МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет мехатроніки та комп'ютерних технологій
(повне найменування інституту, назва факультету)

Кафедра інформаційних та комп'ютерних технологій
(повне найменування інституту, назва факультету)

Дипломна бакалаврська робота

на тему: Автоматизована система моніторингу параметрів обладнання розподілених систем

Виконав: студент групи БА-19
Спеціальності:
151 – Автоматизація та комп'ютерно-інтегровані технології
за освітньо-професійною програмою:
Автоматизація та комп'ютерно-інтегровані технології

Валентин СТРУЧОК

Керівник: к.т.н., доц. Юрій ЛЕБЕДЕНКО

Рецензент: к.т.н., доц. Олександр МАНОЙЛЕНКО

Київ 2023

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Інститут, факультет: Мехатроніки та комп'ютерних технологій
Кафедра: Інформаційних та комп'ютерних технологій
Спеціальність: 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітня програма: Автоматизація та комп'ютерно-інтегровані технології

ЗАТВЕРДЖУЮ
Завідувач кафедри ІКТ
доц., к.т.н. Владислава СКІДАН

_____
«___»_____2023 р.

З А В Д А Н Н Я

НА ДИПЛОМНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

СТРУЧКУ Валентину Васильовичу

1. Тема роботи: Автоматизована система моніторингу параметрів обладнання розподілених систем.
   Науковий керівник роботи Лебеденко Ю. О. к.т.н., доц.,
   Затверджені наказом вищого навчального закладу від «08» листопада 2023 року №224-уч
2. Строк подання студентом роботи 19.06.2023 року
3. Вихідні данні до роботи: Система повинна здійснювати контроль за поточним станом об'єктів нагляду з виводом їх даних до хмарного сховища, не зважаючи на походження об'єкту моніторингу. Система повинна мати функціонал для додавання та видалення об'єктів з листа моніторингу.
4. Зміст дипломної роботи (перелік питань, які потрібно розробити) Вступ; Розділ 1 Аналіз функціоналу мови програмування C#; Розділ 2 Аналіз функціоналу Sql та його використання для реалізації системи автоматизованого моніторингу; Розділ 3 Розробка програмного забезпечення; Загальні висновки;
5. Дата видачі завдання**:** 08.03.2023 року

Календарний план

№ з/п	Назва етапів дипломної бакалаврської роботи	Термін виконання	Примітка про виконання
1	Вступ	24.04.2023	
2	Розділ 1 Аналіз функціоналу мови програмування C#	01.05.2023	
3	Розділ 2 Аналіз функціоналу Sql та його використання для реалізації системи автоматизованого моніторингу	10.05.2023	
4	Розділ 3 Розробка програмного забезпечення	20.05.2023	
5	Висновки	30.05.2023	
6	Оформлення дипломної бакалаврської роботи (чистовий варіант)	01.06.2023	
7	Здача дипломної бакалаврської роботи на кафедру для рецензування (за 14 днів до захисту)	05.06.2023	
8	Перевірка дипломної бакалаврської роботи на наявність текстових співпадінь та помилок (за 10 днів до захисту)	09.06.2023	
9	Подання дипломної бакалаврської роботи на затвердження завідувачу кафедри (за 7 днів до захисту)	12.06.2023	

Студент _____ Валентин СТРУЧОК
(підпис)

Науковий керівник _____ Юрій ЛЕБЕДЕНКО
(підпис)

Рецензент _____ Олександр МАНОЙЛЕНКО
(підпис)

АНОТАЦІЯ

СТРУЧОК В. В. Автоматизована система моніторингу параметрів обладнання розподілених систем. – Рукопис.

Дипломна бакалаврська робота за спеціальністю 151 – Автоматизація та комп'ютерно-інтегровані технології. – Київський національний університет технологій та дизайну, Київ, 2023 рік.

Дипломну бакалаврську роботу присвячено розробленню автоматизованої система моніторингу параметрів обладнання розподілених систем.

Аналіз технологічного процесу роботи систем моніторингу показав, що для покращення ефективності їх роботи слід зменшити кількість звернень до баз даних за рахунок створення в системі власного архіву, на якому можна побудувати більшість функціоналу програми. Для створення системи обрано мову програмування C#, в якості платформи обрано Microsoft Visual Studio.

На основі проведеного аналізу розроблена демонстративна система-додаток для моніторингу за технологічними об’єктами та контролю поточних показників обладнання. Розроблений алгоритм взаємодії бази даних, внутрішнього архіву і додатка для ефективного моніторингу за технологічними об’єктами.

Ключові слова: система моніторингу, C#, база даних, внутрішній архів, демонстративна програма, контроль поточних показників обладнання.

ANNOTATION

STRUCHOK V.V. Automated system for monitoring the parameters of distributed systems equipment. - Manuscript.

Bachelor's thesis in the specialty 151 - Automation and Computer-Integrated Technologies. - Kyiv National University of Technologies and Design, Kyiv, 2023.

The bachelor's thesis is dedicated to the development of an automated system for monitoring the parameters of distributed systems equipment.

The analysis of the technological process of the monitoring systems operation showed that to improve their efficiency, it is necessary to reduce the number of database queries by creating a dedicated archive within the system, on which most of the program's functionality can be built. The C# programming language was chosen for system development, and Microsoft Visual Studio was selected as the platform.

Based on the conducted analysis, a demonstrative system-application has been developed for monitoring technological objects and controlling current equipment indicators. An algorithm for interaction between the database, internal archive, and application has been developed to enable efficient monitoring of technological objects.

Keywords: monitoring system, C#, database, internal archive, demonstrative program, equipment's current indicators control

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API - Інтерфейс програмування додатків (Application Programming Interface)

BCL - Base Class Library

CLR - Common Language Runtime

DCL - Мова управління даними (Data Control Language)

DDL - Мова визначення даних (Data Definition Language)

DML - Мова маніпулювання даними (Data Manipulation Language)

DQL - Мова запитів (Data Query Language)

FCL - Framework Class Library

IDE - інтегроване середовище розробки

IL - Intermediate Language (проміжна мова)

IntelliSense - функціонал автодоповнення коду

JIT - Just-In-Time (динамічний) компілятор

MAMP - Macintosh, Apache, MySQL, PHP

NGWS - Next Generation Windows Services

PE - Portable Executable

phpMyAdmin - PHP MySQL Administration

RAM - Оперативна пам'ять (Random Access Memory)

SQL - Мова структурованих запитів (Structured Query Language)

U+xxxx - Unicode code point

UI - Графічний інтерфейс користувача (User Interface)

Unicode - Universal Coded Character Set

VB.NET - Visual Basic .NET, мова програмування VB.NET

VOS - Virtual Object System

БД - База даних

ООП - Об'єктно-орієнтоване програмування

СУБД - Система управління базами даних

ЗМІСТ

ВСТУП

У сучасному світі питання автономності та незалежності у сфері електропостачання, теплопостачання, водопостачання або інтернету стає дедалі популярнішим та перспективнішим. Це пов'язано зі зростаючою свідомістю людей щодо необхідності екологічної сталості та забезпечення своєї безпеки та комфорту у повсякденному житті. А згадуючи про ситуацію в нашій країні, ця тема набуває актуальності та потребує комплексного підходу та вдосконалення відповідних систем та технологій з метою забезпечення максимальної надійності та ефективності їх роботи.

Саме тому для цього використовують найсучасніші технологічні прилади, які мають достатній рівень самостійності та потребують лише мінімальних втручань з боку людини. Зазвичай, для більш зручного використання цих технологічних приладів та комфортного стеження за ними, їх конструкція може припускати можливість передачі даних свого поточного стану через «хмару» у додаток на будь-якому електронному пристрої, наприклад телефоні або комп’ютері. Але у випадках використання технологічних приладів від різних фірм часто виникає проблема несумісності, через що за кожним приладом доводиться стежити у різних додатках. Це незручно і іноді може призвести до неприємних наслідків або навіть катастроф.

Через це виникає потреба у створенні єдиного інтерфейсу, який міг би сприймати інформацію з «хмари» незалежно від фірми-виробника та відображати дані усіх технологічних приладів у будівлі, або у кількох будівлях.

Об’єктом дослідження є системи збору інформації з технологічних об’єктів спостереження від різних фірм та виробників.

Предметом дослідження є моделі та методи збору інформації з технологічних об’єктів спостереження.

Метою дослідження є підвищення ефективності процесів збору інформації з віддалених технологічних об’єктів спостереження. За рахунок створення додатку, здатного запам’ятовувати велику кількість об’єктів спостереження та їх характеристики, використовуючи БД, при цьому знайти максимально ефективний алгоритм експлуатації БД.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Провести аналіз функціоналу мови програмування C# для його подальшого використання для реалізації поставленої мети.
2. Після проведення аналізу функціоналу C# та підібрати шлях для розробки програмного забезпечення автоматизованого моніторингу, проаналізувати функціонал Sql та підібрати спосіб для ефективної реалізації БД в програмному забезпечені.
3. Після проведення всіх необхідних досліджень, почати розробку програмного забезпечення спостереження за поточним станом об’єкту спостереження.

Апробація результатів бакалаврської роботи: тези доповіді представлені на X Всеукраїнській науково-практичній конференції, присвяченої Дню космонавтики від 12 квітня 2023 року, м. Херсон – Хмельницький (Додаток Д) [1].

Структура і обсяг роботи: робота складається зі вступу, 3 розділів, висновків, списку використаних джерел (28 найменувань), 5 додатків. Загальний обсяг бакалаврської роботи 93 сторінки комп'ютерного тексту.

РОЗДІЛ 1. АНАЛІЗ ФУНКЦІОНАЛУ МОВИ ПРОГРАМУВАННЯ C#

1.1. Середовище розробки Visual Studio 2022

Microsoft Visual Studio 2019 - одна з версій продукту компанії Майкрософт, яка включає інтегроване середовище розробки програмного забезпечення та інші інструментальні засоби.

Рис. 1.1 Логотип Visual Studio 2022

Цей продукт дозволяє створювати як консольні програми, так і програми з графічним інтерфейсом, включаючи підтримку технології Windows Forms, а також веб-сайтів та веб-застосунків для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight.[8]

Visual Studio має в собі редактор коду, який підтримує IntelliSense та функціонал рефакторингу. Вбудований відлагоджувач працює як на рівні вихідного коду, так і на рівні машинного коду. Інші вбудовані інструменти включають веб-дизайнер, профіллювальник коду, конструктор класів та

конструктор схем баз даних, а також конструктор для створення додатків з графічним інтерфейсом. Він також має плагіни, які розширюють його функціональність практично на всіх рівнях, включаючи підтримку систем управління версіями та додавання нових наборів інструментів, таких як редактори і візуальні дизайнери для мов, специфічних для конкретного домену, або інструменти для інших аспектів життєвого циклу розробки програмного забезпечення. Нижче зображений інтерфейс редактора.[9]

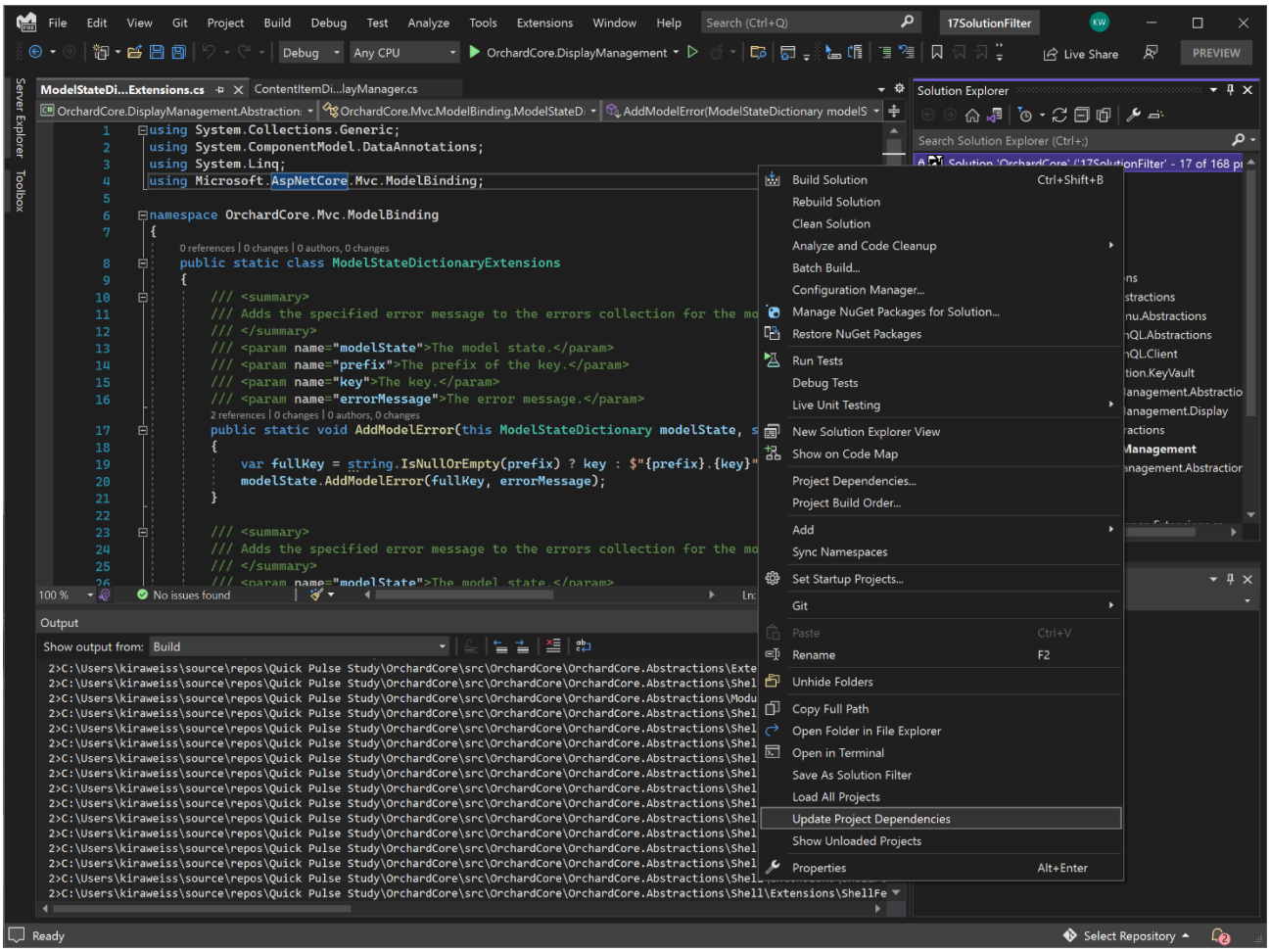


Рис. 1.2 Робочий інтерфейс Visual Studio 2022

Однією з особливостей та великою перевагою Microsoft Visual Studio є те, що цей продукт постійно оновлюється, і користувач може використовувати лише ті компоненти розробки, які йому потрібні, не завантажуючи зайві на свій персональний комп'ютер або ноутбук. Крім того, Microsoft Visual Studio надає

підтримку при написанні коду та рекомендації з використання різних методик програмування.

1.2. Мова програмування C#

C# є мовою програмування, що базується на об'єктно-орієнтованому підході і була розроблена компанією Microsoft Research з метою надання безпечної типізації для платформи .NET. Синтаксис мови строго дотримується статичної типізації змінних і підтримує всі основні принципи об'єктно-орієнтованого програмування, такі як перевантаження операторів, поліморфізм, атрибути, події, вказівники на функції-члени класів, властивості, винятки та інші.[26]

Мова C# також підтримує препроцесорні директиви, які базуються на препроцесорі C і дозволяють розробнику визначати символи для прискорення розробки. Різноманітні умовні директиви, такі як #if, #endif або #else, можуть бути використані під час написання коду. Директиви типу #region надають команди для згортання та розгортання фрагментів коду в середовищі розробки.[5]

Загалом, мова програмування C# була розроблена як мова складового рівня, яка використовує всі можливості Common Language Runtime (CLR). Особлива увага була приділена системі типів, яка використовується в C#, і її відповідності до Framework Class Library (FCL).[6]

1.3. Технологія .NET FRAMEWORK

Технологія .NET Framework надає середовище для розробки та виконання програм незалежно від платформи (операційної системи). Вона дозволяє

розробляти проекти, в яких частини коду можуть бути написані різними мовами програмування. Основна мова програмування в .NET Framework - це C#. В рамках цієї технології використовуються єдиний спосіб опису програмного коду (метадані), спільне середовище виконання (Common Language Runtime - CLR) та базова бібліотека класів (Base Class Library - BCL).[7]

Для досягнення міжмовної інтеграції програми, написані на високорівневих мовах (наприклад, C#, Basic, C++), спочатку компілюються у проміжну мову (Intermediate Language - IL), а не в машинний код. Незалежність від платформи забезпечується тим, що під час виконання програми команди проміжної мови перетворюються у машинні команди за допомогою середовища CLR та JIT-компіляторів. Таким чином, технологія .NET Framework реалізована в середовищі CLR, яке використовується для перетворення команд проміжної мови (IL) в команди процесора.[11]

CLR надає такі можливості для розробки програм:

- Міжмовна інтеграція.
- Автоматичне керування пам'яттю (збирання сміття).
- Контроль типів.
- Контроль версій програм.

