

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ВІДСТЕЖЕННЯ ЗМІН
РАДІАЦІЙНОГО ФОНУ
SOFTWARE FOR TRACKING CHANGES IN RADIATION BACKGROUND

Виконав(ла): студент(ка) 2 курсу, групи КНТ-124м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

РОГОЖКІН Я.А.

(ПРІЗВИЩЕ та ініціали)

Керівник ШИТІКОВА О.В.

(ПРІЗВИЩЕ та ініціали)

Рецензент ЗЕЛІК О.В.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

РОГОЖКІНА Ярослава Андрійовича
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення для відстеження змін
радіаційного фону Software for Tracking Changes in Radiation Background

керівник проєкту (роботи) к.т.н., ШИТІКОВА Олена Вікторівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 30 ” вересня 2025 року № 447

2. Строк подання студентом проєкту (роботи) 08 грудня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Вибір та обґрунтування структури системи, яка проєктується. 3. Основні рішення щодо реалізації компонентів системи. 4. Імітація роботи радіаційного датчика. 5. Керівництво програміста. 6. Керівництво оператора. 7. Експерименти та результати.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-6 Основна частина	ШИТІКОВА О.В., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання « 30 » вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тиждень	Розділ 1
3	Вибір засобів розробки програмного продукту.	4-5 тиждень	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми	7-8 тижні	Розділ 4, 5
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 6, 7
7	Оформлення пояснювальної записки та документів до неї.	10-11 тиждень	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

Ярослав РОГОЖКІН
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

Олена ШИТІКОВА
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
193 с., 10 табл., 51 рис., 3 дод., 31 джерела.

РАДІАЦІЙНИЙ ФОН, ЗАБРУДНЕННЯ, ВИМІРЮВАННЯ,
РАДІАЦІЯ, C#.

Об'єкт дослідження – процес інженерії програмного забезпечення для моніторингу радіаційного фону.

Предмет дослідження – методи, алгоритми та програмні засоби вимірювання, аналізу та оперативного контролю змін радіаційного фону.

Мета роботи – збільшення точності збору та аналізу даних вимірювання радіаційного випромінювання та швидкості оперативного інформування користувача про його небезпечні зміни.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мова програмування C#, середовище розробки Visual Studio 2022, персональний комп'ютер з процесором Intel Core i5 під управлінням операційної системи Microsoft Windows 10.

Результати. Проведено аналіз предметної області та створено програмний Windows-застосунок, який здійснює моніторинг рівня радіаційного фону, реєструє зміни показників у реальному часі та забезпечує інформування про відхилення від безпечних значень.

Галузь використання – екологічний моніторинг, контроль радіаційної безпеки на промислових об'єктах, у побутових умовах та на відкритих територіях.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 193 pages, 10 tables, 51 figures, 3 appendices, 31 references.

RADIATION BACKGROUND, CONTAMINATION, MEASUREMENT, RADIATION, C#.

Object of the study – the process of software engineering for monitoring background radiation levels.

Subject of the study – methods, algorithms, and software tools for the measurement, analysis, and real-time control of changes in background radiation.

Purpose of the study – to increase the accuracy of data collection and analysis of radiation measurements, as well as to enhance the speed of prompt user notification about hazardous changes in radiation levels.

Materials, methods, and technical means – object-oriented programming; the C# programming language; the Visual Studio 2022 integrated development environment; a personal computer equipped with an Intel Core i5 processor running the Microsoft Windows 10 operating system.

Results. The subject area was analyzed and a Windows-based software application was developed that monitors background radiation levels, records changes in indicators in real time, and provides notifications about deviations from safe threshold values.

Field of application – environmental monitoring, radiation safety control at industrial facilities, in domestic settings, and in open areas.

ЗМІСТ

	С.
Перелік скорочень та умовних позначень	8
Вступ.....	9
1 Аналіз предметної області.....	12
1.1 Основні поняття	12
1.2 Огляд існуючих методів збору даних	13
1.3 Порівняння методів збору даних	17
1.4 Порівняння застосунків зі схожим функціоналом.....	18
1.5 Постановка завдання до роботи.....	20
1.6 Висновок	21
2 Вибір та обґрунтування структури системи, яка проектується	23
2.1 Вибір мови програмування	23
2.2 Порівняння мов програмування	29
2.3 Вибір середовища розробки.....	30
2.4 Порівняння середовищ розробки	34
2.5 Висновок	35
3 Основні рішення щодо реалізації компонентів системи.....	37
3.1 Опис основних вимог до програми	37
3.2 Вимоги до технічних і програмних засобів.....	38
3.3 Розробка архітектури програми.....	39
3.4 Розробка користувацького інтерфейсу	51
3.5 Висновок	57
4 Імітація роботи радіаційного датчика	59
4.1 Загальна характеристика імітації радіаційного датчика	59
4.2 Обґрунтування вибору програмної імітації радіаційного датчика	60
4.3 Архітектурний підхід до реалізації датчика.....	61
4.4 Принцип роботи імітованого радіаційного датчика.....	63
4.5 Реалізація імітації радіаційного датчика в програмному коді	64

4.6 Особливості тестування та демонстрації роботи системи.....	67
4.7 Висновок	68
5 Керівництво програміста.....	70
5.1 Призначення й умови використання	70
5.2 Звертання до програмного продукту (файли проєкту).....	70
5.3 Вхідні і вихідні дані	81
5.4 Повідомлення.....	85
5.5 Запити до бази даних	87
5.6 Повідомлення оператору	90
5.7 Висновок	91
6 Керівництво оператора	92
6.1 Призначення й умови виконання програмного продукту.....	92
6.2 Виконання програмного продукту	93
6.3 Повідомлення оператору	97
6.4 Висновок	97
7 Експерименти та результати	99
7.1 Проведення експериментів	99
7.2 Висновок	107
Висновки	108
Перелік джерел посилання	109
Додаток А Технічне завдання	112
Додаток Б Текст програми	119
Додаток В Слайди презентації.....	187

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API	– Application Programming Interface;
IDE	– Integrated Development Environment;
ПП	– програмний продукт;
PВ	– радіоактивне випромінювання;
ПЗ	– програмне забезпечення;
МН	– машинне навчання.

