Міністерство освіти і науки України  
Харківський національний університет радіоелектроніки

Кафедра «Програмної інженерії»

Лабораторна робота № 2  
«РОЗРОБКА БАЗИ ДАНИХ ДЛЯ СЕРВЕРНОЇ ЧАСТИНИ ПРОГРАМНОЇ СИСТЕМИ ТА ПРИКЛАДНОГО ПРОГРАМНОГО ІНТЕРФЕЙСУ (API)..»

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** |
| --- | --- | --- | --- |
| 1 | 25.08.2025 | 0.1 | Створено початковий варіант звіту |
| 2 | 25.08.2025 | 1.0 | Додано опис серверної архітектури, ER-модель та UML-діаграми |
| 3 | 25.08.2025 | 1.1 | Додано приклади API та кодових фрагментів |

ЗАВДАННЯ

1. Розробити серверну частину програмної системи FoodCare Monitor System.
2. Виконати аналіз вимог та створити архітектуру API.
3. Описати базу даних і зв’язки між сутностями.
4. Реалізувати основні контролери для роботи з користувачами, продуктами, партіями та сенсорами.
5. Провести функціональне тестування REST API.
6. Створити відеозапис із демонстрацією тестування API та завантажити його на YouTube.
7. Оформити звіт із додатками (UML-діаграми, ER-модель, кодові фрагменти).

3. ОПИС ВИКОНАНОЇ РОБОТИ

3.1 Архітектура серверної частини

Серверна частина системи реалізована за допомогою ASP.NET Core мовою C#. Архітектурний підхід — клієнт-серверна модель, де бекенд виступає центральним вузлом, що взаємодіє з базою даних та IoT-шлюзом.

Використано патерни:

1. Repository — для відокремлення логіки доступу до БД.
2. Dependency Injection — для гнучкого керування залежностями.
3. MVC — для структурованої організації контролерів.

Зображення, що містить текст, схема, знімок екрана, ряд

Вміст на основі ШІ може бути неправильним.

3.2 База даних

У системі використовується PostgreSQL. Схема включає таблиці:

1. products — інформація про продукти (назва, опис, допустимі умови зберігання).
2. batches — партії товарів із датою виготовлення та терміном придатності.
3. sensors — IoT-пристрої для збору даних.
4. readings — історія показників (температура, вологість, CO₂).
5. users — облікові записи з різними ролями.
6. alerts — події про порушення умов.
7. warehouses — інформація про склади.

Зображення, що містить текст, схема, почерк, Паралель

Вміст на основі ШІ може бути неправильним.3.3 API інтерфейси

API побудоване за принципом REST і має наступні групи:

Аутентифікація та авторизація:

POST /auth/login — вхід користувача.

POST /auth/register — реєстрація нового користувача.

Продукти та партії:

GET /product/{id} — отримати інформацію про продукт.

POST /product — створення продукту.

PUT /product/edit — редагування.

DELETE /product/{id} — видалення.

GET /batch/{id} — отримати інформацію про партію.

Сенсори та дані:

POST /sensor — створити сенсор.

PATCH /sensor/updatemeasureddata — передати показники від IoT.

GET /sensor/{warehouseId}/sensors — отримати всі сенсори складу.

Сповіщення та порушення умов:

POST /alert — створення повідомлення.

GET /alert/{warehouseId} — отримати історію сповіщень.

Зображення, що містить схема, текст, ескіз, дизайн

Вміст на основі ШІ може бути неправильним.

3.4 UML-діаграми

У процесі розробки були створені наступні діаграми:

Use-Case для ролей (адміністратор, менеджер, працівник складу, IoT-сенсор).

Deployment — структура серверної частини та інтеграція з клієнтами.

ER-модель — сутності БД та їхні зв’язки.

Activity — бізнес-процеси (наприклад, перевірка умов зберігання).

4. ВИСНОВКИ

У результаті виконання ЛР2 було реалізовано серверну частину програмної системи FoodCare Monitor System.

Отримані результати:

Створено REST API з підтримкою CRUD-операцій для основних сутностей.

Розроблено базу даних PostgreSQL із нормалізованими таблицями.

Використано патерни Repository та Dependency Injection.

Проведено тестування API та перевірено роботу з користувачами, продуктами, партіями й сенсорами.

Підготовлено UML-діаграми, що відображають архітектуру та бізнес-логіку системи.

ДОДАТОК А

Хронологія:

0:00 – Вступ

0:25 – Огляд бази даних

1:10 – Тестування створення користувача

2:05 – Тестування входу в систему

3:30 – Тестування CRUD для продуктів

5:00 – Тестування партій

6:10 – Тестування сенсорів

7:20 – Демонстрація сповіщень

8:00 – Висновки

ДОДАТОК Б

Графічні матеріали

Рисунок Б.1 – UML діаграма розгортання

Рисунок Б.2 – UML діаграма прецедентів для менеджера складу

Рисунок Б.3 – UML діаграма прецедентів для адміністратора

Рисунок Б.4 – ER-діаграма даних

Рисунок Б.5 – Діаграма діяльності «Додавання продукту»

ДОДАТОК В

Фрагменти коду

В.1 Приклад контролера користувачів

[ApiController]

[Route("api/[controller]")]

public class UserController : ControllerBase

{

private readonly IUserRepository \_repository;

public UserController(IUserRepository repository)

{

\_repository = repository;

}

[HttpGet("{id}")]

public async Task<IActionResult> GetUser(int id)

{

var user = await \_repository.GetUserAsync(id);

if (user == null) return NotFound();

return Ok(user);

}

[HttpPost]

public async Task<IActionResult> CreateUser(UserDto model)

{

var id = await \_repository.CreateUserAsync(model);

return Ok(new { userId = id });

}

}

В.2 Приклад сервісу для обчислення середнього показника температури

public class TemperatureService

{

public double CalculateAverage(List<double> readings)

{

if (readings == null || readings.Count == 0) return 0;

return readings.Average();

}

}