Ці можливості досягаються завдяки використанню метаданих. Метадані містять описи програм та посилання на типи, що використовуються у системі. Вони об'єднують програмний код та іншу необхідну інформацію в одному файлі формату Portable Executable (PE), що спрощує установку програми та її залежностей.[27]

Основна мова програмування в .NET Framework - це C#, але програми можуть бути розроблені і на інших мовах програмування з використанням правил, визначених у середовищі CLR. Система віртуальних об'єктів (NGWS Virtual Object System або VOS) визначає правила опису та використання типів у CLR, спрощуючи міжмовну інтеграцію та забезпечуючи безпечність типів без

значного впливу на роботу програми. VOS складається з чотирьох основних компонентів: системи типів, метаданих, специфікації єдиної мови (Common Language Specification) та віртуального середовища виконання (Virtual Execution System), яке відповідає за завантаження та виконання програм, використовуючи CLR.[28]

1.4. Типи даних

Тип даних представляє сукупність характеристик певного набору даних, які включають діапазон значень, які ці дані можуть приймати, а також набір операцій, які можна виконувати над цими даними. Тип даних визначає множину значень, до яких відносяться дані відповідного типу.

З іншого боку, тип даних можна розглядати як опис того, яка структура та розмір мають комірки оперативної пам'яті під час збереження даних певного типу. Іншими словами, тип даних можна розглядати як опис розподілу групи двійкових розрядів.

Елемент даних певного типу представляє собою комірку або комірки оперативної пам'яті з фіксованою адресою, розряди яких розподіляються відповідно до опису даного типу даних.

Слід зауважити, що тип даних є абстрактним описом і, отже, не може бути безпосередньо використаний. Його використання здійснюється через елементи даних відповідного типу. Наприклад, елемент даних типу "byte" є коміркою оперативної пам'яті з певною адресою, розряди якої декодуються або розуміються згідно з описом типу "byte". Оскільки безпосередньо взаємодіяти з елементами даних через вказівку адреси є незручним, кожному елементу даних надається ідентифікатор (ім'я змінної або константи), за допомогою якого можна отримати доступ до елемента даних.

Кожен тип даних має своє унікальне ім'я (ідентифікатор), яке є синонімом певного опису елементу даних відповідного типу. Наприклад, ідентифікатор "byte" є синонімом опису: 8 послідовних розрядів, які містять ціле значення без знаку в діапазоні від 0 до 255 (представлені у двійковій формі і займають 1 байт).[10]

У програмах дані можуть бути представлені як константи, змінні (значення яких зберігаються в оперативній пам'яті) та файли (на зовнішніх носіях інформації).

Ідентифікатор, який може мати різні значення під час виконання програми, називається змінною. Якщо вміст елемента даних залишається незмінним (сталим), відповідний ідентифікатор називається константою.

Отже, у програмах змінна або константа характеризується такими властивостями: ім'ям, типом і значенням.

У мові C# підтримуються стандартні типи даних, які можуть бути простими типами значень або типами посилань. Прості типи значень включають такі типи, як "char", "int", "float". Типи посилань включають класи, інтерфейси, масиви та делегати.

До стандартних «простих» типів у мові C# відносяться наступні (див. табл.1.1)

Таблиця 1.1

Стандартні «прості» типи у мові C#

Тип	**Опис**	**Область значень**
object	Базовий клас для всіх інших типів	
string	Рядковий тип, послідовність символів Unicode	
sbyte	8-розрядне ціле число з знаком	-128 до 127
short	16-розрядное ціле число з знаком	-32768 до 32767

Продовження табл. 1.1

long	64-розрядне ціле число з знаком	-9223372036854775808 до 9223372036854775807
byte	8-розрядне ціле число без знака	0 до 255
ushort	16-розрядне ціле число без знака	0 до 65535
uint	32-розрядне ціле число без знака	0 до 4294967295
ulong	64-розрядне ціле число без знака	0 до 18446744073709551615
float	Число з плаваючою крапкою 4 байти, точність — 7 розрядів	±1,5*10-45 до ±3,4*1033
double	Число з плаваючою крапкою 8 байт, точність —16 розрядів	±5*10-324 до ±1,7*10306
bool	Логічний тип	true або false
char	Тип символу Unicode	U+0000 до U+ffff
decimal	Тип десяткового числа 12 байт, точність — 28 розрядів	(Від -7,9 x 1028 до 7,9 x 1028) / (100–28)

1.5. Класи

Клас у програмуванні є шаблоном для створення об'єктів. Він подібний до типів, які використовуються в традиційних мовах програмування, з тією відмінністю, що користувач може створювати власні класи в об'єктно-орієнтованому програмуванні.

Якщо клас є аналогією типу в традиційних мовах програмування, то об'єкт відповідає змінній. Інколи об'єкти називаються екземплярами класу. Об'єкт класу реалізує певний варіант опису класу. При створенні об'єкту класу виділяється пам'ять.[3]

При описі класу формується лише опис його елементів, тобто полів і методів, а пам'ять для цих елементів не виділяється. Код класу описує роботу з даними, які зберігаються в класі, а поля визначають дані, які містить клас.

Після оголошення об'єкту класу виділяється пам'ять для збереження його даних. Клас є фізичною абстракцією, яка отримує конкретну форму лише після створення об'єкту.

У класі оголошуються дані (поля) і програмний код (методи). Більшість класів містять як дані, так і програмний код. Дані зберігаються у полях класу, які можуть мати будь-який допустимий тип в програмі.

Рис. 1.3 Опис класу

До функцій-членів класу можуть відноситись:

- методи;
- конструктори;
- деструктори;
- індексатори;
- події;

- оператори;
- властивості.

Клас створюється за допомогою ключового слова class. Загальна форма опису класу має такий вигляд:

```
class ім'я_класу
{
    // Оголошення змінних екземпляру класу.
    доступ тип змінна1;
    доступ тип змінна2;
    // ...
    доступ тип змінnaN;

    // Оголошення методів.
    доступ тип_повернення метод1(параметри)
    {
        // тіло методу
    }
    доступ тип_повернення метод2(параметри)
    {
        // тіло методу
    }

    // ...

    доступ тип_повернення методN(параметри)
    {
        // тіло методу
    }
}
```

У вищезазначеному описі:

- ім’я класу - ім'я, яке використовується для класу. Це ім'я визначає новий тип даних і повинно відображати сутність класу.
- доступ - тип доступу до членів класу, такий як public, private, protected, internal або комбінація модифікаторів доступу.
- тип - тип змінної, яка є членом класу.
- змінна1, змінна2, ... - змінні, які є полями класу (даними в класі).
- метод1, метод2, ... - методи, які є членами класу і виконують певну функцію.
- тип_повернення - тип даних, який метод класу повертає.
- параметри - параметри методів, які визначаються в класі.

Доступ до членів класу може мати п'ять типів:

- private - закритий доступ, доступний тільки в межах самого класу.
- public - відкритий доступ, доступний ззовні класу.
- protected - захищений доступ, доступний у нащадках класу або в межах ієрархії класів.
- internal - доступний в межах збірки, але недоступний за її межами.
- protected internal - комбінація захищеного і внутрішнього доступу.

Таким чином, опис класу визначає його структуру і функціональні можливості, а створення об'єкту класу виділяє пам'ять для збереження даних.

1.6. Умовні оператори та цикли

Умовні конструкції - одна з базових складових багатьох мов програмування, які керують роботою програми по одному зі шляхів залежно від певних умов. Однією з таких конструкцій у мові програмування C# є конструкція if..else.[4]

Конструкція if/else перевіряє істинність певної умови і, залежно від результатів перевірки, виконує певний код.

Найпростіша форма цієї конструкції має вигляд блоку if:

```
if (умова)
{
    // дії, якщо умова істинна
}
```

Блок else виконується, якщо умова після if є хибною, тобто дорівнює false. Якщо блок else містить лише одну інструкцію, то можна скоротити його, видаливши фігурні дужки:

```
if (умова)
{
    // дії, якщо умова істинна
}
else
{
    // дії, якщо умова хибна
}
```

При порівнянні чисел, ми можемо мати три стани: перше число більше другого, перше число менше другого і числа рівні.

Тернарний оператор також дозволяє перевірити певну умову і, залежно від її істинності, виконати певні дії. Тут ми маємо відразу три операнди. Залежно від умови тернарний оператор повертає другий або третій операнд: якщо умова дорівнює true, то повертається другий операнд; якщо умова дорівнює false, то третій. Результат тернарного оператора (тобто другий або третій операнд залежно від умови) присвоюється змінній.

Цикли є управляючими конструкціями, які дозволяють в залежності від певних умов виконувати певні дії багато разів. У C# існують такі види циклів:

- for
- foreach
- while
- do...while

Цикл for

Цикл for використовується для повторення певного блоку коду задану кількість разів або залежно від певної умови. Він має наступний синтаксис:

```
for (ініціалізація; умова; ітерація)
{
    // дії циклу
}
```

- Ініціалізація: це частина, де ви можете визначити змінні, які будуть використовуватися в циклі. Зазвичай це початкове значення лічильника циклу.
- Умова: це умова, яка перевіряється на початку кожної ітерації циклу. Якщо умова є істинною, код усередині циклу виконується. Якщо умова є хибною, виконання циклу завершується.
- Ітерація: це дія, яка відбувається в кінці кожної ітерації циклу. Зазвичай це зміна значення лічильника циклу.

Цикл do/while:

Цикл do/while виконується принаймні один раз, навіть якщо умова не є істинною. Він має наступний синтаксис:

```
do
{
    // дії циклу
}
while (умова);
```

- Дії циклу: це блок коду, який виконується спочатку до перевірки умови.
- Умова: це умова, яка перевіряється після виконання блока коду. Якщо умова є істинною, цикл повторюється. Якщо умова є хибною, виконання циклу завершується.

Цикл foreach:

Цикл foreach використовується для перебору елементів колекції, такої як масив або список. Він має наступний синтаксис:

```
foreach (тип_даних змінна in колекція)
{
    // дії циклу
}
```

- Тип_даних: це тип даних елементів в колекції.
- Змінна: це змінна, в яку кожен елемент колекції буде поміщений під час кожної ітерації циклу.
- Колекція: це колекція, яка містить елементи, що будуть перебрані.

Під час виконання цикл послідовно перебирає елементи колекції і поміщає їх у змінну, так що в межах блоку циклу можна виконувати з ними певні дії.

1.7. Windows Forms

Для реалізації програмного застосунку був вибраний інтерфейс програмування додатків Windows Forms. Ця технологія є відкритою бібліотекою, яка входить до складу сімейства .NET Framework.

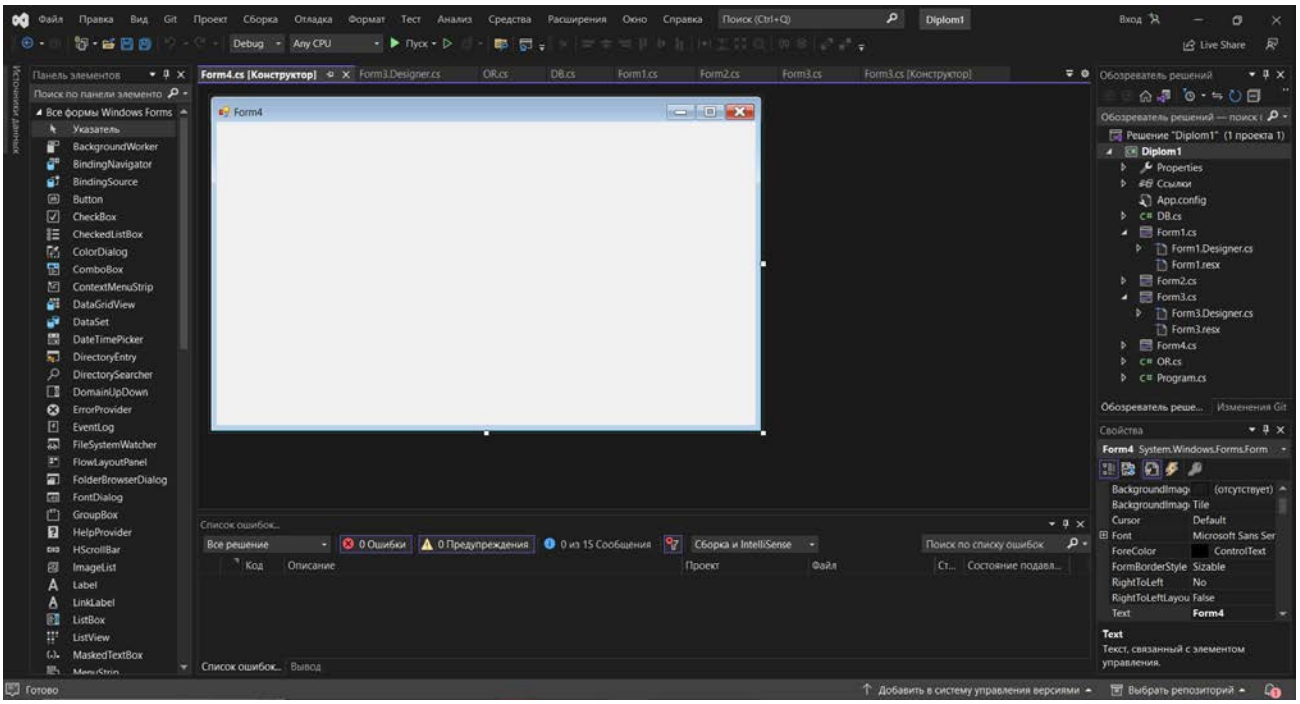


Рис.1.4 Інтерфейс програми Visual Studio для проекту WinForms

Windows Forms є мультиплатформеним API, що дозволяє розробляти додатки як для настільних комп'ютерів і ноутбуків, так і для планшетних ПК. В основному WinForms дозволяє розробникам розміщувати елементи керування на формі Windows за допомогою технології "перетягни й відпусти" (drag & drop) і змінювати їх властивості за допомогою коду, написаного на мовах C#, VB.NET або будь-якій іншій мові з сімейства .NET. Кожен елемент керування WinForms є екземпляром класу, оскільки WinForms існує як "обгортка", що використовує набір класів C#.

Visual Studio від Microsoft полегшує використання технології WinForms, оскільки розробники можуть швидко перетягувати готові елементи керування з

панелі інструментів. У десктопних програмах розробник може отримати доступ тільки до файлу коду, де він може маніпулювати подіями керування.

Переваги використання Windows Forms:

- Велика база документації для розробки додатків;
- Велика кількість прикладів;
- Зручний конструктор, оснований на Visual Studio;
- Можливість додавання сторонніх елементів з магазину додатків.

Недоліки використання Windows Forms:

- Деякі сторонні елементи можуть бути комерційними;
- Без додаткових елементів складно налаштовувати дизайн інтерфейсу додатку.

РОЗДІЛ 2. АНАЛІЗ ФУНКЦІОНАЛУ SQL ТА ЙОГО ВИКОРИСТАННЯ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ АВТОМАТИЗОВАНОГО МОНІТОРИНГУ

2.1. Бази даних

База даних (БД) є набором логічно зв'язаних даних та їх описом, який призначений для задоволення інформаційних потреб організації.[12]

Система управління базами даних (СУБД) є програмним забезпеченням, що дозволяє користувачам визначати, створювати і підтримувати базу даних, а також здійснювати контрольований доступ до неї.[15]

У БД кожна таблиця представляється як сукупність рядків і стовпців, де рядки відповідають конкретним об'єктам, подіям або явищам, а стовпці представляють атрибути цих об'єктів, подій або явищ.[17]

У кожній таблиці БД необхідний первинний ключ, який однозначно ідентифікує кожен запис. Значення первинного ключа повинні бути унікальними, тобто не може бути двох записів з однаковими значеннями первинного ключа. Первинний ключ повинен бути мінімальним та не повинен містити поля, видалення яких призведе до втрати його унікальності.[14]

Зв'язки між об'єктами реального світу можуть бути відображені в структурі даних або можуть мати неформальний характер.