ВСТУП

Актуальність. Контроль радіаційного фону є критично важливим у сучасних умовах, оскільки військові дії, аварії, пожежі чи збої на радіаційно небезпечних об'єктах можуть спричинити різкі зміни рівня випромінювання. Навіть незначні відхилення становлять загрозу для людей та довкілля, тому своєчасне виявлення небезпечних змін є необхідним. Автоматизовані системи забезпечують безперервний контроль і швидке сповіщення про перевищення норм.

Новизна. Хоча на ринку існують різні прилади для вимірювання радіації, більшість із них лише фіксують поточні значення без глибокого аналізу чи прогнозування. Розроблене програмне забезпечення не лише вимірює радіацію у реальному часі, а й аналізує динаміку, визначає ризики, прогнозує зміни та автоматично інформує користувача про небезпечні відхилення. Подібні комплексні рішення трапляються рідко.

Практична цінність. Своєчасний контроль радіаційного фону дає змогу зменшити ризики для здоров'я населення, довкілля та інфраструктури. Програмне забезпечення допомагає планувати заходи безпеки, організовувати евакуацію, контролювати промислові процеси та оцінювати екологічний стан територій. Воно корисне для підприємств, рятувальних служб, місцевої влади, фермерських господарств та приватних користувачів, оскільки автоматизований аналіз забезпечує прийняття обґрунтованих рішень.

Радіаційний фон – це рівень іонізуючого випромінювання, що впливає на живі організми та матеріальні об'єкти. Навіть невеликі перевищення норми можуть спричинити накопичення радіонуклідів, мутації або довготривалі екологічні проблеми. Коректний контроль дозволяє вчасно оцінювати стан довкілля та попереджати небезпечні наслідки.

Іонізуюче випромінювання негативно впливає на здоров'я: пошкоджує клітини, може провокувати рак, знижувати імунітет і викликати

генетичні зміни. Тому моніторинг є необхідним для:

- швидкого виявлення небезпечних рівнів;
- запобігання опроміненню людей та працівників;
- оцінки територій після аварій;
- контролю процесів, пов'язаних із радіацією;
- забезпечення довгострокової екологічної безпеки.

Сучасні системи дозволяють збирати історичні дані, аналізувати тенденції та оперативно сповіщати про відхилення. Проте мережі радіаційного контролю не завжди охоплюють локальні зони ризику, тому важливим є регулярне отримання актуальних показників та швидке реагування на зміни. Несвоєчасне виявлення підвищеної радіації може спричинити локальні чи масштабні екологічні наслідки.

Розвиток технологій дозволив створювати програмні продукти, що автоматизують вимірювання, аналіз і прогнозування радіаційного фону. Такі рішення стають ефективним інструментом для побутових, промислових, екологічних та аварійно-рятувальних потреб, підвищуючи рівень безпеки та допомагаючи запобігти небезпечним ситуаціям.

Об'єктом дослідження є процес інженерії програмного забезпечення для моніторингу радіаційного фону.

Предметом дослідження є методи, алгоритми та програмні засоби вимірювання, аналізу та оперативного контролю змін радіаційного фону.

Враховуючи вищезгадане, метою роботи стало збільшення точності збору та аналізу даних вимірювання радіаційного випромінювання та швидкості оперативного інформування користувача про його небезпечні зміни.

Для досягнення цієї мети необхідно вирішити ряд задач:

- дослідити предметну область та проаналізувати існуючі аналоги;
- визначити необхідні інструменти та засоби розробки;
- провести проектування та програмну реалізацію майбутнього застосунку;

- протестувати розроблений застосунок.

Результатом цієї роботи буде створення ПЗ з реалізацією методів моніторингу радіаційного фону і виявленню його відхилень від нормативних значень.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні поняття

Радіаційний фон – це рівень іонізуючого випромінювання, що постійно присутній у навколишньому середовищі. Він може мати природне походження (космічне випромінювання, радіоактивні елементи в ґрунті, будівельних матеріалах) або штучне (викиди підприємств, аварії на радіаційно небезпечних об'єктах, випробування ядерної зброї). Контроль радіаційного фону дозволяє оцінити рівень безпеки для людини, тварин і рослин, а також вчасно реагувати на можливі небезпечні відхилення [1].

Іонізуюче випромінювання – це вид випромінювання, який має достатню енергію для іонізації атомів і молекул середовища, через яке проходить. Найпоширенішими його видами є α -, β - та γ -випромінювання, а також рентгенівське випромінювання та нейтронний потік. Залежно від виду випромінювання змінюються його проникаюча здатність, небезпека та методи захисту [2].

Радіаційне забруднення – це наявність у навколишньому середовищі радіонуклідів у концентраціях, що перевищують природні показники або установлені норми. Воно може виникати через аварії, несанкціонований викид радіоактивних речовин, порушення правил зберігання чи транспортування матеріалів. Радіаційне забруднення небезпечне тим, що може накопичуватись у біологічних організмах і мати довготривалу дію [3].

Дозиметр – це прилад, призначений для вимірювання рівня іонізуючого випромінювання або накопиченої дози. Дозиметри можуть бути побутові, промислові, портативні, стаціонарні, а також інтегровані в автоматизовані системи моніторингу. Вони мають різну чутливість, діапазони вимірювань та принципи роботи (іонізаційні камери, сцинтиляційні датчики, напівпровідникові сенсори тощо).

Автоматизований моніторинг – це комплекс технологій і програмних засобів, що забезпечують безперервне вимірювання рівня радіації та

передачу даних у реальному часі.

Такі системи дозволяють:

- фіксувати зміну радіаційного фону;
- зберігати історичні показники;
- аналізувати тенденції;
- виявляти небезпечні відхилення;
- попереджати користувача про перевищення порогових значень

[4].

Інформаційні системи екологічного контролю – це програмно-апаратні комплекси, що забезпечують збір, зберігання, обробку та аналіз екологічних даних, зокрема пов'язаних із радіаційною безпекою. Такі системи можуть інтегруватися з дозиметрами, сенсорами IoT, хмарними платформами та мобільними пристроями.

