-alerts',

standalone: true,

imports: [CommonModule],

templateUrl: './alerts.component.html',

styleUrl: './alerts.component.css'

})

export class AlertsComponent implements OnInit, OnDestroy {

alerts: Alert[] = [];

private alertSubscription: Subscription | undefined;

private alertService = inject(AlertService);

ngOnInit(): void {

this.alertSubscription = this.alertService.getAlerts().subscribe({

next: (alerts) => {

this.alerts = alerts;

},

error: (err) => {

console.error('Error fetching alerts', err);

}

});

}

ngOnDestroy(): void {

this.alertSubscription?.unsubscribe();

}

getAlertClass(level: string): string {

switch (level.toLowerCase()) {

case 'critical':

return 'alert-danger';

case 'warning':

return 'alert-warning';

case 'info':

return 'alert-info';

default:

return 'alert-secondary';

}

}

dismissAlert(alertId: number): void {

// Логіка для "приховування" або "видалення" сповіщення

this.alerts = this.alerts.filter(a => a.id !== alertId);

}

}