У базі даних можуть існувати відношення підлеглості між двома або більш таблицями. Вони визначають, що для кожного запису в головній таблиці може бути один або декілька записів в підлеглій таблиці.[13]

Існують три типи зв'язків між таблицями в базі даних:

1. Відношення "один-до-багатьох":

У відношенні "один-до-багатьох" одному запису в батьківській таблиці відповідає декілька записів у дочірній таблиці. Цей тип зв'язку іноді називають "багато-до-одного". Відношення "один-до-багатьох" є найпоширенішим у реляційних базах даних і дозволяє моделювати ієрархічні структури даних.[24]

2. Відношення "один-до-одного":

У відношенні "один-до-одного" одному запису в батьківській таблиці відповідає лише один запис у дочірній таблиці. Цей тип зв'язку зустрічається рідше, ніж "один-до-багатьох". Він використовується, коли необхідно уникнути зайвої інформації у таблиці, але для отримання пов'язаної інформації з декількох таблиць потрібно провести декілька операцій читання замість однієї, якщо дані зберігаються в одній таблиці.[21]

3. Відношення "багато-до-багатьох":

Відношення "багато-до-багатьох" застосовується випадках, коли:

- Один запис в батьківській таблиці відповідає багатьом записам у дочірній таблиці.
- Один запис в дочірній таблиці відповідає багатьом записам у батьківській таблиці.

У реляційних базах даних тип "багато-до-багатьох" вимагає додаткових таблиць для заміни, використовуючи зв'язок "один-до-багатьох" (один або більше).[19]

2.2. MySQL

MySQL - це система управління базами даних, яка використовує реляційний підхід для зберігання та обробки даних. Вона базується на сервер-клієнтській архітектурі і використовує мову структурованих запитів (SQL) для взаємодії з базою даних. Реляційна база даних організовує дані в таблиці, або

кілька таблиць, де типи даних можуть бути пов'язані, що допомагає структурувати дані.[1]

Незважаючи на відсутність деяких функціональних можливостей, які можуть бути присутні в інших базах даних, MySQL має широкий спектр доступних інструментів для створення додатків.[16]

Крім універсальності та поширеності, MySQL має кілька важливих переваг серед інших систем .[25]

Деякі з цих переваг включають:

- Простота використання: MySQL має простий процес встановлення, а також плагіни та утиліти, які спрощують роботу з базами даних.
- Широка функціональність: MySQL надає майже всі необхідні інструменти для реалізації різноманітних проектів.
- Безпека: Система має вбудовані функції безпеки, які працюють за замовчуванням.
- Масштабованість: MySQL є універсальною базою даних і може легко працювати з як малими, так і великими обсягами даних.
- Швидкість: Система відзначається високою продуктивністю, завдяки спрощенню деяких стандартів, які використовуються в ній.[18]

Однак, MySQL має обмеження у своїй функціональності, які можуть унеможливлювати його використання для програм з певними вимогами. Деякі з недоліків MySQL включають:

- Недостатня надійність: MySQL може поступатися іншим базам даних щодо надійності певних процесів обробки даних, таких як зв'язок та аудит.
- Низькі темпи розробки: Як і багато інших програмних продуктів з відкритим кодом, MySQL може не мати певної технічної досконалості, що впливає на ефективність процесів розробки.[23]

2.3. Типи команд SQL

SQL, відповідно до концепції операцій, орієнтованих на табличне представлення даних, має невеликий набір команд. Цей мова може використовуватися як для виконання запитів до даних, так і для побудови прикладних програм.[2]

У мові SQL існують різні категорії команд, які виконують різні функції, включаючи побудову та маніпулювання об'єктами бази даних, завантаження даних в таблиці, оновлення та видалення існуючої інформації, виконання запитів до бази даних, управління доступом та загальне адміністрування.[20]

Зважаючи на ваше прохання, надаю більш детальний опис кожної команди SQL:

1. DDL (Data Definition Language) - мова визначення даних:

- CREATE TABLE: створює нову таблицю в базі даних з вказаною структурою.
- ALTER TABLE: змінює структуру існуючої таблиці, додаючи, змінюючи або видаляючи колонки.
- DROP TABLE: видаляє таблицю з бази даних разом з усією її структурою та даними.
- CREATE INDEX: створює індекс для швидкого доступу до даних у таблиці.
- ALTER INDEX: змінює властивості існуючого індексу, наприклад, додає або видаляє колонки з індексу.
- DROP INDEX: видаляє індекс з таблиці.

2. DML (Data Manipulation Language) - мова маніпулювання даними:

- INSERT: вставляє нові рядки даних в таблицю або таблиці.
- UPDATE: змінює існуючі рядки даних в таблиці, оновлюючи їх значення.

- DELETE: видаляє рядки даних з таблиці згідно з вказаними умовами.

3. DQL (Data Query Language) - мова запитів:

- SELECT: вибирає дані з однієї або кількох таблиць бази даних з використанням різних умов і сортування.

4. DCL (Data Control Language) - мова управління даними:

- GRANT: надає користувачам права доступу до об'єктів бази даних, таких як таблиці, представлення або процедури.
- REVOKE: відбирає або знімає права доступу користувачів до об'єктів бази даних.

5. Команди адміністрування даних:

Команди адміністрування даних надають можливість керувати та аналізувати операції бази даних, такі як резервне копіювання та відновлення даних, контроль транзакцій та аналіз продуктивності системи.[19]

Ці команди дозволяють розробникам та адміністраторам ефективно керувати структурою бази даних, маніпулювати даними, виконувати запити та контролювати доступ до інформації.[22]

2.4. Програмне забезпечення для хостингу

Для створення локального серверу, на якому буде зберігатися БД було вирішено використовувати програмне забезпечення MAMP.

Рис. 2.1 Логотип MAMP

MAMP - це зручний комплект програмного забезпечення, який містить у собі все необхідне для локальної розробки та тестування веб-сайтів. Встановлюючись з одного пакету, MAMP автоматично налаштовується і перетворює ваш комп'ютер в локальний сервер. Цей інструмент дозволяє розробникам зосередитись на створенні і відладці сайтів, не витрачаючи час на окреме налаштування різних компонентів.

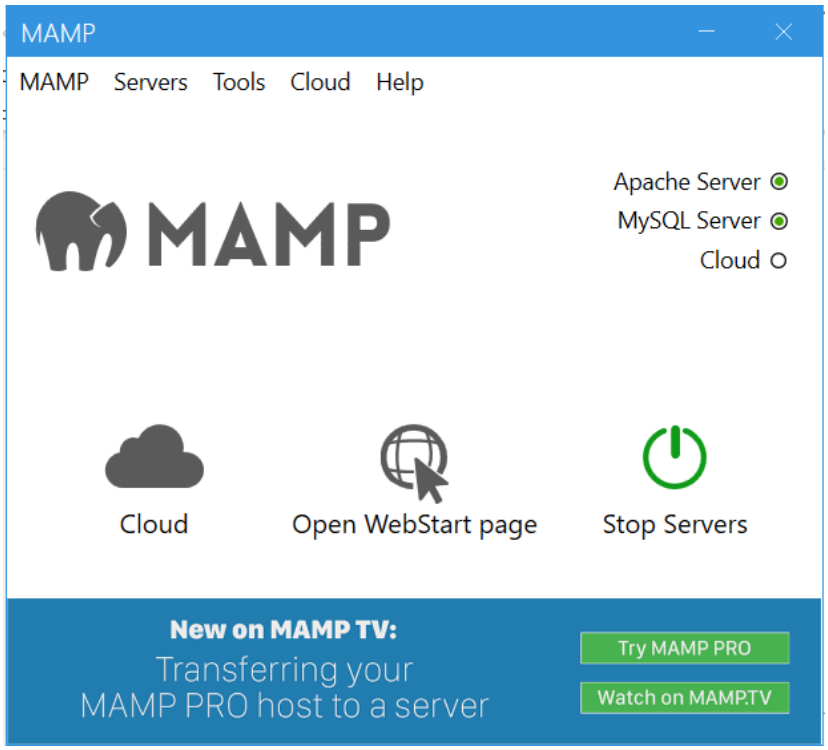


Рис. 2.2 Робочий інтерфейс MAMP

При запуску цього програмного забезпечення, сервер створюється автоматично, та через MAMP ви можете відкрити їх сайт(кнопка по центру робочого інтерфейсу MAMP на рис.2.3) і дізнатися усі дані вашого серверу, необхідні для підключення для нього, такі як хост, порт, ім’я та пароль серверу.

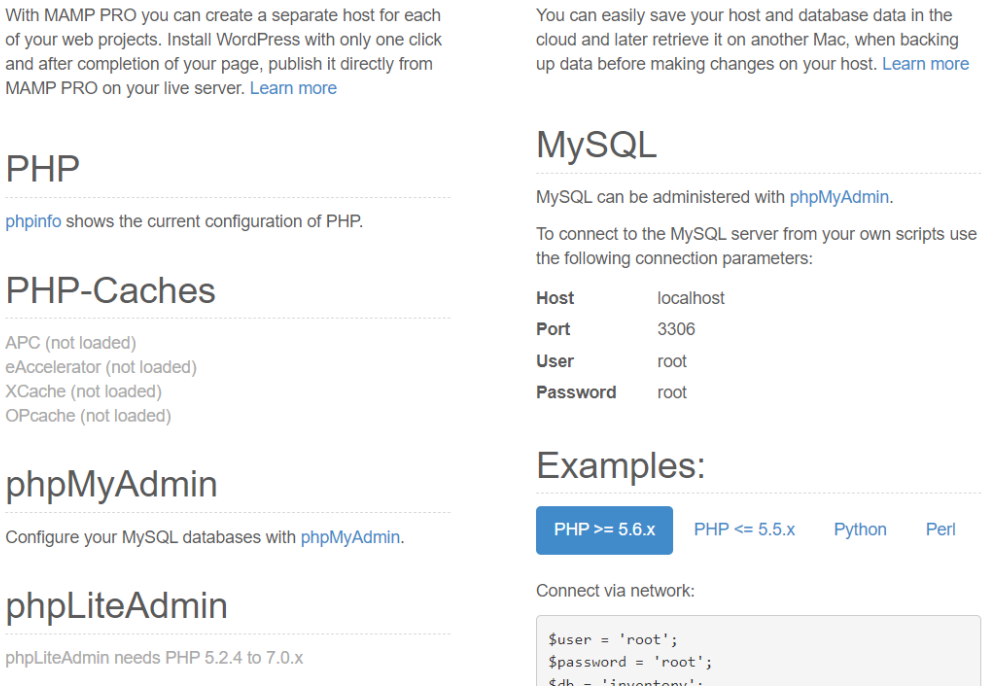


Рис.2.3 Сайт MAMP з даними адреси, ім’я та пароля від вашого серверу

Також з цього сайту ви можете перейти до веб-додатку phpMyAdmin, де ви вже зможете створити БД та таблиці в них, та проводити операції з цими БД.

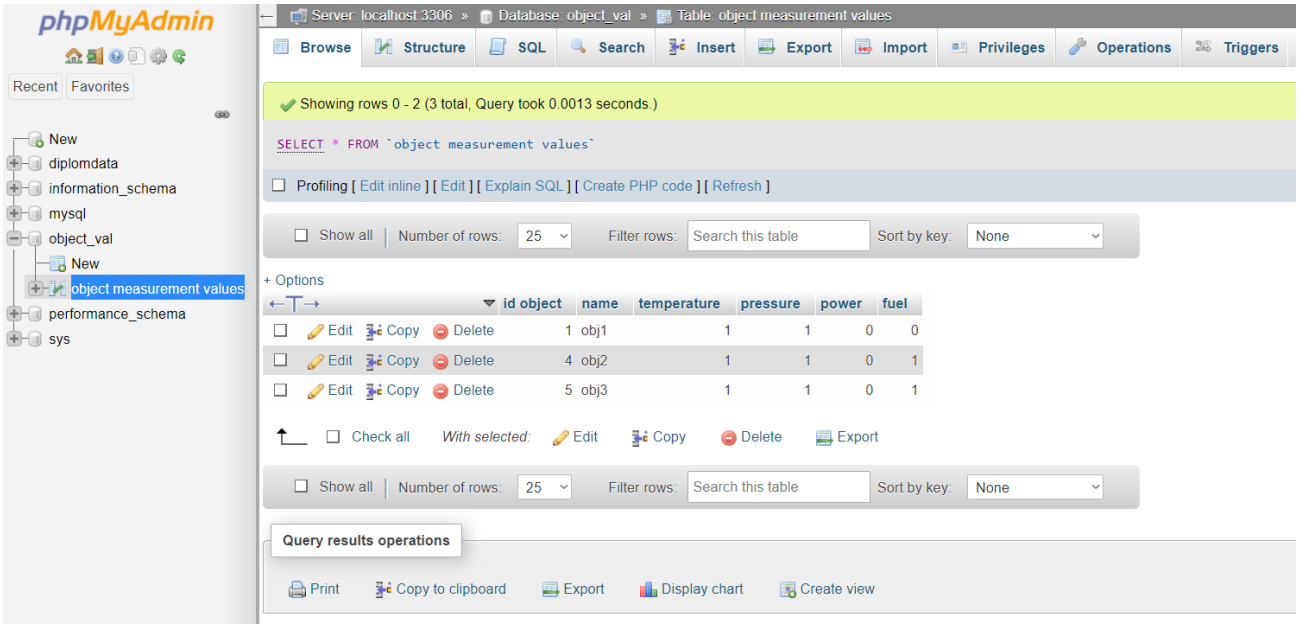


Рис.2.4 Інтерфейс phpMyAdmin для роботи з БД та таблицями в них

2.5.Створення та налаштування таблиці в базі даних для інтеграції її в додаток автоматизованого моніторингу

При створенні таблиці в БД, було створено 6 колонок, їх назвали “id object”, “name”, “temperature”, “pressure”, “power”, та “fuel”.

Для колонки id object були встановлені спеціальні налаштування: тип даних INT; максимальна довжина змінної – 11 символів; атрибут – UNSIGNED, який не дозволяє змінній бути від’ємною; індекс – UNIQUE, який не дозволяє змінній мати таке ж значення, як в іншій колонці; для автоінкременту ставимо true, щоб при створенні кожного нового об’єкту в БД, значення новоствореного id object збільшувалося за лічильником на 1.

Всі інші змінні не мають спеціальних налаштувань: типом даних для name є VARCHAR, для temperature, pressure, power та fuel мають тип даних INT, атрибут – UNSIGNED.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Основна ідея даної системи – зареєструвати технологічний об’єкт та обрані для нього характеристики, зберегти цю інформацію в БД, зчитувати поточний стан цього об’єкту та подавати ці дані користувачу для перегляду. При відсутності необхідності спостереження за зареєстрованим об’єктом – видалити його з листу об’єктів.

3.1. Створений клас OR

Через те, що програма не зможе ефективно вести роботу через зависання при підключенні та відключені від БД, було вирішено створити в програмі свій власний внутрішній архів. Він створюється та копіює інформацію з БД при включенні додатку, оновлюється лише при операціях реєстрації та видалення об’єктів з БД.[8]

Для збільшення безпеки коду та більш зручної взаємодії з БД під час програмування, було вирішено створити спеціальний клас OR.cs(Object Registration), який буде відповідати за відкриття та закриття з’єднання з БД, а також «повертати з’єднання» з нею.[11]

```
using MySql.Data.MySqlClient;
```

Підключається бібліотека MySql.Data.MySqlClient, яка надає доступ до функцій для роботи з базою даних MySQL.

```
namespace Diplom1
{
    internal class OR
    {
        MySqlConnection connection = new
MySqlConnection("server=localhost;port=3306;username=root;password=root;database=ob
ject_val");
```

Визначається простір імен Diplom1, а всередині нього оголошується внутрішній клас OR. В цьому класі створюється об'єкт MySqlConnection з назвою connection, який ініціалізується рядком підключення до бази даних MySQL. Цей рядок містить параметри підключення, такі як сервер (localhost), порт (3306), ім'я користувача (root), пароль (root) і назва бази даних (object_val).

```
public void openConnection()
        {
            if (connection.State == System.Data.ConnectionState.Closed)
                connection.Open();
        }
```

Визначається метод openConnection(), який відповідає за відкриття з'єднання з базою даних. Він перевіряє поточний стан з'єднання (connection.State) і, якщо він дорівнює System.Data.ConnectionState.Closed (закритий), то виконує відкриття з'єднання (connection.Open()).

```
public void closeConnection()
        {
            if (connection.State == System.Data.ConnectionState.Open)
                connection.Close();
        }
```

Визначається метод closeConnection(), який відповідає за закриття з'єднання з базою даних. Він перевіряє поточний стан з'єднання (connection.State) і, якщо він дорівнює System.Data.ConnectionState.Open (відкритий), то виконує закриття з'єднання (connection.Close()).

```
public MySqlConnection getConnection()
        {
            return connection;
        }
    }
}
```

Визначається метод getConnection(), який повертає об'єкт з'єднання connection. Цей метод дозволяє отримати доступ до з'єднання з базою даних для виконання запитів та інших операцій.