Нормативи радіаційної безпеки – це встановлені державними та міжнародними органами межі допустимого рівня опромінення.

Вони включають:

- допустимі рівні опромінення населення;
- норми для працівників радіаційно небезпечних підприємств;
- рівні втручання (коли слід здійснити евакуацію чи інші дії);
- допустимі рівні забруднення довкілля та продуктів харчування

[5].

1.2 Огляд існуючих методів збору даних

Контроль радіаційного фону здійснюється різними методами, що відрізняються точністю, оперативністю, ступенем автоматизації та вимогами до оператора. Як і в інших галузях моніторингу, розрізняють три основні підходи: ручний, напівавтоматичний та автоматичний методи збору даних. Кожен із них має свої переваги та недоліки, а вибір залежить від масштабу завдань, необхідної точності, доступних ресурсів та вимог до швидкості

реагування.

1.2.1 Ручний метод збору даних радіаційного фону

Ручний метод передбачає використання побутових або професійних дозиметрів, дані з яких оператор фіксує вручну. Для цього застосовують такі прилади, як побутові радіометри, портативні дозиметри або професійні дозиметри-радіометри [6].

Розбір процедури збору даних:

а) підготовка обладнання:

- 1) вибір дозиметра з відповідним діапазоном вимірювань;
- 2) перевірка калібрування приладу;
- 3) підготовка журналу вимірювань або електронної таблиці;

б) зчитування показників:

- 1) вимірювання рівня радіації у визначених точках;
- 2) фіксація значення потужності дози або накопиченої дози;
- 3) урахування фонових умов (вітер, опади, близькість джерел

випромінювання);

в) запис даних:

- 1) зазначення часу, місця, умов та значення вимірювання;
- 2) фіксація оператора, що здійснив вимірювання;

г) аналіз даних:

- 1) пошук перевищень допустимих норм;
- 2) визначення можливих джерел аномалій;
- 3) прийняття рішень про додаткові перевірки або

обмежувальні заходи [6].

Переваги:

- не потребує складного обладнання;
- простий у впровадженні;
- підходить для локальних перевірок або разових вимірювань [7].

Недоліки:

- висока ймовірність людської помилки;
- неможливість постійного моніторингу;
- потреба у частому втручанні оператора;
- ризик затримки у виявленні небезпечних змін радіаційного фону

[7].

1.2.2 Напіваавтоматичний метод збору даних радіаційного фону

Напіваавтоматичний метод поєднує автоматичне вимірювання радіації із ручним отриманням та аналізом даних. Вимірювання здійснюється за допомогою цифрових дозиметрів або переносних радіологічних логерів, які зберігають інформацію у внутрішній пам'яті [8].

Розбір процедури збору даних:

а) підготовка інструментів:

- 1) встановлення логерів або цифрових дозиметрів;
- 2) налаштування інтервалів вимірювання (хвилина, година, доба);
- 3) підготовка програмного забезпечення для зчитування даних;

б) зчитування показників:

- 1) прилади автоматично вимірюють рівень радіації;
- 2) дані накопичуються у внутрішній пам'яті;

в) ручне зчитування даних:

- 1) періодичне підключення приладу до комп'ютера;
- 2) перенесення даних у базу або таблицю;
- 3) перевірка коректності вимірювань;

г) аналіз даних:

- 1) пошук тенденцій та пікових значень;
- 2) надсилання результатів відповідальним особам;

3) надсилання результатів відповідальним особам [8].

Переваги:

- зменшується кількість ручних операцій;
- дані збираються регулярно та з однаковими інтервалами;
- точність вища, ніж при ручному методі [9].

Недоліки:

- дані все ще необхідно зчитувати вручну;
- немає миттєвого реагування на відхилення;
- залежність від частоти перевірок оператором [9].

1.2.3 Автоматичний збір даних радіаційного фону

Автоматичний метод базується на використанні стаціонарних або бездротових IoT-сенсорів, що постійно вимірюють рівень радіації та передають дані в реальному часі на сервер або у хмарну систему. Це найбільш ефективний та точний спосіб моніторингу [10].

Розбір процедури збору даних:

а) підготовка інструментів:

- 1) встановлення датчиків у необхідних точках;
- 2) підключення пристроїв до мережі або радіоканалу;
- 3) налаштування програмної платформи для прийому та аналізу даних;

б) встановлення датчиків:

- 1) розміщення у критичних або контрольних точках;
- 2) калібрування для забезпечення точності;
- 3) тестова перевірка зв'язку;

в) автоматичний збір даних:

- 1) безперервне вимірювання рівня радіації;
- 2) передача інформації кожні кілька секунд або хвилин;
- 3) збереження даних у базі (сервер або хмара);

г) аналіз даних:

- 1) програмне забезпечення виявляє відхилення від норми;
- 2) система формує графіки, звіти, тенденції;
- 3) автоматичне сповіщення (SMS, e-mail, push) у випадку небезпеки;

г) реагування на відхилення:

- 1) оператор негайно отримує повідомлення;
- 2) приймаються відповідні заходи (обмеження доступу, додаткове вимірювання, виклик фахівців) [10].

Переваги:

- найвища точність та оперативність;
- можливість миттєвого реагування;
- повна автоматизація процесу;
- доступ до історичних даних і аналітики;
- масштабованість для великих територій [11].

Недоліки:

- висока вартість обладнання;
- потреба у стабільному живленні та зв'язку;
- необхідність технічного обслуговування системи [11].

1.3 Порівняння методів збору даних

Проаналізувавши роботи [6]-[11] було проведено порівняльний аналіз досліджених методів збору даних (табл.1.1).

Таблиця 1.1 – Порівняльний аналіз досліджених методів збору даних

Показники/Методи	Ручний	Напівавтоматичний	Автоматичний
Дешевизна	+	–	–
Зручність	–	–	+
Ефективність	–	–	+
Точність	–	+	+
Моніторинг в реальному часі	–	+	+
Простота	+	+	+

Як видно з таблиці 1.1, найкращим методом виявився автоматичний метод збору даних. Він має безліч переваг, що робить найкращим серед розглянутих методів. При застосуванні цього метода від оператора вимагається мінімум зусиль та зменшується такий чинник, як людський фактор. Саме тому цей метод планується взяти за основу при розробці ПП.

1.4 Порівняння застосунків зі схожим функціоналом

Як зазначено в попередніх підрозділах, найбільш ефективним методом збору даних є автоматичний моніторинг радіаційного фону, який забезпечує високу точність, безперервність вимірювань та швидке реагування на небезпечні зміни. Для порівняння із розроблюваним програмним забезпеченням було обрано такі системи моніторингу радіації:

- radonEye RD200 – побутовий і професійний прилад з мобільним застосунком [12];
- safecast SC – відкрита екологічна мережа з датчиками та вебплатформою [13];
- gammaGuard – мобільний застосунок із підтримкою Bluetooth-дозиметрів [14].

Порівняльний аналіз застосунків [12] – [14] наведено в таблиці 1.2.

Таблиця 1.2 – Порівняльний аналіз застосунків

Показники/ Застосунки	RadonEye RD200	Safecast SC	GammaGuard
Тип вимірювання	Локальне	Мережеве	Локальне
Моніторинг в реальному часі	Так	Так	Так
Встановлення порогових значень	Ні	Ні	Так
Автоматичне сповіщення про небезпеку	Ні	Ні	Так
Формування історії вимірювань	Так	Так	Так
Кросплатформність	Ні	Так	Ні
Підтримка зовнішніх датчиків	Ні	Так	Так
Аналітика та прогнозування	Ні	Так	Ні
Масштабованість мережі	Ні	Так	Ні

Із таблиці 1.2 видно, що всі аналоги мають як переваги так і недоліки. У розроблюваному ПЗ для можливості конкурентоспроможності треба реалізувати наступний набір функціоналу:

- підтримка локального та мережевого моніторингу одночасно;
- розширена система сповіщень про небезпечні зміни радіаційного фону;
- модулі аналітики та прогнозування, які відсутні у більшості побутових та мобільних рішень;
- висока масштабованість, що дозволяє використовувати систему як у побуті, так і на промислових об'єктах;
- гнучкі порогові значення, які користувач може налаштовувати відповідно до вимог конкретної території.

Завдяки розширеному функціоналу, можливості інтеграції з різними видами датчиків та підтримці аналітичних алгоритмів, запропоноване програмне забезпечення має перевагу серед існуючих рішень і може

забезпечити більш ефективний радіаційний моніторинг у різних сферах застосуванн.

1.5 Постановка завдання до роботи

В процесі роботи буде розроблено програмне забезпечення, яке забезпечить підвищення точності збору та аналізу даних вимірювань радіаційного забруднення, а також швидкості інформування користувача про небезпечні зміни рівня іонізуючого випромінювання.

Робота ПЗ забезпечується завдяки дослідженню та програмній реалізації методів моніторингу радіаційного фону і виявленню його відхилень від нормативних значень.

У роботі необхідно розглянути такі ключові проблеми:

- моніторинг поточного рівня радіаційного фону у визначених точках;
- аналіз динаміки змін радіаційного фону у часовому розрізі;
- виявлення небезпечних відхилень від встановлених нормативів;
- формування сповіщень та попереджень у реальному часі;
- збереження історичних даних у базі та можливість їх подальшого аналізу;
- підтримка декількох точок моніторингу та робота з різними типами дозиметричних пристроїв.

Одним із основних функціональних компонентів майбутнього ПЗ буде автоматичний моніторинг рівня радіації. Система періодично (частота налаштовується користувачем) зчитує показники з датчиків та аналізує їх на предмет перевищення допустимих значень. Якщо програмне забезпечення фіксує рівень, що перевищує встановлені пороги, користувач негайно отримує повідомлення. Це дозволяє швидко реагувати на потенційно небезпечну ситуацію, обмежити доступ до відповідної території або вжити інших заходів безпеки.

Ще одним важливим функціоналом є аналіз тенденцій змін радіаційного фону. Програма веде базу даних усіх вимірювань, що дозволяє наочно відстежувати підвищення або зниження рівнів радіації у різні часові періоди. Такий підхід дає змогу виявляти аномалії, прогнозувати можливі зміни та здійснювати довгостроковий контроль стану територій.

Оскільки система моніторингу може включати кілька датчиків, доречно розробити можливість керувати списком підключених пристроїв, що включає:

- додавання нового датчика;
- його налаштування;
- збереження параметрів;
- видалення або деактивацію;
- можливість перегляду показників з кожної точки вимірювання окремо.

Це дозволяє масштабувати систему та використовувати її як у невеликих приміщеннях, так і на великих територіях або промислових об'єктах.

Таким чином, з метою вирішення проблем, пов'язаних з контролем радіаційного фону та оперативним реагуванням на небезпечні ситуації, було прийнято рішення розробити програмне забезпечення з описаним вище функціоналом. Воно має забезпечити користувача точними, оперативними та аналітично обробленими даними, що значно підвищить рівень екологічної та радіаційної безпеки.

1.6 Висновок

У цьому розділі було проведено аналіз предметної області, визначено основні поняття, пов'язані з радіаційним фоном, іонізуючим випромінюванням, методами вимірювання та принципами роботи систем радіаційного моніторингу. Було розглянуто існуючі технічні рішення та їхні

функціональні характеристики.

Проведено аналіз існуючих методів збору даних: ручного, напівавтоматичного та автоматичного. Встановлено, що найбільш ефективним, точним та оперативним є автоматичний метод, який забезпечує постійний моніторинг, передавання даних у реальному часі та швидке реагування на потенційно небезпечні відхилення. Саме цей метод було обрано як основу для розробки програмного забезпечення.

Було виконано порівняння існуючих аналогів систем радіаційного моніторингу. Визначено майбутні функції та переваги створюваного ПЗ, зокрема розширені можливості аналізу, масштабованість, підтримка мережі датчиків, система сповіщень та інтелектуальний аналіз динаміки змін радіаційного фону.