3.2. Опис роботи Form1

За виконання головного функціоналу програми відповідає Form1, а саме надання можливості користувачу почати процес додавання або видалення об’єкту спостереження в листі об’єктів, обрати цікавий користувачу об’єкт в листі та виведення на екран даних про обраний об’єкт. Через те, що під час виконання роботи ми не маємо доступу до будь-якого технологічного обладнання з виведенням інформації про свій стан до «хмари», програма є ілюстративною, та містить в собі генератори даних, які відповідають за демонстрацію функціонування програми.

```
using MySql.Data.MySqlClient;
using System;
using System.Collections.Generic;
using System.Data;
using System.Threading;
using System.Windows.Forms;
```

Оголошуються необхідні простори імен та бібліотеки для роботи програми. MySql.Data.MySqlClient використовується для роботи з базою даних MySQL. System містить базові класи та типи даних. System.Collections.Generic надає загальні колекції, такі як словники та списки. System.Data використовується для роботи з даними та базами даних. System.Threading дозволяє працювати з потоками. System.Windows.Forms містить класи для створення графічного інтерфейсу користувача.

```
namespace Diplom1
{
    public partial class Form1 : Form
    {
        private bool isFormLoaded = false;

        public Form1()
        {
            InitializeComponent();
            Shown += Form1_Shown;
        }
```

Клас Form1 є головною формою програми. В ньому визначено приватне поле isFormLoaded, яке відстежує, чи форма вже завантажена. Конструктор Form1 викликає метод InitializeComponent() для ініціалізації компонентів форми. Подія Shown призначена обробнику Form1_Shown, який виконується після показу форми на екрані.

```
static double temp_generator()
        {
            double variable = 65.0; // Початкове значення змінної
            Random random = new Random();

            while (true)
            {
                int number = random.Next(0, 1001); // Генеруємо випадкове число в
діапазоні від 0 до 1000

                // Змінюємо значення змінної згідно з умовами
                if (number >= 0 && number <= 400)
                {
                    variable += 0.3;
                }
                else if (number >= 401 && number <= 800)
                {
                    variable -= 0.3;
                }
                else if (number >= 801 && number <= 900)
                {
                    variable += 1;
                }
                else if (number >= 901 && number <= 1000)
                {
                    variable -= 1;
                }

                // Перевіряємо обмеження на мінімальне та максимальне значення
змінної
                if (variable < 60)
                {
                    variable += 3;
                }
                else if (variable > 75)
                {
                    variable -= 3;
                }
```

```
                // Робимо паузу в 1 секунду
                Thread.Sleep(1000);
            }
        }

        static double press_generator()
        {
            double variable = 250.0; // Початкове значення змінної
            Random random = new Random();

            while (true)
            {
                int number = random.Next(0, 1001); // Генеруємо випадкове число в
діапазоні від 0 до 1000

                // Змінюємо значення змінної згідно з умовами
                if (number >= 0 && number <= 400)
                {
                    variable += 2.3;
                }
                else if (number >= 401 && number <= 800)
                {
                    variable -= 2.3;
                }
                else if (number >= 801 && number <= 900)
                {
                    variable += 7.6;
                }
                else if (number >= 901 && number <= 1000)
                {
                    variable -= 7.6;
                }

                // Перевіряємо обмеження на мінімальне та максимальне значення
змінної
                if (variable < 101)
                {
                    variable += 15;
                }
                else if (variable > 405)
                {
                    variable -= 15;
                }

                // Робимо паузу в 1 секунду
                Thread.Sleep(1000);
            }
```

```
        }

        static double power_generator()
        {
            double variable = 150.0; // Початкове значення змінної
            Random random = new Random();

            while (true)
            {
                int number = random.Next(0, 1001); // Генеруємо випадкове число в
діапазоні від 0 до 1000

                // Змінюємо значення змінної згідно з умовами
                if (number >= 0 && number <= 400)
                {
                    variable += 2.3;
                }
                else if (number >= 401 && number <= 800)
                {
                    variable -= 2.3;
                }
                else if (number >= 801 && number <= 900)
                {
                    variable += 7.6;
                }
                else if (number >= 901 && number <= 1000)
                {
                    variable -= 7.6;
                }

                // Перевіряємо обмеження на мінімальне та максимальне значення
змінної
                if (variable < 120)
                {
                    variable += 10;
                }
                else if (variable > 170)
                {
                    variable -= 10;
                }

                // Робимо паузу в 1 секунду
                Thread.Sleep(1000);
            }
        }

        static double fuel_generator()
```

```
        {
            double variable = 80.0; // Початкове значення змінної
            Random random = new Random();

            while (true)
            {
                int number = random.Next(0, 501); // Генеруємо випадкове число в
діапазоні від 0 до 1000

                // Змінюємо значення змінної згідно з умовами
                if (number >= 0 && number <= 200)
                {
                    variable -= 1;
                }
                else if (number >= 201 && number <= 400)
                {
                    variable -= 2;
                }
                else if (number >= 401 && number <= 500)
                {
                    variable -= 4.5;
                }

                // Перевіряємо обмеження на мінімальне та максимальне значення
змінної
                if (variable < 65)
                {
                    variable = 100;
                }

                // Робимо паузу в 1 секунду
                Thread.Sleep(1000);
            }
        }
```

Функція temp_generator є генератором даних, який відповідає за імітацію потоку даних температури з котла опалення. Вона є статичною і повертає значення типу double. Початкове значення змінної variable встановлено на 65.0. Створюється об'єкт Random для генерації випадкових чисел.

У безкінечному циклі виконується наступне:

1. Генерується випадкове число number в діапазоні від 0 до 1000.
2. Змінюється значення змінної variable відповідно до умов:

- Якщо number знаходиться в діапазоні від 0 до 400, variable збільшується на 0.3.
- Якщо number знаходиться в діапазоні від 401 до 800, variable зменшується на 0.3.
- Якщо number знаходиться в діапазоні від 801 до 900, variable збільшується на 1.
- Якщо number знаходиться в діапазоні від 901 до 1000, variable зменшується на 1.

3. Перевіряється обмеження на мінімальне та максимальне значення змінної variable:

- Якщо variable менше 60, воно збільшується на 3.
- Якщо variable більше 75, воно зменшується на 3.

4. Виконується пауза у 1 секунду.

Функція press_generator є генератором даних, який відповідає за імітацію потоку даних тиску з котла опалення. Початкове значення змінної variable встановлено на 250.0. Умови для зміни значень variable виглядають так:

- Якщо випадкове число number знаходиться в діапазоні від 0 до 400, variable збільшується на 2.3.
- Якщо number знаходиться в діапазоні від 401 до 800, variable зменшується на 2.3.
- Якщо number знаходиться в діапазоні від 801 до 900, variable збільшується на 7.6.
- Якщо number знаходиться в діапазоні від 901 до 1000, variable зменшується на 7.6.

Обмеження на мінімальне та максимальне значення змінної variable такі:

- Якщо variable менше 101, воно збільшується на 15.
- Якщо variable більше 405, воно зменшується на 15.

Функція power_generator є генератором даних, який відповідає за імітацію потоку даних потужності з дизельного генератора. Початкове значення змінної variable встановлено на 150.0. Умови для зміни значень variable виглядають так:

- Якщо випадкове число number знаходиться в діапазоні від 0 до 400, variable збільшується на 2.3.
- Якщо number знаходиться в діапазоні від 401 до 800, variable зменшується на 2.3.
- Якщо number знаходиться в діапазоні від 801 до 900, variable збільшується на 7.6.
- Якщо number знаходиться в діапазоні від 901 до 1000, variable зменшується на 7.6.

Обмеження на мінімальне та максимальне значення змінної variable такі:

- Якщо variable менше 120, воно збільшується на 10.
- Якщо variable більше 170, воно зменшується на 10.

Функція fuel_generator є генератором даних, який відповідає за імітацію потоку даних прокількість палива та є універсальним для усіх технологічних приладів з системою паливопостачання. Початкове значення змінної variable встановлено на 80.0. Умови для зміни значень variable виглядають так:

- Якщо випадкове число number знаходиться в діапазоні від 0 до 200, variable зменшується на 1.
- Якщо number знаходиться в діапазоні від 201 до 400, variable зменшується на 2.
- Якщо number знаходиться в діапазоні від 401 до 500, variable зменшується на 4.5.

Обмеження на мінімальне значення змінної variable таке:

- Якщо variable менше 65, воно встановлюється на 100.

У кожній з цих функцій також виконується пауза у 1 секунду, щоб затримати виконання наступної ітерації циклу. Значення variable повертається з функцій.

```
private void створитиToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Form2 form2 = new Form2();
            form2.FormClosed += Form2_FormClosed;
            form2.ShowDialog();
        }

        private void Form2_FormClosed(object sender, FormClosedEventArgs e)
        {
            MessageBox.Show("Вікно Form2 закрите!");
            UpdateTable();
        }

        private void Form3_FormClosed(object sender, FormClosedEventArgs e)
        {
            MessageBox.Show("Вікно Form3 закрите!");
            UpdateTable();
        }
```

Цей код містить обробники подій для закриття форм.

У методі створитиToolStripMenuItem_Click створюється форма Form2 і встановлюється обробник події FormClosed, який викликає метод Form2_FormClosed при закритті форми Form2. Далі викликається метод ShowDialog(), який відображає форму Form2 в модальному режимі.

Метод Form2_FormClosed є обробником події закриття форми Form2. При закритті форми виводиться повідомлення MessageBox, яке повідомляє користувача про закриття форми, та викликається метод UpdateTable().

Аналогічно, є метод Form3_FormClosed, який є обробником події закриття форми Form3. При закритті форми виводиться повідомлення MessageBox, яке повідомляє користувача про закриття форми, та викликається метод UpdateTable().

Ці методи дозволяють обробляти події закриття окремих форм і виконувати необхідні дії після їх закриття.

```
private Dictionary<string, double> temperatureVariables = new Dictionary<string,
double>();
        private Dictionary<string, double> pressureVariables = new
Dictionary<string, double>();
        private Dictionary<string, double> powerVariables = new Dictionary<string,
double>();
        private Dictionary<string, double> fuelVariables = new Dictionary<string,
double>();
        private Dictionary<string, int> counters = new Dictionary<string, int>();
        private List<ProgressBar> progressBars = new List<ProgressBar>();
        private List<Label> labels = new List<Label>();

        private void Form1_Shown(object sender, EventArgs e)
        {
            // Створюємо таблицю, заповнюємо змінні та лічильники
            CreateTable();
            FillVariablesAndCounters();
            UpdateTable();
        }

        private void CreateTable()
        {
            string sourceConnectionString =
"server=localhost;port=3306;username=root;password=root;database=object_val";
            string destinationTableName = "object measurement values";
            DataTable newTable = new DataTable();

            using (MySqlConnection sourceConnection = new
MySqlConnection(sourceConnectionString))
            {
                sourceConnection.Open();

                // Отримуємо структуру таблиці з бази даних
                string schemaQuery = $"SELECT * FROM {destinationTableName} LIMIT
1";
                using (MySqlDataAdapter schemaAdapter = new
MySqlDataAdapter(schemaQuery, sourceConnection))
                {
                    schemaAdapter.FillSchema(newTable, SchemaType.Source);
                }

                // Створюємо нову таблицю в додатку
                newTable.TableName = "NewTable";
```

```
                // Заповнюємо нову таблицю даними з бази даних
                string dataQuery = $"SELECT * FROM {destinationTableName}";
                using (MySqlDataAdapter dataAdapter = new
MySqlDataAdapter(dataQuery, sourceConnection))
                {
                    dataAdapter.Fill(newTable);
                }

                sourceConnection.Close();
            }

            // Заповнення ListBox значеннями з колонки "name"
            foreach (DataRow row in newTable.Rows)
            {
                string nameValue = row["name"].ToString();
                ObjectList.Items.Add(nameValue);

                string id = row["id"].ToString();

                if (row["temperature"].ToString() == "1")
                {
                    double temperatureVariable = temp_generator();
                    temperatureVariables.Add($"temp_{id}", temperatureVariable);
                    counters.Add($"counter_{id}", 0);
                }

                if (row["pressure"].ToString() == "1")
                {
                    double pressureVariable = press_generator();
                    pressureVariables.Add($"press_{id}", pressureVariable);
                    counters.Add($"counter_{id}", 0);
                }

                if (row["power"].ToString() == "1")
                {
                    double powerVariable = power_generator();
                    powerVariables.Add($"power_{id}", powerVariable);
                    counters.Add($"counter_{id}", 0);
                }

                if (row["fuel"].ToString() == "1")
                {
                    double fuelVariable = fuel_generator();
                    fuelVariables.Add($"fuel_{id}", fuelVariable);
                    counters.Add($"counter_{id}", 0);
                }
            }
```

```
        }

        private void FillVariablesAndCounters()
        {
            ObjectList.Items.Clear();

            foreach (var kvp in counters)
            {
                string id = kvp.Key.Substring(8);
                int counter = kvp.Value;

                string name = GetObjectNameById(id);
                ObjectList.Items.Add(name);

                counters[kvp.Key] = counter;
            }
        }

        private string GetObjectNameById(string id)
        {
            string sourceConnectionString =
"server=localhost;port=3306;username=root;password=root;database=object_val";
            string tableName = "object measurement values";
            string name = "";

            using (MySqlConnection sourceConnection = new
MySqlConnection(sourceConnectionString))
            {
                sourceConnection.Open();

                string query = $"SELECT name FROM {tableName} WHERE id = {id}";
                using (MySqlCommand command = new MySqlCommand(query,
sourceConnection))
                {
                    using (MySqlDataReader reader = command.ExecuteReader())
                    {
                        if (reader.Read())
                        {
                            name = reader["name"].ToString();
                        }
                    }
                }

                sourceConnection.Close();
            }

            return name;
```

```
        }

        private void ObjectList_SelectedIndexChanged(object sender, EventArgs e)
        {
            // Видалення попередніх ProgressBar та міток
            RemoveExistingProgressBarsAndLabels();

            // Отримання вибраного об'єкта зі списку
            string selectedObjectName = ObjectList.SelectedItem.ToString();

            // Отримання відповідних змінних та лічильників для вибраного об'єкта
            List<double> selectedTemperatureVariables = new List<double>();
            List<double> selectedPressureVariables = new List<double>();
            List<double> selectedPowerVariables = new List<double>();
            List<double> selectedFuelVariables = new List<double>();
            int counterValue = 0;

            foreach (var kvp in temperatureVariables)
            {
                if (kvp.Key.StartsWith($"temp_{selectedObjectName}"))
                {
                    selectedTemperatureVariables.Add(kvp.Value);
                    counterValue++;
                }
            }

            foreach (var kvp in pressureVariables)
            {
                if (kvp.Key.StartsWith($"press_{selectedObjectName}"))
                {
                    selectedPressureVariables.Add(kvp.Value);
                    counterValue++;
                }
            }

            foreach (var kvp in powerVariables)
            {
                if (kvp.Key.StartsWith($"power_{selectedObjectName}"))
                {
                    selectedPowerVariables.Add(kvp.Value);
                    counterValue++;
                }
            }

            foreach (var kvp in fuelVariables)
            {
                if (kvp.Key.StartsWith($"fuel_{selectedObjectName}"))
```

```
                {
                    selectedFuelVariables.Add(kvp.Value);
                    counterValue++;
                }
            }

            // Створення ProgressBar та міток залежно від значення лічильника
            for (int i = 0; i < counterValue; i++)
            {
                ProgressBar progressBar = new ProgressBar();
                progressBar.Left = 250;
                progressBar.Top = 50 + (i * 100);
                progressBar.Width = 200;
                progressBar.Visible = false;

                Label label = new Label();
                label.Left = progressBar.Left + progressBar.Width + 10;
                label.Top = progressBar.Top + (progressBar.Height / 2) - 10;
                label.AutoSize = true;

                string labelSuffix = GetLabelSuffix(i);
                label.Text = labelSuffix;

                panel1.Controls.Add(progressBar);
                panel1.Controls.Add(label);

                progressBars.Add(progressBar);
                labels.Add(label);
            }

            // Оновлення значень ProgressBar
            UpdateProgressBarValues(selectedTemperatureVariables,
selectedPressureVariables, selectedPowerVariables, selectedFuelVariables);
        }

        private string GetLabelSuffix(int index)
        {
            string labelSuffix = "";

            if (index % 4 == 0)
            {
                labelSuffix = "°C";
            }
            else if (index % 4 == 1)
            {
                labelSuffix = "кПа";
            }
```

```
            else if (index % 4 == 2)
            {
                labelSuffix = "кВт";
            }
            else if (index % 4 == 3)
            {
                labelSuffix = "% палива";
            }

            return labelSuffix;
        }


        private void RemoveExistingProgressBarsAndLabels()
        {
            // Видаляємо попередні ProgressBar та мітки з панелі
            foreach (ProgressBar progressBar in progressBars)
            {
                panel1.Controls.Remove(progressBar);
                progressBar.Dispose();
            }

            foreach (Label label in labels)
            {
                panel1.Controls.Remove(label);
                label.Dispose();
            }

            // Очищаємо списки ProgressBar та міток
            progressBars.Clear();
            labels.Clear();
        }
```

Ця частина є ключовою частиною всієї програми. Код створює внутрішню таблицю-архів, яка копіює структуру та інформацію про таблицю з БД, виводить усі найменування об’єктів до листа та створює для кожного об’єкту змінні, в яких зберігається інформація про поточний стан об’єкту та створює змінні-лічильники для виведення циферблатів з поточними даними в характерній для обраного об’єкту кількості. Також цей код відповідає за графічне відтворення лічильників та видалення зайвих при зміні об’єкту спостереження у листі об’єктів ObjectList.