Також було сформульовано і поставлено завдання до роботи згідно з вимогами до функціональності майбутнього програмного забезпечення.

Основні завдання включають автоматичне зчитування даних з датчиків радіаційного фону, фіксацію і аналіз показників, ведення історії вимірювань, виявлення небезпечних відхилень, формування сповіщень і можливість роботи з кількома точками моніторингу.

Отже, проведений аналіз підтверджує актуальність теми та необхідність створення програмного продукту, який дозволить підвищити рівень безпеки за рахунок своєчасного виявлення змін радіаційного фону та оперативного реагування на них.

2 ВИБІР ТА ОБҐРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ, ЯКА ПРОЄКТУЄТЬСЯ

2.1 Вибір мови програмування

Існують сотні мов програмування, кожна з яких має власні переваги та специфіку. Мова, що чудово підходить для одного проєкту, може виявитися абсолютно непридатною для іншого. Тому вибір мови програмування для створення програмного забезпечення є ключовим етапом. Найчастіше для розробки застосунків подібного рівня та призначення використовують Python, C#, R, Java та Swift [15].

2.1.1 R

R – це мова програмування, створена для статистичної обробки даних, роботи з графікою та виконання складних обчислень. Вона також є вільним обчислювальним середовищем з відкритим вихідним кодом у межах проєкту GNU. Мову R розробили як альтернативу мові S із Bell Labs: попри наявність відмінностей між ними, більшість програм, написаних мовою S, може бути виконана в середовищі R.

R спеціалізується на аналізі даних та розробленні програмних рішень, здатних виконувати статистичний і інтелектуальний аналіз. Вона широко використовується науковцями та дослідниками у сфері науки про дані завдяки великій кількості бібліотек та інструментів, що підтримують:

- очищення та підготовка даних;
- створення візуалізацій;
- навчання та оцінка алгоритмів МН та глибокого навчання.

Мову R часто обирають за здатність створювати високоякісні, публікаційні графіки, а також за наявність вбудованих математичних позначень і функцій. Сильні можливості візуалізації полегшують інтерпретацію даних, дають змогу швидко виявляти закономірності,

аномалії й тенденції у великих наборах даних [16].

Переваги:

- потужні статистичні та аналітичні можливості, що дозволяють виконувати складні розрахунки й моделювання;
- наявність великої кількості пакетів, у тому числі для створення графічних інтерфейсів (Shiny, RGtk2);
- активна та чисельна спільнота, яка постійно розширює екосистему пакетів;
- розплатформеність: програми на R працюють у Windows, macOS і Linux [16].

Недоліки:

- обмежені можливості для створення складних GUI-застосунків порівняно з такими мовами, як Python та C#;
- як інтерпретована мова, R зазвичай повільніша за компільовані мови;
- складніша процедура розгортання додатків через необхідність встановлення R-середовища та залежностей;
- відносно невеликий вибір інструментів, орієнтованих саме на створення настільних застосунків [16].

2.1.2 C#

C# (C-sharp) – сучасна об'єктно-орієнтована мова програмування, розроблена компанією Microsoft у межах платформи .NET. Її створено для забезпечення швидкої розробки продуктивних, стабільних і масштабованих програмних рішень. C# широко застосовується для створення настільних і вебдодатків, ігор (зокрема на базі Unity) та багатьох інших типів програмного забезпечення [17].

Під час розробки настільних застосунків мовою C# зазвичай застосовують такі фреймворки, як Windows Forms, WPF (Windows

Presentation Foundation) та сучасний .NET MAUI (Multi-platform App UI), що підтримує створення кросплатформних додатків [17].

Переваги:

- C# розроблено для роботи з платформою .NET, що забезпечує тісну інтеграцію з операційною системою Windows та доступ до широких можливостей ОС;
- Visual Studio – потужне середовище розробки від Microsoft – пропонує зручні інструменти для програмування, відлагодження та розгортання застосунків;
- платформа .NET містить велику кількість вбудованих бібліотек, які полегшують роботу з базами даних, графікою, мережами та іншими компонентами;
- завдяки .NET Core та .NET 5/6/7 програми на C# можуть працювати на Windows, macOS і Linux;
- мова активно розвивається та підтримує сучасні можливості, такі як асинхронне програмування, шаблони, LINQ та інші інструменти, що підвищують гнучкість і зручність розробки [17].

Недоліки:

- через широкі можливості та багатий функціонал C# може бути складною мовою для новачків;
- попри розширення кросплатформності, екосистема C# усе ще значною мірою орієнтована на продукти Microsoft, що може створювати певні обмеження;
- навіть прості програми можуть потребувати додаткових зусиль через структуру .NET-фреймворків і формальні вимоги мови;
- хоча .NET забезпечує високу продуктивність, у деяких випадках застосунки на C# поступаються за швидкістю рішенням, створеним на низькорівневих мовах, наприклад C++ [17].

2.1.3 Python

Python – високорівнева мова програмування загального призначення, відома своєю простотою, читабельністю та широкими можливостями. Хоча Python часто пов'язують із веброзробкою, аналізом даних та машинним навчанням, він також є ефективним інструментом для створення настільних застосунків. Для цього використовуються фреймворки Tkinter, PyQt, PySide, Kivy, wxPython та інші, які забезпечують засоби для побудови графічних інтерфейсів користувача (GUI), що працюють на Windows, macOS і Linux [18].

Переваги Python:

- Python має простий і зрозумілий синтаксис, що робить його зручним для початківців і дозволяє швидко створювати прототипи;
- більшість GUI-фреймворків підтримують кросплатформеність, завдяки чому додатки можуть працювати на різних операційних системах;
- мова має велику екосистему сторонніх бібліотек, які спрощують реалізацію роботи з мережами, базами даних, файлами та іншими ресурсами;
- швидкість розробки на Python є високою завдяки лаконичному синтаксису та великій кількості готових інструментів;
- Python має одну з найбільших та найактивніших спільнот, що забезпечує доступ до документації, навчальних матеріалів та регулярну підтримку [18].

Недоліки Python:

- Python є інтерпретованою мовою, тому настільні програми можуть працювати повільніше порівняно з програмами, написаними на компільованих мовах, таких як C++ або Java;
- створення інсталяційних пакетів може бути складним через особливості структури Python-проектів;
- застосунки часто залежать від великої кількості сторонніх бібліотек, що ускладнює налаштування середовища для кінцевих

користувачів;

- хоча Python має декілька GUI-бібліотек, їх функціональність або зручність інколи поступаються нативним рішенням інших мов [18].

2.1.4 Swift

Swift – мова програмування, створена компанією Apple та представлена у 2014 році. Спочатку її розробляли для створення застосунків під iOS і macOS, однак з часом Swift здобув популярність і в інших напрямках. Використання Swift для розробки настільних програм під macOS дозволяє повністю інтегруватися з екосистемою Apple та застосовувати сучасні фреймворки для побудови графічних інтерфейсів, зокрема SwiftUI та AppKit [19].

Переваги:

- Swift компілюється в машинний код, що забезпечує високу продуктивність програм, порівнянну з рішеннями, написаними на C++;
- мова має розвинені засоби забезпечення безпеки коду: автоматичне управління пам'яттю, механізми запобігання переповненню буфера та інші;
- Swift поєднує потужність і простоту, пропонуючи сучасний, лаконічний і легкий для читання синтаксис, який значно зручніший за Objective-C;
- SwiftUI – новий декларативний фреймворк для побудови користувацьких інтерфейсів, що суттєво спрощує розробку UI для macOS та інших платформ Apple;
- Swift забезпечує зручну інтеграцію з екосистемою Apple, включаючи сервіси та технології на кшталт iCloud, Core Data та інших [20].

Недоліки:

- основна сфера застосування Swift зосереджена в екосистемі Apple – iOS, macOS, watchOS та tvOS. Попри наявність проєктів для інших

платформ, кросплатформні можливості залишаються обмеженими;

- порівняно з мовами на кшталт JavaScript чи Python, можливості Swift для кросплатформної розробки є менш розвиненими;
- Swift, хоч і активно розвивається, є відносно молодою мовою, а отже має меншу кількість бібліотек, інструментів і фреймворків;
- через швидкі темпи розвитку нові версії Swift іноді вводять зміни, що не завжди є зворотно сумісними, і це може вимагати оновлення існуючого коду [21].

2.1.5 Java

Java – об'єктно-орієнтована мова програмування загального призначення та водночас обчислювальна платформа. Спочатку її створювали для роботи у вбудованих мережевих системах, що потребували можливості запуску на різних апаратних і програмних платформах. Java орієнтована на розроблення високопродуктивних і портативних програмних рішень, що забезпечує підтримку ключових принципів доступності та міжплатформної сумісності.

Завдяки використанню інтерпретатора Java (Java Virtual Machine) програми, написані цією мовою, працюють однаково на будь-якому пристрої незалежно від операційної системи та апаратного забезпечення. Окрім портативності, важливою перевагою Java є широкий набір механізмів безпеки, що захищають систему від помилок у коді та шкідливих програм [22].

Переваги:

- Java дозволяє кільком комп'ютерам у мережі працювати спільно над виконанням одного завдання;
- мова є практичною та зручною для програмування завдяки чіткій структурі та універсальності;
- програми на Java можна запускати на різних платформах без

повторної компіляції;

- Java не використовує явні покажчики, а вбудований менеджер безпеки контролює доступ до класів і ресурсів;
- мова підтримує багатопоточність, що дає змогу розбивати складні завдання на паралельні потоки та підвищувати продуктивність [22].

Недоліки:

- виконання програм у середовищі віртуальної машини робить Java менш швидкою та більш ресурсоємною порівняно з нативними мовами;
- хоча Java підтримує автоматичне збирання сміття, програмісти не мають можливості вручну керувати очищенням пам'яті, оскільки мова не пропонує аналогів методів `delete()` або `free()` [22].

2.2 Порівняння мов програмування

Проаналізувавши роботи [15]-[22] було проведено порівняльний аналіз мов програмування, який наведено у табл. 2.1.

Таблиця 2.1 – Порівняльна характеристика мов програмування

Критерії порівняння	R	C#	Python	Swift	Java
Ліцензія	GNU GPL	RAND	Python Software Foundation License	GNU GPL	MIT
Простота використання	—	+	+	+	—
Підтримка кросплатформеності	+	+	+	—	+
Продуктивність	+	+	+	+	+
Розвиток та підтримка	+	+	+	+	+
Візуальні інтерфейси (GUI)	+	+	—	+	+
Підтримка великих проєктів	—	+	—	—	+
Спільнота та ресурси	+	+	+	+	+
Інтеграція з іншими технологіями	+	+	+	—	+

Як видно з таблиці 2.1, мова програмування C# має найбільшу кількість переваг порівняно з іншими розглянутими мовами. Вона

характеризується високою гнучкістю та здатністю ефективно працювати в різних середовищах. Завдяки мінімальним змінам C# може функціонувати на будь-якій операційній системі та легко інтегруватися з іншими мовами програмування.

Мова також має розвинену екосистему бібліотек і інструментів, що суттєво спрощує виконання завдань, пов'язаних з аналізом даних і розробленням програмних рішень. Важливою перевагою є й велика активна спільнота, яка забезпечує підтримку та обмін досвідом між розробниками.

Отже, для створення програмного забезпечення було обрано саме мову програмування C#.

2.3 Вибір середовища розробки

Середовище розробки дає змогу значно спростити процес написання, тестування та налагодження програмного коду. Воно зазвичай забезпечує засоби автодоповнення, підсвічування синтаксису, керування ресурсами та інструменти для налагодження. Використання середовища розробки дозволяє суттєво заощадити час під час створення програм, швидко виправляти типові помилки та ефективно працювати навіть із великими проєктами.

Найпоширенішими середовищами розробки для створення програмного забезпечення є Visual Studio, Visual Studio Code, JetBrains Rider та MonoDevelop (Xamarin Studio) [23].