1. Змінні:

- temperatureVariables, pressureVariables, powerVariables, fuelVariables - словники, що зберігають значення температури, тиску, потужності та палива для кожного об'єкта.
- counters - словник, що зберігає значення лічильників для кожного об'єкта.
- progressBars - список, що містить об'єкти типу ProgressBar.
- labels - список, що містить об'єкти типу Label.

2. Метод Form1_Shown:

- Викликається при показі форми Form1.
- Створює таблицю, заповнює змінні та лічильники та оновлює таблицю.

3. Метод CreateTable:

- Використовує підключення до бази даних для отримання структури таблиці та заповнення нової таблиці в додатку.
- Заповнює ObjectList значеннями з колонки "name".
- Для кожного рядка в таблиці:
  - Отримує значення з колонки "name".
  - Додає значення до ObjectList.
  - Якщо колонка "temperature" має значення "1", генерується випадкове значення для температури та додається до temperatureVariables разом зі значенням лічильника.
  - Аналогічно для колонок "pressure", "power" і "fuel".

4. Метод FillVariablesAndCounters:

- Очищує ObjectList.
- Для кожного лічильника в словнику counters:
  - Отримує значення лічильника.
  - Знаходить відповідний об'єкт за ідентифікатором.

- Додає назву об'єкта до ObjectList.
- Оновлює значення лічильника в словнику counters.

5. Метод GetObjectNameById:

- Отримує назву об'єкта за його ідентифікатором з бази даних.

6. Метод ObjectList_SelectedIndexChanged:

- Викликається при виборі елемента з ObjectList.
- Видаляє попередні ProgressBar та мітки.
- Отримує назву вибраного об'єкта.
- Отримує відповідні значення змінних та лічильників для вибраного об'єкта.
- Створює ProgressBar та мітки залежно від значення лічильника.
- Оновлює значення ProgressBar.

7. Метод GetLabelSuffix:

- Отримує суфікс для мітки залежно від індексу.
- Залежно від значення індексу повертає відповідний суфікс.

8. Метод RemoveExistingProgressBarsAndLabels:

- Видаляє попередні ProgressBar та мітки з panel1.
- Очищує списки progressBars та labels.

```
        private void UpdateProgressBarValues(List<double> temperatureValues,
List<double> pressureValues, List<double> powerValues, List<double> fuelValues)
        {
            // Оновлення значень ProgressBar
            int progressBarIndex = 0;

            foreach (double temperatureValue in temperatureValues)
            {
                ProgressBar progressBar = progressBars[progressBarIndex];
                int progressPercentage = Convert.ToInt32(temperatureValue * 100);
                progressBar.Value = progressPercentage;
                progressBar.Visible = true;

                progressBarIndex++;
            }
```

```
            foreach (double pressureValue in pressureValues)
            {
                ProgressBar progressBar = progressBars[progressBarIndex];
                int progressPercentage = Convert.ToInt32(pressureValue * 100);
                progressBar.Value = progressPercentage;
                progressBar.Visible = true;

                progressBarIndex++;
            }

            foreach (double powerValue in powerValues)
            {
                ProgressBar progressBar = progressBars[progressBarIndex];
                int progressPercentage = Convert.ToInt32(powerValue * 100);
                progressBar.Value = progressPercentage;
                progressBar.Visible = true;

                progressBarIndex++;
            }

            foreach (double fuelValue in fuelValues)
            {
                ProgressBar progressBar = progressBars[progressBarIndex];
                int progressPercentage = Convert.ToInt32(fuelValue * 100);
                progressBar.Value = progressPercentage;
                progressBar.Visible = true;

                progressBarIndex++;
            }

            // Видаляємо зайві ProgressBar та мітки, якщо їх кількість менша за
поточне значення індексу ProgressBar
            if (progressBars.Count > progressBarIndex)
            {
                for (int i = progressBarIndex; i < progressBars.Count; i++)
                {
                    ProgressBar progressBar = progressBars[i];
                    panel1.Controls.Remove(progressBar);
                    progressBar.Dispose();
                }

                progressBars.RemoveRange(progressBarIndex, progressBars.Count -
progressBarIndex);
            }

            if (labels.Count > progressBarIndex)
            {
```

```
                for (int i = progressBarIndex; i < labels.Count; i++)
                {
                    Label label = labels[i];
                    panel1.Controls.Remove(label);
                    label.Dispose();
                }

                labels.RemoveRange(progressBarIndex, labels.Count -
progressBarIndex);
            }
        }

        private void UpdateTable()
        {
            string sourceConnectionString =
"server=localhost;port=3306;username=root;password=root;database=object_val";
            string destinationTableName = "object measurement values";
            DataTable newTable = new DataTable();

            using (MySqlConnection sourceConnection = new
MySqlConnection(sourceConnectionString))
            {
                sourceConnection.Open();

                // Отримуємо структуру таблиці з бази даних
                string schemaQuery = $"SELECT * FROM {destinationTableName} LIMIT
1";
                using (MySqlDataAdapter schemaAdapter = new
MySqlDataAdapter(schemaQuery, sourceConnection))
                {
                    schemaAdapter.FillSchema(newTable, SchemaType.Source);
                }

                // Створюємо нову таблицю в додатку
                newTable.TableName = "NewTable";

                // Заповнюємо нову таблицю даними з бази даних
                string dataQuery = $"SELECT * FROM {destinationTableName}";
                using (MySqlDataAdapter dataAdapter = new
MySqlDataAdapter(dataQuery, sourceConnection))
                {
                    dataAdapter.Fill(newTable);
                }

                sourceConnection.Close();
            }
```

```
            // Видаляємо попередні дані зі списків та словників
            temperatureVariables.Clear();
            pressureVariables.Clear();
            powerVariables.Clear();
            fuelVariables.Clear();
            counters.Clear();

            // Очищуємо ListBox та видаляємо всі ProgressBar та мітки з панелі
            ObjectList.Items.Clear();
            RemoveExistingProgressBarsAndLabels();

            // Заповнення ListBox значеннями з колонки "name" та додавання змінних
та лічильників
            foreach (DataRow row in newTable.Rows)
            {
                string nameValue = row["name"].ToString();
                ObjectList.Items.Add(nameValue);

                string id = row["id"].ToString();

                if (row["temperature"].ToString() == "1")
                {
                    double temperatureVariable = temp_generator();
                    temperatureVariables.Add($"temp_{id}", temperatureVariable);
                    counters.Add($"counter_{id}", 0);
                }

                if (row["pressure"].ToString() == "1")
                {
                    double pressureVariable = press_generator();
                    pressureVariables.Add($"press_{id}", pressureVariable);
                    counters.Add($"counter_{id}", 0);
                }

                if (row["power"].ToString() == "1")
                {
                    double powerVariable = power_generator();
                    powerVariables.Add($"power_{id}", powerVariable);
                    counters.Add($"counter_{id}", 0);
                }

                if (row["fuel"].ToString() == "1")
                {
                    double fuelVariable = fuel_generator();
                    fuelVariables.Add($"fuel_{id}", fuelVariable);
                    counters.Add($"counter_{id}", 0);
                }
```

```
            }
        }
```

Метод UpdateProgressBarValues(List<double> temperatureValues, List <double> pressureValues, List<double> powerValues, List<double> fuelValues) оновлює значення ProgressBar.

Він отримує списки значень температури (temperatureValues), тиску (pressureValues), потужності (powerValues) і палива (fuelValues). Кожен список містить значення для відповідного об'єкта.

Метод працює наступним чином:

1. Ініціалізуємо змінну progressBarIndex зі значенням 0. Вона використовується для відстеження індексу ProgressBar.

2. Проходимо по кожному значенню temperatureValue у списку temperatureValues.

- Отримуємо ProgressBar за допомогою progressBars[progressBarIndex].
- Обчислюємо відсоток прогресу для ProgressBar, перетворюючи значення температури у ціле число.
- Встановлюємо значення ProgressBar на обчислений відсоток.
- Встановлюємо видимість ProgressBar на значення true.
- Збільшуємо progressBarIndex на 1.

3. Аналогічно проходимо по спискам pressureValues, powerValues і fuelValues, оновлюючи значення відповідних ProgressBar.

4. Перевіряємо, чи є деякі ProgressBar або мітки, які потрібно видалити. Якщо кількість ProgressBar у списку progressBars більша за progressBarIndex, то видаляємо зайві ProgressBar та мітки з panel1.

- Проходимо по ProgressBar, починаючи з індексу progressBarIndex.
- Видаляємо ProgressBar з panel1 та звільняємо його ресурси (progressBar.Dispose()).

5. Видаляємо зайві ProgressBar зі списку progressBars, використовуючи RemoveRange(progressBarIndex, progressBars.Count - progressBarIndex).

6. Аналогічно, якщо кількість міток у списку labels більша за progressBarIndex, видаляємо зайві мітки.

Метод UpdateTable() оновлює таблицю даних. Він працює наступним чином:

1. Ініціалізуємо рядок sourceConnectionString, який містить рядок підключення до бази даних.

2. Ініціалізуємо рядок destinationTableName, який містить назву таблиці в базі даних.

3. Створюємо новий об'єкт DataTable з назвою newTable.

4. Використовуючи MySqlConnection, встановлюємо з'єднання з базою даних за допомогою рядка підключення.

5. Відкриваємо з'єднання до бази даних.

6. Отримуємо структуру таблиці з бази даних за допомогою запиту schemaQuery. Використовуємо MySqlDataAdapter для заповнення схеми таблиці.

7. Встановлюємо назву нової таблиці як "NewTable".

8. Заповнюємо нову таблицю даними з бази даних за допомогою запиту dataQuery.

9. Закриваємо з'єднання до бази даних.

10. Очищуємо списки temperatureVariables, pressureVariables, powerVariables, fuelVariables та counters.

11. Очищуємо ListBox ObjectList та видаляємо всі ProgressBar та мітки з panel1, використовуючи метод RemoveExistingProgressBarsAndLabels().

12. Проходимо по кожному рядку таблиці newTable.

- Отримуємо значення стовпця "name" та додаємо його до ListBox ObjectList.

- Отримуємо значення стовпця "id".
- Якщо стовпець "temperature" дорівнює "1", то генеруємо значення температури за допомогою temp_generator() і додаємо його до списку temperatureVariables. Додаємо також лічильник для даного об'єкта до списку counters.
- Аналогічно для стовпців "pressure", "power" і "fuel".

```
        private string selectedObjectId;
        private void видалитиToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Form3 form3 = new Form3(selectedObjectId);
            form3.FormClosed += Form2_FormClosed;
            form3.ShowDialog();
        }
    }
}
```

Цей код містить змінну selectedObjectId і метод видалитиToolStripMenuItem_Click. Метод видалитиToolStripMenuItem_Click виконується при натисканні на пункт меню "видалитиToolStripMenuItem" і містить наступні кроки:

1. Створення нового екземпляра Form3 з аргументом selectedObjectId і надання йому обробника події FormClosed.
2. Відображення форми Form3 у модальному режимі за допомогою методу ShowDialog.

3.3. Опис роботи Form2

Form2.cs відповідає за функціонування вікна “Form2”, в якій користувач створює новий об’єкт та додає його до БД. Також є функція, яка перевіряючи всю таблицю БД не дозволяє новостворенному об’єкту повторювати ім’я іншого вже існуючого об’єкта.

```
using MySql.Data.MySqlClient;
using System;
```

```
using System.Data;
using System.Windows.Forms;
```

Код використовує бібліотеку MySql.Data.MySqlClient для підключення до MySQL бази даних. Він також включає простори імен System та System.Windows.Forms, які потрібні для роботи з загальними класами та для створення графічного інтерфейсу користувача за допомогою Windows Forms. Код містить функціонал для роботи з даними у базі даних та для відображення їх у вікні програми.

```
namespace Diplom1
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }

        private void ok_button_Click(object sender, EventArgs e)
        {
            if (nameBox.Text == "")
            {
                MessageBox.Show("Введіть ім'я для об'єкту"); // Повідомлення про
помилку, якщо поле для імені порожнє
                return;
            }

            if (!temCheckBox.Checked && !pressCheckBox.Checked &&
!powerCheckBox.Checked && !fuelCheckBox.Checked)
            {
                MessageBox.Show("Оберіть, які дані повинні відстежуватися в цьому
об'єкті"); // Повідомлення про помилку, якщо жодне поле не вибране для
відстежування
                return;
            }

            if (checkname())
                return;

            OR or = new OR();
```

```
            MySqlCommand command = new MySqlCommand("INSERT INTO `object
measurement values` ( `name`, `temperature`, `pressure`, `power`, `fuel`) VALUES
(@name, @temperature, @pressure, @power, @fuel);", or.getConnection());

            command.Parameters.Add("@name", MySqlDbType.VarChar).Value =
nameBox.Text;
            command.Parameters.Add("@temperature", MySqlDbType.Int32).Value =
temCheckBox.Checked;
            command.Parameters.Add("@pressure", MySqlDbType.Int32).Value =
pressCheckBox.Checked;
            command.Parameters.Add("@power", MySqlDbType.Int32).Value =
powerCheckBox.Checked;
            command.Parameters.Add("@fuel", MySqlDbType.Int32).Value =
fuelCheckBox.Checked;

            or.openConnection();

            if (command.ExecuteNonQuery() == 1)
            {
                MessageBox.Show("Об'єкт створено"); // Повідомлення про успішне
створення об'єкта
                this.Close();
            }
            else
            {
                MessageBox.Show("Об'єкт не створено"); // Повідомлення про невдале
створення об'єкта
            }
            or.closeConnection();
        }
```

Даний код належить до простору імен "Diplom1" і включає частковий клас "Form2", який є формою вікна програми. У формі присутній графічний інтерфейс, який складається з елементів керування, таких як текстове поле "nameBox" і прапорці "temCheckBox", "pressCheckBox", "powerCheckBox" і "fuelCheckBox".

Конструктор класу Form2 ініціалізує компоненти форми за допомогою методу InitializeComponent().

Метод ok_button_Click викликається при натисканні кнопки з ім'ям "ok_button". В цьому методі перевіряється, чи введено значення в поле для імені "nameBox". Якщо поле порожнє, виводиться повідомлення про помилку за

допомогою MessageBox.Show() і виконання методу припиняється за допомогою return.

Також перевіряється, чи вибрані прапорці "temCheckBox", "pressCheckBox", "powerCheckBox" або "fuelCheckBox". Якщо жоден з них не вибраний, виводиться повідомлення про помилку за допомогою MessageBox.Show() і виконання методу припиняється за допомогою return.

Є створення об'єкту класу OR, який використовується для взаємодії з базою даних MySQL. Створюється команда MySqlCommand для вставки даних в таблицю "object measurement values" бази даних. Параметри команди заповнюються значеннями з елементів форми.

Виконується відкриття підключення до бази даних за допомогою or.openConnection(). Виконується команда ExecuteNonQuery(), яка виконує SQL-запит до бази даних для вставки даних. Якщо команда успішно виконана (повернула 1), виводиться повідомлення про успішне створення об'єкта за допомогою MessageBox.Show() і закривається поточна форма this.Close(). У протилежному випадку виводиться повідомлення про невдале створення об'єкта.

Закривається підключення до бази даних за допомогою or.closeConnection().

```
 public Boolean checkname()// Повідомлення про помилку, якщо ім'я вже
використовується
        {
            OR or = new OR();

            DataTable table = new DataTable();

            MySqlDataAdapter adapter = new MySqlDataAdapter();

            MySqlCommand command = new MySqlCommand("SELECT * FROM 'object
measurement values' WHERE 'name'=@nO");

            command.Parameters.Add("nO", MySqlDbType.VarChar).Value = nameBox.Text;

            adapter.SelectCommand = command;
            adapter.Fill(table);
```

```
            if (table.Rows.Count > 0)
            {
                MessageBox.Show("Оберіть інше ім'я, це ім'я вже використовується");
                return true;
            }
            else
                return false;
        }
    }
}
```

Метод checkname() повертає значення типу, яке вказує, чи ім'я вже використовується.

У методі створюється об'єкт класу OR, який використовується для взаємодії з базою даних MySQL. Створюється порожня таблиця table типу DataTable.