2.3.1 Visual Studio

Visual Studio – інтегроване середовище розробки (IDE), створене компанією Microsoft для операційних систем Windows та macOS. Воно підтримує широкий спектр мов програмування та платформ, зокрема .NET, C++, Python, JavaScript та інші. Visual Studio інтегрується з системами

контролю версій, такими як Git, TFS і SVN, що дає змогу ефективно керувати проєктами, відстежувати зміни та працювати з репозиторіями.

Середовище також пропонує потужну систему IntelliSense, яка забезпечує автодоповнення коду, підсвічування синтаксису, інструменти для рефакторингу та інші функції, спрямовані на підвищення продуктивності розробника. Крім того, Visual Studio підтримує створення, налагодження та тестування додатків, включно з можливістю виконання модульних тестів [24].

Переваги:

- потужні інструменти для розробки, налагодження та тестування програм;
- підтримка великої кількості мов програмування та платформ;
- інтеграція із системами контролю версій (Git, TFS, SVN);
- розширені можливості IntelliSense для автодоповнення, підсвічування синтаксису та рефакторингу;
- інтеграція з Azure, що спрощує розгортання та керування хмарними застосунками [24].

Недоліки:

- висока ресурсомісткість, що може впливати на продуктивність під час роботи з великими проєктами;
- обмежена кросплатформеність порівняно з іншими IDE, наприклад Visual Studio Code;
- висока вартість ліцензій для професійних і корпоративних версій [24].

2.3.2 Visual Studio Code

Visual Studio Code – це безкоштовне, кросплатформне середовище розробки з відкритим вихідним кодом, створене компанією Microsoft для Windows, Linux та macOS. Воно інтегрується з локальними та віддаленими репозиторіями Git, включаючи GitHub і Azure Repos, що дає змогу клонувати

репозиторії, фіксувати зміни та управляти гілками безпосередньо з редактора.

Середовище має IntelliSense – систему розумного автодоповнення, яка пропонує варіанти коду на основі типів змінних, функцій і підключених модулів, а також забезпечує підсвічування синтаксису й автоматичне заповнення. Крім Python, Visual Studio Code підтримує багато мов програмування, таких як C, C#, JavaScript, HTML, CSS та інші, що робить його зручним інструментом для роботи з різними API та бібліотеками. Також редактор підтримує запуск модульних тестів через unittest або pytest [25].

Переваги:

- просте управління життєвим циклом проєктів у сфері аналізу даних;
- широкі можливості налаштування функціоналу та інтерфейсу;
- наявність IntelliSense, що допомагає визначати та доповнювати фрагменти коду;
- зручність швидкого створення прототипів і подальшого об'єднання компонентів у великі проєкти;
- підтримка роботи на Windows, Linux та macOS [25].

Недоліки:

- менш розвинений інтерфейс користувача порівняно з Visual Studio;
- можливі збої під час встановлення або роботи деяких розширень;
- відсутність вбудованих засобів для перегляду SVG-файлів [25].

2.3.3 JetBrains Rider

JetBrains Rider – це комерційне кросплатформне середовище розробки, створене компанією JetBrains для Windows, Linux та macOS. Воно інтегрується з локальними та віддаленими репозиторіями Git, зокрема GitHub і Bitbucket, що дає змогу клонувати репозиторії, фіксувати зміни та керувати

гілками безпосередньо в IDE.

Середовище побудоване на основі IntelliJ IDEA та оснащене розширеною системою IntelliSense, яка пропонує автодоповнення, підсвічування синтаксису, інструменти рефакторингу та інші функції для підвищення ефективності розробки. Окрім C#, JetBrains Rider підтримує численні мови програмування, включаючи Java, JavaScript, TypeScript, HTML і CSS, що робить його зручним для роботи з різними API та бібліотеками. IDE також підтримує виконання модульних тестів через фреймворки NUnit, xUnit та MSTest [26].

Переваги:

- професійні інструменти для розробки, налагодження та тестування додатків;
- широка підтримка мов програмування й платформ;
- інтеграція з популярними системами контролю версій (Git, GitHub, Bitbucket);
- потужна IntelliSense, заснована на IntelliJ IDEA, із підтримкою автодоповнення, підсвічування синтаксису та рефакторингу;
- кросплатформна підтримка Windows, Linux і macOS [26].

Недоліки:

- висока вартість ліцензії порівняно з безкоштовними аналогами;
- ресурсомісткість, яка може знижувати продуктивність під час роботи з великими проєктами;
- складність для новачків через велику кількість налаштувань і можливостей [26].

2.3.4 MonoDevelop (Xamarin Studio)

MonoDevelop (також відомий як Xamarin Studio) – це безкоштовне кросплатформне середовище розробки з відкритим вихідним кодом, призначене для роботи в операційних системах Windows, Linux та macOS.

Воно інтегрується з локальними й віддаленими репозиторіями Git, що дає змогу клонувати репозиторії, фіксувати зміни та управляти гілками без виходу з IDE [27].

Середовище підтримує систему IntelliSense, яка забезпечує автодоповнення коду, підсвічування синтаксису та інтелектуальний рефакторинг. Окрім C#, MonoDevelop підтримує такі мови програмування, як F# та Visual Basic .NET, що робить його корисним інструментом для розробки мультиплатформних застосунків. IDE також дозволяє виконувати модульні тести через різні фреймворки, спрощуючи налагодження та контроль якості програмного забезпечення [27].

Переваги:

- зручне керування життєвим циклом проєктів;
- широкі можливості індивідуального налаштування;
- підтримка IntelliSense для автодоповнення та рефакторингу;
- можливість швидкого створення прототипів і подальшого об'єднання компонентів у великі проєкти;
- кросплатформена підтримка Windows, Linux та macOS [27].

Недоліки:

- менш розвинена функціональність інтерфейсу користувача порівняно з Visual Studio;
- потенційно нижча стабільність у порівнянні з іншими сучасними IDE;
- обмежена підтримка частини мов програмування в порівнянні з більш універсальними середовищами [27].

2.4 Порівняння середовищ розробки

Проаналізувавши роботи [23] – [27] було проведено порівняльний аналіз середовищ розробки, який наведений у табл. 2.2.