Створюється об'єкт adapter класу MySqlDataAdapter, який використовується для виконання SQL-запитів до бази даних.

Створюється команда command типу MySqlCommand, яка містить SQL-запит для вибірки даних з таблиці "object measurement values", де значення стовпця "name" дорівнює значенню поля nameBox.Text. Параметр команди @nO встановлюється зі значенням nameBox.Text.

adapter.SelectCommand встановлюється як команда command.

Виконується заповнення таблиці table даними з бази даних за допомогою adapter.Fill(table).

Після цього перевіряється, чи кількість рядків у таблиці table більше 0. Якщо так, виводиться повідомлення про помилку "Оберіть інше ім'я, це ім'я вже використовується" за допомогою MessageBox.Show() і метод повертає true, що вказує на те, що ім'я вже використовується.

У протилежному випадку метод повертає false, що вказує на те, що ім'я не використовується і можна продовжувати виконання інших дій.

3.4. Опис роботи Form3

Після того, як користувач вибере об’єкт та натисне на кнопку «Видалити об’єкт», з’явиться вікно Form3, яке запросить у користувача підтвердження бажання видалити об’єкт. У разі підтвердження, починається процес видалення об’єкту з БД. Form3 не відповідає за видалення об’єктів з внутрішнього архіву, відповідних до об’єкту змінних або лічильників, для цього був створений метод UpdateTable() в Form1, який здійснюється при запуску програми, закриті Form2 та Form3, тобто при спробі додати чи видалити об’єкт.

```
using MySql.Data.MySqlClient;
using System;
using System.Windows.Forms;

namespace Diplom1
{
    public partial class Form3 : Form
    {
        private string selectedObjectId;

        public Form3(string objectId)
        {
            InitializeComponent();
            selectedObjectId = objectId;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            OR or = new OR();

            try
            {
                // Відкрити з'єднання
                or.openConnection();

                // Створити команду для видалення об'єкта за обраним
ідентифікатором
                MySqlCommand command = new MySqlCommand("DELETE FROM `object
measurement values` WHERE `id` = @id;", or.getConnection());
                command.Parameters.Add("@id", MySqlDbType.VarChar).Value =
selectedObjectId;

                // Виконати команду видалення
                int rowsAffected = command.ExecuteNonQuery();
```

```
                if (rowsAffected > 0)
                {
                    MessageBox.Show("Об'єкт видалено");
                    this.Close();
                }
                else
                {
                    MessageBox.Show("Помилка видалення об'єкта");
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка підключення до бази даних: " +
ex.Message);
            }
            finally
            {
                // Закрити з'єднання
                or.closeConnection();
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

Використовуються простори імен MySql.Data.MySqlClient, System та System.Windows.Forms.

Клас має приватне поле selectedObjectId типу string, яке зберігає обраний ідентифікатор об'єкта.

Конструктор класу Form3 приймає параметр objectId типу string і встановлює значення selectedObjectId з переданого значення. Крім цього, виконується ініціалізація компонентів форми за допомогою методу InitializeComponent().

Метод button1_Click обробляє подію натискання на кнопку button1. У методі створюється об'єкт класу OR, який використовується для роботи з базою

даних MySQL. В блоку try відкривається з'єднання з базою даних за допомогою or.openConnection().

Створюється об'єкт command класу MySqlCommand, який містить SQL-запит для видалення об'єкта з таблиці "object measurement values" за ідентифікатором, переданим у параметрах команди. Параметр команди @id встановлюється зі значенням selectedObjectId.

Виконується команда видалення за допомогою command.ExecuteNonQuery(), і результат виконання зберігається у змінній rowsAffected. Якщо кількість задіяних рядків більше 0, виводиться повідомлення "Об'єкт видалено", і форма закривається. В іншому випадку виводиться повідомлення "Помилка видалення об'єкта".

У блоку catch перехоплюється виняток типу Exception і виводиться повідомлення про помилку підключення до бази даних разом з повідомленням про саму помилку (ex.Message).

У блоку finally закривається з'єднання з базою даних за допомогою or.closeConnection().

Метод button2_Click обробляє подію натискання на кнопку button2. У цьому методі просто закривається поточна форма.

ВИСНОВКИ

Дипломна робота присвячена створенню додатку, здатного запам’ятовувати велику кількість об’єктів спостереження та їх характеристики, використовуючи БД, з метою підвищення ефективності процесів збору інформації з віддалених технологічних об’єктів спостереження

Проаналізовані можливості мови програмування C#, його синтаксис та команди. Обґрунтовано вибір методів, завдяки яким вона здатна виконувати завдання з автоматизованого моніторингу та графічного відображення поточного стану об’єктів спостереження. Були проаналізовані можливості та синтаксис Sql, його можливості взаємодії з мовою C#, проаналізовані його сильні та слабкі сторони, та розглянуті різні методи реалізації баз даних в системі моніторингу, серед яких була обрана найбільш ефективна.

Враховуючи всі зроблені висновки, на базі Visual Studio 2022 та бази даних Sql було розроблено автоматизовану систему моніторингу за технологічними об’єктами. Спроектована система здатна проводити аналіз інформації з різних хмарних сервісів з різних джерел з різним форматуванням та часовими характеристиками. Все це дозволяє підвищити ефективність процесу збору інформації а також подальший її аналіз. Результати дипломної роботи можуть бути впроваджені в системах збору інформації з різноманітних технологічних об’єктів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Заікіна Т. В. Дослідження та розробка системи контролю вводу інформації при формуванні баз даних інформаційних систем / Т. В. Заікіна; наук. кер. В. Б. Дроменко // Наукові розробки молоді на сучасному етапі : тези доповідей XVII Всеукраїнської наукової конференції молодих вчених та студентів (26-27 квітня 2018 р., Київ). - Київ : КНУТД, 2018. - Т. 2 : Мехатронні системи і комп'ютерні технології. Ресурсозбереження та охорона навколишнього середовища. - С. 273-274.
2. Карпенко К. О. Програмне забезпечення для дослідницької лабораторії з функцією автоматизованого пошуку наукових заходів / К. О. Карпенко; наук. кер. В. Г. Резанова // Наукові розробки молоді на сучасному етапі : тези доповідей XVII Всеукраїнської наукової конференції молодих вчених та студентів (26-27 квітня 2018 р., Київ). - Київ : КНУТД, 2018. - Т. 2 : Мехатронні системи і комп'ютерні технології. Ресурсозбереження та охорона навколишнього середовища. - С. 185-186.
3. Резанова В. Г. Розробка програмного забезпечення для малого автопідприємства / В. Г. Резанова, С. Ю. Бартницький // Інформаційні технології в науці, виробництві та підприємництві : збірник наукових праць молодих вчених, аспірантів, магістрів кафедри комп’ютерних наук та технологій / за заг. наук. ред. В. Ю. Щербаня. – Київ : Освіта України ; ФОП Маслаков, 2020. – С. 178-181.
4. Резанова В. Г. Дослідження та розробка графічних програмних засобів для інтерактивного планування експерименту / В. Г. Резанова. С. І. Куценко. О. В. Яблоков // Мехатронні системи: інновації та інжиніринг : тези доповідей IV Міжнародної науково-практичної конференції, м. Київ, 22 жовтня 2020 р. / відп. за вип. Г. І. Хімічева, В. М. Дворжак. – Київ : КНУТД, 2020. – С. 135-136.

5. Троелсен Э. Мова Програмування C# 2010 і платформа .NET 4.0, 5-е изд. / Э Троелсен. – М.: ООО “И.Д. Вильямс”, 2011. – 1392 с.
6. Шилдт Г. C# 4.0: полное руководство / Г. Шилдт. – М.: ООО “И.Д. Вильямс”, 2011. – 1056 с
7. Entity Framework [Електронний ресурс] // Документація Microsoft. — 2020. — Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Entity_Framework .
8. Моделювання систем: навч. посіб. [Електронний ресурс, текст] / І.В.Стеценко ; М-во освіти і науки України, Черкас. держ. технол. ун-т. — Черкаси : ЧДТУ, 2010. — 399 с.
9. Брила А.Ю., Антосяк П.П., Глебена М.І., Чупов С.В., Семйон І.В. Основи програмування у C#. Методичні вказівки до лабораторних робіт для студентів І-го курсу математичного факультету спеціальності "Прикладна математика". – Ужгород, 2014. – 60с.
10. C# 7.0 in a Nutshell: The Definitive Reference // Книга — Джозеф Албані — 2021 — 1007 с.
11. Pro ASP.NET MVC 3 Framework Languages // Книга — Адам Фрімен — 124 с.
12. Наукометричні бази даних [Електронний ресурс] // Національна бібліотека України. — 2014. — Режим доступу до ресурсу: http://www.nbuv.gov.ua/node/1367 .
13. The Definitive ANTLR Reference: Building Domain-Specific Languages //Книга — Терренс Парк — 334 с.
14. Руководство по ASP.NET MVC 5 [Електронний ресурс] // Посібник — 2017. — Режим доступу до ресурсу: https://metanit.com/sharp/mvc5/ .
15. SQL-injections: vulnerabilities and how to prevent attacks [Електронний ресурс]. - Режим доступу URL: https://www.veracode.com/security/sql-injection

16. Libinjection [Електронний ресурс]. - Режим доступу URL: https://github.com/client9/libinjection
17. Cloud Web Application Firewall [Електронний ресурс]. - Режим доступу URL: https://www.cloudflare.com/waf/
18. Best Practices to prevent SQL-injections [Електронний ресурс]. – Режим доступу URL: https://tableplus.io/blog/2018/08/best-practices-to-prevent-sqlinjection-attacks.html
19. SQL Injections Top Attack Statistics [Електронний ресурс]. - Режим доступу URL: https://www.darkreading.com/risk/sql-injections-top-attack-statistics/d/did/1132988
20. AWS Documantation [Електронний ресурс]. - Режим доступу URL: https://docs.aws.amazon.com/index.html?nc2=h_ql_doc
21. Free CDN to Speed Up and Secure WebSite [Електронний ресурс]. – Режим доступу URL: https://geekflare.com/free-cdn-list/
22. OWASP CheatSheetSeries [Електронний ресурс]. - Режим доступу URL:https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.md
23. Akamai Report. The State of the Internet [Електронний ресурс]. – Режим доступу URL: https://www.akamai.com/us/en/resources/our-thinking/state-ofthe-internet-report/global-state-of-the-internet-security-ddos-attack-reports.jsp
24. Балик Н.Р., Мандзюк В.І. Бази даних MySQL: Навчальний посібник. – Тернопіль: Навчальна книга – Богдан, 2010. – 160 с.
25. Атре Ш. Структурний підхід до організації баз даних. - М.: Фінанси і статистика, 1983. - 320 с.
26. Лебеденко Ю.О. Автоматизована система віддаленого моніторінгу стану дощувальних машин / Ю.О. Лебеденко, А.А. Омельчук, О.В. Поливода // Прикладні питання математичного моделювання, 2019 №1, т 2, С. 89-97.

27. Божок Д.С. Система тестовой диагностики цифровых устройств систем управления / Д.С. Божок, Н.Н. Вакаров, Ю.О. Лебеденко // VI Всеукраїнська науково-практичної конференції студентів, аспірантів та молодих вчених з автоматичного управління присвяченої дню космонавтики, 12 квітня 2018 р., м. Херсон
28. Вакаров М.М. Методи і засоби технічної діагностики систем управління / Ю.О. Лебеденко, М.М. Вакаров, В.Є. Крайнов // V Всеукраїнська науково-практичної конференції студентів, аспірантів та молодих вчених з автоматичного управління присвяченої дню космонавтики, 12 квітня 2017 р., м. Херсон, С. 27-30

Додаток А

Клас для підключення до бази даних

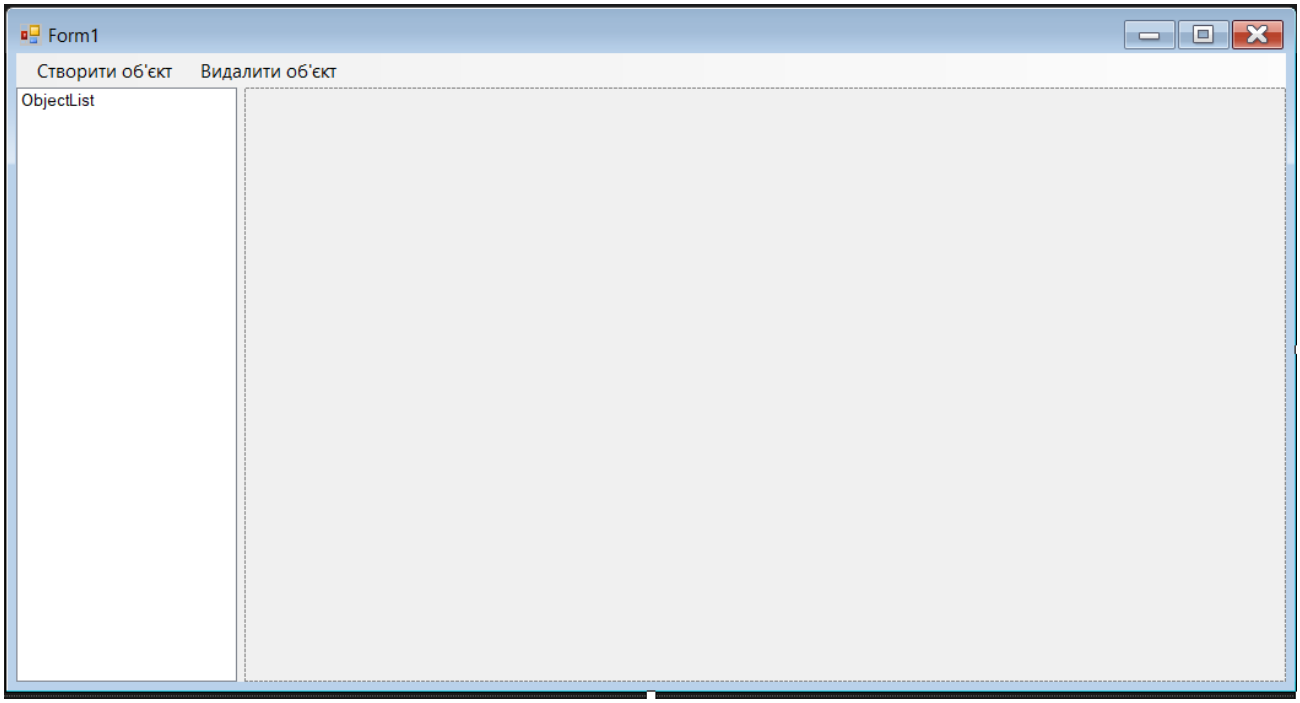
Клас OR

```
using MySql.Data.MySqlClient;

namespace Diplom1
{
    internal class OR
    {
        MySqlConnection connection = new
MySqlConnection("server=localhost;port=3306;username=root;password=root;database=ob
ject_val");

        public void openConnection()
        {
            if (connection.State == System.Data.ConnectionState.Closed)
                connection.Open();
        }

        public void closeConnection()
        {
            if (connection.State == System.Data.ConnectionState.Open)
                connection.Close();
        }

        public MySqlConnection getConnection()
        {
            return connection;
        }
    }
}
```

Додаток Б

Код відображення поточного стану об’єктів спостереження

Form1




```
using MySql.Data.MySqlClient;
using System;
using System.Collections.Generic;
using System.Data;
using System.Threading;
using System.Windows.Forms;

namespace Diplom1
{
    public partial class Form1 : Form
    {
        private bool isFormLoaded = false;

        public Form1()
        {
            InitializeComponent();
            Shown += Form1_Shown;
        }

        static double temp_generator()
        {
```

```
            double variable = 65.0; // Початкове значення змінної
            Random random = new Random();

            while (true)
            {
                int number = random.Next(0, 1001); // Генеруємо випадкове число в
діапазоні від 0 до 1000

                // Змінюємо значення змінної згідно з умовами
                if (number >= 0 && number <= 400)
                {
                    variable += 0.3;
                }
                else if (number >= 401 && number <= 800)
                {
                    variable -= 0.3;
                }
                else if (number >= 801 && number <= 900)
                {
                    variable += 1;
                }
                else if (number >= 901 && number <= 1000)
                {
                    variable -= 1;
                }

                // Перевіряємо обмеження на мінімальне та максимальне значення
змінної
                if (variable < 60)
                {
                    variable += 3;
                }
                else if (variable > 75)
                {
                    variable -= 3;
                }

                // Робимо паузу в 1 секунду
                Thread.Sleep(1000);
            }
        }

        static double press_generator()
        {
            double variable = 250.0; // Початкове значення змінної
            Random random = new Random();
```

```
            while (true)
            {
                int number = random.Next(0, 1001); // Генеруємо випадкове число в
діапазоні від 0 до 1000

                // Змінюємо значення змінної згідно з умовами
                if (number >= 0 && number <= 400)
                {
                    variable += 2.3;
                }
                else if (number >= 401 && number <= 800)
                {
                    variable -= 2.3;
                }
                else if (number >= 801 && number <= 900)
                {
                    variable += 7.6;
                }
                else if (number >= 901 && number <= 1000)
                {
                    variable -= 7.6;
                }

                // Перевіряємо обмеження на мінімальне та максимальне значення
змінної
                if (variable < 101)
                {
                    variable += 15;
                }
                else if (variable > 405)
                {
                    variable -= 15;
                }

                // Робимо паузу в 1 секунду
                Thread.Sleep(1000);
            }
        }

        static double power_generator()
        {
            double variable = 150.0; // Початкове значення змінної
            Random random = new Random();

            while (true)
            {
```

```
                int number = random.Next(0, 1001); // Генеруємо випадкове число в
діапазоні від 0 до 1000

                // Змінюємо значення змінної згідно з умовами
                if (number >= 0 && number <= 400)
                {
                    variable += 2.3;
                }
                else if (number >= 401 && number <= 800)
                {
                    variable -= 2.3;
                }
                else if (number >= 801 && number <= 900)
                {
                    variable += 7.6;
                }
                else if (number >= 901 && number <= 1000)
                {
                    variable -= 7.6;
                }

                // Перевіряємо обмеження на мінімальне та максимальне значення
змінної
                if (variable < 120)
                {
                    variable += 10;
                }
                else if (variable > 170)
                {
                    variable -= 10;
                }

                // Робимо паузу в 1 секунду
                Thread.Sleep(1000);
            }
        }

        static double fuel_generator()
        {
            double variable = 80.0; // Початкове значення змінної
            Random random = new Random();

            while (true)
            {
                int number = random.Next(0, 501); // Генеруємо випадкове число в
діапазоні від 0 до 1000
```

```
                // Змінюємо значення змінної згідно з умовами
                if (number >= 0 && number <= 200)
                {
                    variable -= 1;
                }
                else if (number >= 201 && number <= 400)
                {
                    variable -= 2;
                }
                else if (number >= 401 && number <= 500)
                {
                    variable -= 4.5;
                }

                // Перевіряємо обмеження на мінімальне та максимальне значення
змінної
                if (variable < 65)
                {
                    variable = 100;
                }

                // Робимо паузу в 1 секунду
                Thread.Sleep(1000);
            }
        }

        private void створитиToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Form2 form2 = new Form2();
            form2.FormClosed += Form2_FormClosed;
            form2.ShowDialog();
        }

        private void Form2_FormClosed(object sender, FormClosedEventArgs e)
        {
            MessageBox.Show("Вікно Form2 закрите!");
            UpdateTable();
        }

        private void Form3_FormClosed(object sender, FormClosedEventArgs e)
        {
            MessageBox.Show("Вікно Form3 закрите!");
            UpdateTable();
        }

        private Dictionary<string, double> temperatureVariables = new
Dictionary<string, double>();
```

```
        private Dictionary<string, double> pressureVariables = new
Dictionary<string, double>();
        private Dictionary<string, double> powerVariables = new Dictionary<string,
double>();
        private Dictionary<string, double> fuelVariables = new Dictionary<string,
double>();
        private Dictionary<string, int> counters = new Dictionary<string, int>();
        private List<ProgressBar> progressBars = new List<ProgressBar>();
        private List<Label> labels = new List<Label>();

        private void Form1_Shown(object sender, EventArgs e)
        {
            // Створюємо таблицю, заповнюємо змінні та лічильники
            CreateTable();
            FillVariablesAndCounters();
            UpdateTable();
        }

        private void CreateTable()
        {
            string sourceConnectionString =
"server=localhost;port=3306;username=root;password=root;database=object_val";
            string destinationTableName = "object measurement values";
            DataTable newTable = new DataTable();

            using (MySqlConnection sourceConnection = new
MySqlConnection(sourceConnectionString))
            {
                sourceConnection.Open();

                // Отримуємо структуру таблиці з бази даних
                string schemaQuery = $"SELECT * FROM {destinationTableName} LIMIT
1";
                using (MySqlDataAdapter schemaAdapter = new
MySqlDataAdapter(schemaQuery, sourceConnection))
                {
                    schemaAdapter.FillSchema(newTable, SchemaType.Source);
                }

                // Створюємо нову таблицю в додатку
                newTable.TableName = "NewTable";

                // Заповнюємо нову таблицю даними з бази даних
                string dataQuery = $"SELECT * FROM {destinationTableName}";
                using (MySqlDataAdapter dataAdapter = new
MySqlDataAdapter(dataQuery, sourceConnection))
                {
```

```
                dataAdapter.Fill(newTable);
            }

            sourceConnection.Close();
        }

        // Заповнення ListBox значеннями з колонки "name"
        foreach (DataRow row in newTable.Rows)
        {
            string nameValue = row["name"].ToString();
            ObjectList.Items.Add(nameValue);

            string id = row["id"].ToString();

            if (row["temperature"].ToString() == "1")
            {
                double temperatureVariable = temp_generator();
                temperatureVariables.Add($"temp_{id}", temperatureVariable);
                counters.Add($"counter_{id}", 0);
            }

            if (row["pressure"].ToString() == "1")
            {
                double pressureVariable = press_generator();
                pressureVariables.Add($"press_{id}", pressureVariable);
                counters.Add($"counter_{id}", 0);
            }

            if (row["power"].ToString() == "1")
            {
                double powerVariable = power_generator();
                powerVariables.Add($"power_{id}", powerVariable);
                counters.Add($"counter_{id}", 0);
            }

            if (row["fuel"].ToString() == "1")
            {
                double fuelVariable = fuel_generator();
                fuelVariables.Add($"fuel_{id}", fuelVariable);
                counters.Add($"counter_{id}", 0);
            }
        }
    }

    private void FillVariablesAndCounters()
    {
        ObjectList.Items.Clear();
```

```
            foreach (var kvp in counters)
            {
                string id = kvp.Key.Substring(8);
                int counter = kvp.Value;

                string name = GetObjectNameById(id);
                ObjectList.Items.Add(name);

                counters[kvp.Key] = counter;
            }
        }

        private string GetObjectNameById(string id)
        {
            string sourceConnectionString =
"server=localhost;port=3306;username=root;password=root;database=object_val";
            string tableName = "object measurement values";
            string name = "";

            using (MySqlConnection sourceConnection = new
MySqlConnection(sourceConnectionString))
            {
                sourceConnection.Open();

                string query = $"SELECT name FROM {tableName} WHERE id = {id}";
                using (MySqlCommand command = new MySqlCommand(query,
sourceConnection))
                {
                    using (MySqlDataReader reader = command.ExecuteReader())
                    {
                        if (reader.Read())
                        {
                            name = reader["name"].ToString();
                        }
                    }
                }

                sourceConnection.Close();
            }

            return name;
        }

        private void ObjectList_SelectedIndexChanged(object sender, EventArgs e)
        {
            // Видалення попередніх ProgressBar та міток
```

```
            RemoveExistingProgressBarsAndLabels();

            // Отримання вибраного об'єкта зі списку
            string selectedObjectName = ObjectList.SelectedItem.ToString();

            // Отримання відповідних змінних та лічильників для вибраного об'єкта
            List<double> selectedTemperatureVariables = new List<double>();
            List<double> selectedPressureVariables = new List<double>();
            List<double> selectedPowerVariables = new List<double>();
            List<double> selectedFuelVariables = new List<double>();
            int counterValue = 0;

            foreach (var kvp in temperatureVariables)
            {
                if (kvp.Key.StartsWith($"temp_{selectedObjectName}"))
                {
                    selectedTemperatureVariables.Add(kvp.Value);
                    counterValue++;
                }
            }

            foreach (var kvp in pressureVariables)
            {
                if (kvp.Key.StartsWith($"press_{selectedObjectName}"))
                {
                    selectedPressureVariables.Add(kvp.Value);
                    counterValue++;
                }
            }

            foreach (var kvp in powerVariables)
            {
                if (kvp.Key.StartsWith($"power_{selectedObjectName}"))
                {
                    selectedPowerVariables.Add(kvp.Value);
                    counterValue++;
                }
            }

            foreach (var kvp in fuelVariables)
            {
                if (kvp.Key.StartsWith($"fuel_{selectedObjectName}"))
                {
                    selectedFuelVariables.Add(kvp.Value);
                    counterValue++;
                }
            }
```

```
            // Створення ProgressBar та міток залежно від значення лічильника
            for (int i = 0; i < counterValue; i++)
            {
                ProgressBar progressBar = new ProgressBar();
                progressBar.Left = 250;
                progressBar.Top = 50 + (i * 100);
                progressBar.Width = 200;
                progressBar.Visible = false;

                Label label = new Label();
                label.Left = progressBar.Left + progressBar.Width + 10;
                label.Top = progressBar.Top + (progressBar.Height / 2) - 10;
                label.AutoSize = true;

                string labelSuffix = GetLabelSuffix(i);
                label.Text = labelSuffix;

                panel1.Controls.Add(progressBar);
                panel1.Controls.Add(label);

                progressBars.Add(progressBar);
                labels.Add(label);
            }

            // Оновлення значень ProgressBar
            UpdateProgressBarValues(selectedTemperatureVariables,
selectedPressureVariables, selectedPowerVariables, selectedFuelVariables);
        }

        private string GetLabelSuffix(int index)
        {
            string labelSuffix = "";

            if (index % 4 == 0)
            {
                labelSuffix = "°C";
            }
            else if (index % 4 == 1)
            {
                labelSuffix = "кПа";
            }
            else if (index % 4 == 2)
            {
                labelSuffix = "кВт";
            }
            else if (index % 4 == 3)
```

```
            {
                labelSuffix = "% палива";
            }

            return labelSuffix;
        }


        private void RemoveExistingProgressBarsAndLabels()
        {
            // Видаляємо попередні ProgressBar та мітки з панелі
            foreach (ProgressBar progressBar in progressBars)
            {
                panel1.Controls.Remove(progressBar);
                progressBar.Dispose();
            }

            foreach (Label label in labels)
            {
                panel1.Controls.Remove(label);
                label.Dispose();
            }

            // Очищаємо списки ProgressBar та міток
            progressBars.Clear();
            labels.Clear();
        }

        private void UpdateProgressBarValues(List<double> temperatureValues,
List<double> pressureValues, List<double> powerValues, List<double> fuelValues)
        {
            // Оновлення значень ProgressBar
            int progressBarIndex = 0;

            foreach (double temperatureValue in temperatureValues)
            {
                ProgressBar progressBar = progressBars[progressBarIndex];
                int progressPercentage = Convert.ToInt32(temperatureValue * 100);
                progressBar.Value = progressPercentage;
                progressBar.Visible = true;

                progressBarIndex++;
            }

            foreach (double pressureValue in pressureValues)
            {
                ProgressBar progressBar = progressBars[progressBarIndex];
```

```
                int progressPercentage = Convert.ToInt32(pressureValue * 100);
                progressBar.Value = progressPercentage;
                progressBar.Visible = true;

                progressBarIndex++;
            }

            foreach (double powerValue in powerValues)
            {
                ProgressBar progressBar = progressBars[progressBarIndex];
                int progressPercentage = Convert.ToInt32(powerValue * 100);
                progressBar.Value = progressPercentage;
                progressBar.Visible = true;

                progressBarIndex++;
            }

            foreach (double fuelValue in fuelValues)
            {
                ProgressBar progressBar = progressBars[progressBarIndex];
                int progressPercentage = Convert.ToInt32(fuelValue * 100);
                progressBar.Value = progressPercentage;
                progressBar.Visible = true;

                progressBarIndex++;
            }

            // Видаляємо зайві ProgressBar та мітки, якщо їх кількість менша за
поточне значення індексу ProgressBar
            if (progressBars.Count > progressBarIndex)
            {
                for (int i = progressBarIndex; i < progressBars.Count; i++)
                {
                    ProgressBar progressBar = progressBars[i];
                    panel1.Controls.Remove(progressBar);
                    progressBar.Dispose();
                }

                progressBars.RemoveRange(progressBarIndex, progressBars.Count -
progressBarIndex);
            }

            if (labels.Count > progressBarIndex)
            {
                for (int i = progressBarIndex; i < labels.Count; i++)
                {
                    Label label = labels[i];
```

```
                    panel1.Controls.Remove(label);
                    label.Dispose();
                }

                labels.RemoveRange(progressBarIndex, labels.Count -
progressBarIndex);
            }
        }

        private void UpdateTable()
        {
            string sourceConnectionString =
"server=localhost;port=3306;username=root;password=root;database=object_val";
            string destinationTableName = "object measurement values";
            DataTable newTable = new DataTable();

            using (MySqlConnection sourceConnection = new
MySqlConnection(sourceConnectionString))
            {
                sourceConnection.Open();

                // Отримуємо структуру таблиці з бази даних
                string schemaQuery = $"SELECT * FROM {destinationTableName} LIMIT
1";
                using (MySqlDataAdapter schemaAdapter = new
MySqlDataAdapter(schemaQuery, sourceConnection))
                {
                    schemaAdapter.FillSchema(newTable, SchemaType.Source);
                }

                // Створюємо нову таблицю в додатку
                newTable.TableName = "NewTable";

                // Заповнюємо нову таблицю даними з бази даних
                string dataQuery = $"SELECT * FROM {destinationTableName}";
                using (MySqlDataAdapter dataAdapter = new
MySqlDataAdapter(dataQuery, sourceConnection))
                {
                    dataAdapter.Fill(newTable);
                }

                sourceConnection.Close();
            }

            // Видаляємо попередні дані зі списків та словників
            temperatureVariables.Clear();
            pressureVariables.Clear();
```

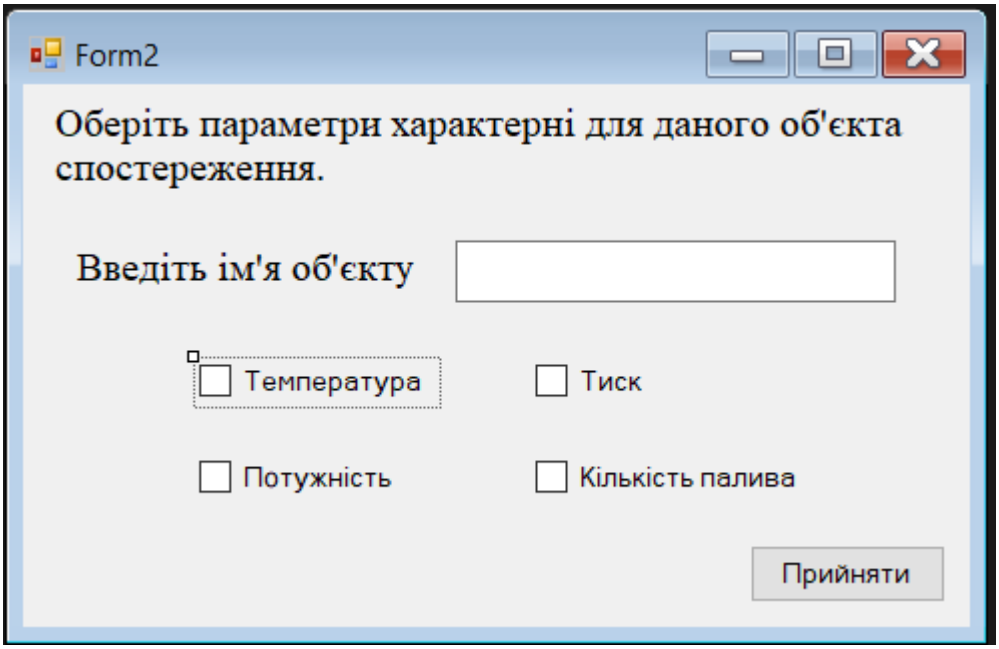
```
            powerVariables.Clear();
            fuelVariables.Clear();
            counters.Clear();

            // Очищуємо ListBox та видаляємо всі ProgressBar та мітки з панелі
            ObjectList.Items.Clear();
            RemoveExistingProgressBarsAndLabels();

            // Заповнення ListBox значеннями з колонки "name" та додавання змінних
та лічильників
            foreach (DataRow row in newTable.Rows)
            {
                string nameValue = row["name"].ToString();
                ObjectList.Items.Add(nameValue);

                string id = row["id"].ToString();

                if (row["temperature"].ToString() == "1")
                {
                    double temperatureVariable = temp_generator();
                    temperatureVariables.Add($"temp_{id}", temperatureVariable);
                    counters.Add($"counter_{id}", 0);
                }

                if (row["pressure"].ToString() == "1")
                {
                    double pressureVariable = press_generator();
                    pressureVariables.Add($"press_{id}", pressureVariable);
                    counters.Add($"counter_{id}", 0);
                }

                if (row["power"].ToString() == "1")
                {
                    double powerVariable = power_generator();
                    powerVariables.Add($"power_{id}", powerVariable);
                    counters.Add($"counter_{id}", 0);
                }

                if (row["fuel"].ToString() == "1")
                {
                    double fuelVariable = fuel_generator();
                    fuelVariables.Add($"fuel_{id}", fuelVariable);
                    counters.Add($"counter_{id}", 0);
                }
            }
        }
```

```
        private string selectedObjectId;
        private void видалитиToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Form3 form3 = new Form3(selectedObjectId);
            form3.FormClosed += Form2_FormClosed;
            form3.ShowDialog();
        }
    }
}
```

Додаток В

Код для додавання та контролю додавання об’єкту спостереження

Form2




```
using MySql.Data.MySqlClient;
using System;
using System.Data;
using System.Windows.Forms;

namespace Diplom1
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }

        private void ok_button_Click(object sender, EventArgs e)
        {
            if (nameBox.Text == "")
            {
                MessageBox.Show("Введіть ім'я для об'єкту"); // Повідомлення про
помилку, якщо поле для імені порожнє
                return;
```

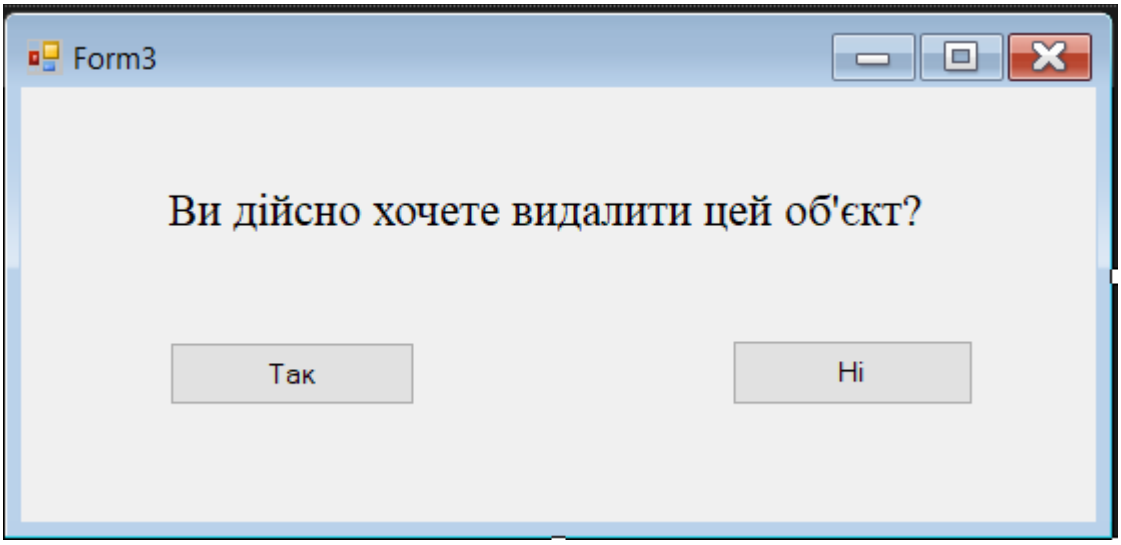
```
            }

            if (!temCheckBox.Checked && !pressCheckBox.Checked &&
!powerCheckBox.Checked && !fuelCheckBox.Checked)
            {
                MessageBox.Show("Оберіть, які дані повинні відстежуватися в цьому
об'єкті"); // Повідомлення про помилку, якщо жодне поле не вибране для
відстежування
                return;
            }

            if (checkname())
                return;

            OR or = new OR();
            MySqlCommand command = new MySqlCommand("INSERT INTO `object
measurement values` ( `name`, `temperature`, `pressure`, `power`, `fuel`) VALUES
(@name, @temperature, @pressure, @power, @fuel);", or.getConnection());

            command.Parameters.Add("@name", MySqlDbType.VarChar).Value =
nameBox.Text;
            command.Parameters.Add("@temperature", MySqlDbType.Int32).Value =
temCheckBox.Checked;
            command.Parameters.Add("@pressure", MySqlDbType.Int32).Value =
pressCheckBox.Checked;
            command.Parameters.Add("@power", MySqlDbType.Int32).Value =
powerCheckBox.Checked;
            command.Parameters.Add("@fuel", MySqlDbType.Int32).Value =
fuelCheckBox.Checked;

            or.openConnection();

            if (command.ExecuteNonQuery() == 1)
            {
                MessageBox.Show("Об'єкт створено"); // Повідомлення про успішне
створення об'єкта
                this.Close();
            }
            else
            {
                MessageBox.Show("Об'єкт не створено"); // Повідомлення про невдале
створення об'єкта
            }
            or.closeConnection();
        }
        public Boolean checkname()// Повідомлення про помилку, якщо ім'я вже
використовується
```

```
        {
            OR or = new OR();

            DataTable table = new DataTable();

            MySqlDataAdapter adapter = new MySqlDataAdapter();

            MySqlCommand command = new MySqlCommand("SELECT * FROM 'object
measurement values' WHERE 'name'=@nO");

            command.Parameters.Add("nO", MySqlDbType.VarChar).Value = nameBox.Text;

            adapter.SelectCommand = command;
            adapter.Fill(table);

            if (table.Rows.Count > 0)
            {
                MessageBox.Show("Оберіть інше ім'я, це ім'я вже використовується");
                return true;
            }
            else
                return false;
        }
    }
}
```

Додаток Г

Код для підтвердження видалення об’єкту спостереження

Form3




```
using MySql.Data.MySqlClient;
using System;
using System.Windows.Forms;

namespace Diplom1
{
    public partial class Form3 : Form
    {
        private string selectedObjectId;

        public Form3(string objectId)
        {
            InitializeComponent();
            selectedObjectId = objectId;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            OR or = new OR();

            try
            {
                // Відкрити з'єднання
                or.openConnection();

                // Створити команду для видалення об'єкта за обраним
ідентифікатором
```

```
                MySqlCommand command = new MySqlCommand("DELETE FROM `object
measurement values` WHERE `id` = @id;", or.getConnection());
                command.Parameters.Add("@id", MySqlDbType.VarChar).Value =
selectedObjectId;

                // Виконати команду видалення
                int rowsAffected = command.ExecuteNonQuery();

                if (rowsAffected > 0)
                {
                    MessageBox.Show("Об'єкт видалено");
                    this.Close();
                }
                else
                {
                    MessageBox.Show("Помилка видалення об'єкта");
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка підключення до бази даних: " +
ex.Message);
            }
            finally
            {
                // Закрити з'єднання
                or.closeConnection();
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

Додаток Д

Тези конференції

X Всеукраїнська науково-практична конференція з автоматичного управління

УДК 004.22

Ю.О. Лебеденко, В.В. Стручок, В.В. Голінко
Київський національний університет технологій та дизайну
lebedenko.yo@knutd.edu.ua

АВТОМАТИЗОВАНА СИСТЕМА МОНІТОРИНГУ ПАРАМЕТРІВ ОБЛАДНАННЯ РОЗПОДІЛЕНИХ СИСТЕМ

В сучасному світі питання автономності та незалежності у сфері електропостачання, теплопостачання, водопостачання або інтернету стає дедалі популярнішим та перспективнішим [1]. Це пов'язано зі зростаючою свідомістю людей щодо необхідності екологічної сталості та забезпечення своєї безпеки та комфорту у повсякденному житті. А згадуючи про ситуацію в нашій країні, ця тема набуває актуальності та потребує комплексного підходу та вдосконалення відповідних систем та технологій з метою забезпечення максимальної надійності та ефективності їх роботи.

Саме тому для цього використовують найсучасніші технологічні прилади, які мають достатній рівень самостійності та потребують лише мінімальних втручань з боку людини. Зазвичай, для більш зручного використання цих технологічних приладів та комфортного стеження за ними, їх конструкція може припускати можливість передачі даних свого поточного стану через «хмару» у додаток на будь-якому електронному пристрої, наприклад телефоні або комп'ютері [2, 3]. Але у випадках використання технологічних приладів від різних фірм часто виникає проблема несумісності, через що за кожним приладом доводиться стежити у різних додатках [4]. Це незручно і іноді може призвести до неприємних наслідків або навіть катастроф.

Через це виникає потреба у створенні єдиного інтерфейсу, який міг би сприймати інформацію з «хмари» незалежно від фірми-виробника та відображати дані усіх технологічних приладів у будівлі, або у кількох будівлях.

Об'єктом дослідження є хмарні сервіси моніторингу стану віддалених технологічних об'єктів. Предметом дослідження є підходи до підвищення ефективності процесу збору інформації з різних хмарних сервісів. Завданням дослідження є оптимізація процесу збору та первинної обробки інформації в системах моніторингу.

Для виконання поставленого завдання, передбачається створення окремого програмного засобу, який би виконував роль системи моніторингу. За приклад ми можемо взяти аналогічну програму Grafana, яка може вести зчитувати, аналізування та порівняння даних з опалювальних котлів [5]. Приклад інтерфейсу програмної системи Grafana наведено на рис. 1.

Рисунок 1 – Інтерфейс програмної системи візуальних даних Grafana

Ця програма написана на мовах програмування Go та TypeScript, тому вона здатна виконувати парсингу. Парсинг (від англ. parsing), або синтаксичний аналіз, – це процес аналізування інформації з метою визначення її структури та витягування корисної інформації [6]. Для вирішення нашої задачі програма повинна спочатку виконувати парсинг, завдяки якому інформація отримуватиме первинну обробку та буде виводитися у вигляді, зручному для користувача.

Парсинг є основною функцією програми та виконує головну роль у моніторингу за поточними даними з технологічного об'єкту, але наша програма передбачає спостереження за декількома об'єктами, які можуть взаємодіяти один з одним та не мають спільного потоку даних [7, 8]. До того ж, для створення більш широкого та докладного аналізу поточних даних та характеру їх змін, потрібно враховувати дані минулих станів технологічних об'єктів. Для збереження цієї інформації потрібно створити базу даних. Основою будь-якої бази даних є сервери, тому включення баз даних до нашої програми передбачає створення власної або використання пропонованої інфраструктури. Архітектура передачі даних до серверів та користувача наведено на рис. 2.

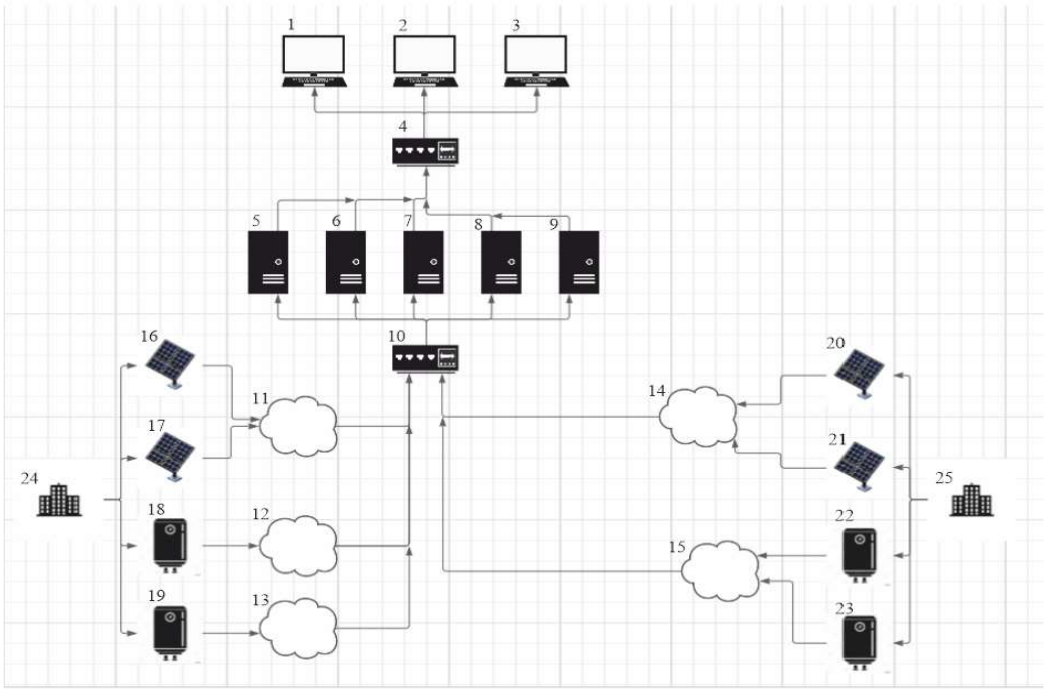


Рисунок 2 – Архітектура мережи передачі даних з технологічних приладів до користувача через різні хмарні сервіси:
1,2,3 – електронні пристрої користувача; 4,10 - балансувальники; 5,6,7,8,9 – сервери; 11,12,13,14,15 – хмарні сервіси, створені фірмами-виробниками технологічних приладів; 16,17,18,19,20,21,22,23 – технологічні прилади(16,17,20,21 – сонячні панелі, 18,19,22,23 - котли опалення, як приклад), 24,25 – будівлі, в яких встановлені технологічні прилади.

Для одночасної реалізації парсингу і роботи з базами даних в одній програмі, доцільно обрати мову програмування C# [9]. Для парсингу ця мова програмування має бібліотеки HtmlAgilityPack та AngleSharp. HtmlAgilityPack є бібліотекою для парсингу HTML-сторінок, яка дозволяє отримувати доступ до елементів сторінки та їх атрибутів. AngleSharp є бібліотекою, яка дозволяє парсити не тільки HTML-сторінки, але й XML-файли та інші

формати даних. Також мова C# має розширення LINQ, яке дозволяє легко обробляти та фільтрувати дані, що отримані з веб-сайту.

.NET Framework має багато вбудованих засобів для роботи з реляційними базами даних, такими як Microsoft SQL Server, Oracle, MySQL та інші [10]. Для цього в .NET Framework є технології ADO.NET та LINQ to SQL, які дозволяють зчитувати та зберігати дані в базі даних.

ADO.NET є технологією, яка дозволяє створювати з'єднання з базою даних, виконувати запити до бази даних та обробляти результати запитів. ADO.NET також містить багато інших функцій, таких як використання транзакцій, обробка помилок, оптимізація запитів та інші.

LINQ to SQL є технологією, яка дозволяє створювати об'єкти-сутності на основі таблиць бази даних та здійснювати з ними роботу на рівні об'єктно-орієнтованої програми.

Висновок. Застосування сучасних інформаційних технологій дозволяє здійснювати моніторинг параметрів розподілених об’єктів, що використовують різні хмарні сервіси. В якості подальшого дослідження, планується створення методів зворотного зв’язку програми та користувача з технологічними об’єктами, що зробить програму системою моніторингу та управління. Це дозволить програмі взаємодіяти з обладнанням у випадках його критичного перенавантаження або дасть можливість користувачу дистанційно керувати технологічним процесом.

ЛІТЕРАТУРА:

1. Енергетичні ресурси та потоки / Шидловський А.К., Віхорєв Ю.О., Гінайло В.О. та ін. К.: Українські енциклопедичні знання, 2003. 472 с.
2. Яковицький І.Л. Технологія «хмарних обчислень» як інструмент створення інформаційної інфраструктури управління. *Комунальне господарство міст*. №102. С. 320-327.
3. Cloud Automation: Why, Where and How. URL: https://bluexp.netapp.com/blog/cloud-automation-why-where-and-how-cvo-blg
4. Шишак А.В., Пупена О.М. На шляху до Індустрії 4.0: інтеграція існуючих АСУТП з хмарними сервісами. *Автоматизація технологічних і бізнес-процесів*, №10, Т. 1, 2018. С. 33-39.
5. Grafana Dashboard Monitoring - Store & Visualize Your Metrics. URL: https://newrelic.com/lp/grafana-monitoring?utm medium=cpc&utm source=google&utm campaign=EVER-GREEN NB SEARCH GRAFANA EMEA CENTRAL EN&utm network=g&utm keyword=grafana&utm device=c& bt=469549587759& bm=e& bn=g&gclid=CjwKCAjwoIqhBhAGEiwArXT7K2ewTFuTloWp-nt2ta7b2aW8DQKPaoXEX jZo AAMa3T 8wBoatc8xoCH2cQAvD BwE
6. Синтаксичний аналіз. URL: https://www.wiki-data.uk-ua.nina.az/Синтаксичний аналіз.html
7. Earley J. An efficient context-free parsing algorithm. *ACM* 13, 2 (Feb 1970). – P. 94–102.
8. Швороб І.Б. Порівняльний аналіз методів синтаксичного розбору текстів. *Вісник Національного університету “Львівська політехніка”. Збірник наукових праць. Серія: Інформаційні системи та мережі*. № 814, 2015. С. 197 -202.
9. The AZ of Programming Languages: C #. URL: https://www.computerworld.com/au/
10. Crane R., Resnick S., Bowen C. Essential Windows Communication Foundation (WCF): For .NET Framework 3.5. *Pearson Education*. 2008. 608 p.