Таблиця 2.2 – Порівняльна характеристика середовищ розробки

Критерії порівняння	Visual Studio	Visual Studio Code	JetBrain s Rider	Xamarin Studio
Ліцензія	MIT	MIT	—	MIT
Кросплатформність	+	+	+	—
Автозавершення коду	+	+	+	—
Підтримка репозиторіїв	+	+	+	+
Простота встановлення	+	—	+	—
Перевірка та аналіз коду	+	+	+	+
Швидкий та безпечний рефакторинг коду	+	+	—	+
Наявність інтерактивної консолі	+	+	—	+
Наявність зручного користувацького інтерфейсу	+	+	+	+
Швидкий та зручний пошук	+	+	—	—
Підсвічування синтаксису	+	+	+	+
Підтримка наукових пакетів	+	+	+	—
Наявність функції статичного аналізу коду	+	+	+	+

З таблиці 2.2 видно, що середовище розробки Visual Studio має найбільшу кількість переваг серед розглянутих аналогів. Воно вирізняється простотою освоєння, широкими можливостями та орієнтованістю на створення багатофункціональних додатків. Visual Studio включає зручний текстовий редактор із підсвічуванням синтаксису, автодоповненням коду, а також інструментами для перегляду та редагування значень змінних через графічний інтерфейс.

Завдяки своїм можливостям Visual Studio забезпечує ефективне створення, налагодження та підтримку програмного забезпечення. Тому для розробки програмного продукту було обрано саме це середовище.

2.5 Висновок

У цьому розділі було проведено аналіз та вибір інструментарію для розробки програмного забезпечення з відстеження змін радіаційного фону, а також спроектовано архітектуру майбутнього ПП. Було здійснено

порівняльний аналіз мов програмування, придатних для створення систем моніторингу радіаційного фону, та встановлено, що найбільші переваги має мова програмування C#. Це зумовлено її високою гнучкістю, широким набором бібліотек, наявністю готових рішень для роботи з обладнанням, сенсорними пристроями та мережевими інтерфейсами, що значно спрощує реалізацію функціоналу моніторингу та обробки даних у реальному часі.

Було проведено порівняльний аналіз середовищ розробки, у ході якого встановлено, що найбільш оптимальним є середовище Visual Studio. Воно відзначається простотою освоєння, потужним інструментарієм для створення графічних інтерфейсів, підтримкою різних платформ та інтеграцією з системами контролю версій. Великий набір доступних бібліотек та розширень дозволяє швидко реалізовувати функції, пов'язані з опрацюванням даних дозиметрів, аналізом показників, збереженням інформації у базах даних та побудовою системи повідомлень.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Опис основних вимог до програми

Опис основних вимог є одним із ключових етапів у процесі розробки програмного забезпечення, оскільки він визначає функціональні можливості системи, вимоги до її роботи та взаємодії з користувачем. Чітке формулювання вимог забезпечує коректну реалізацію програмного продукту, його надійність, зручність використання та відповідність поставленим завданням моніторингу радіаційного фону.

Розглянемо перелік нефункціональних вимог:

- назва програмного продукту – «Radiation Monitoring System»;
- тип програмного продукту – настільний Windows-застосунок;
- мова інтерфейсу – українська;
- інтерфейс користувача повинен бути інтуїтивно зрозумілим та не потребувати спеціальної підготовки;
- програмний продукт повинен забезпечувати стабільну роботу в режимі реального часу;
- забезпечення збереження даних вимірювань та подій у базі даних;
- можливість масштабування шляхом додавання нових датчиків.

Розглянемо перелік функціональних вимог:

а) моніторинг радіаційного фону:

- 1) зчитування показників радіаційного фону з датчиків у реальному часі;
- 2) відображення поточних значень радіації на графіку;
- 3) збереження результатів вимірювань у базі даних;
- 4) перегляд історії вимірювань за вибраний період (хвилини, години, дні, довільний інтервал);

б) контроль порогових значень:

- 1) встановлення мінімального та максимального допустимих порогів радіаційного фону;
 - 2) автоматичне виявлення перевищення порогових значень;
 - 3) формування попереджувальних повідомлень у разі перевищення допустимого рівня радіації;
- в) управління датчиками радіаційного моніторингу:
- 1) додавання нового датчика з вказаними параметрами;
 - 2) редагування параметрів існуючого датчика;
 - 3) видалення датчика з системи;
 - 4) вибір датчика для перегляду даних та моніторингу;
- г) журнал подій:
- 1) автоматичний запис системних подій у журнал (запуск/зупинка програми, дії користувача);
 - 2) фіксація подій перевищення порогових значень радіації;
 - 3) перегляд журналу подій за вибраний період;
 - 4) фільтрація подій за одним або кількома датчиками;
- г) робота з періодами спостереження:
- 1) вибір типу періоду (хвилини, години, дні);
 - 2) вибір довільного часового інтервалу за допомогою DatePicker;
 - 3) коректна обробка часових меж та перевірка правильності введених дат.

3.2 Вимоги до технічних і програмних засобів

Вимоги до технічних та програмних засобів визначають мінімальні характеристики апаратного та програмного забезпечення, які повинні мати робочі станції користувачів для коректної та стабільної роботи програмного продукту моніторингу радіаційного фону. У даному пункті наведено основні вимоги до комп'ютерного обладнання та програмного середовища, що

безпосередньо впливають на продуктивність, надійність і функціональність настільного застосунку.

Врахування зазначених вимог дозволяє забезпечити безперебійну роботу програмного продукту, коректне відображення результатів вимірювань у режимі реального часу, своєчасну обробку даних з датчиків та зручний користувацький інтерфейс незалежно від умов експлуатації.

Вимоги до складу та параметрів технічних засобів:

- платформа x64-based;
- операційна система Microsoft Windows 11/10/8/7 (рекомендується встановити всі останні оновлення);
- процесор з мінімальною тактовою частотою 1,8 ГГц, рекомендується використовувати як мінімум двоядерний процесор;
- оперативна пам'ять від 2 ГБ (рекомендується 4 ГБ та мінімум 2,5 ГБ при використанні на віртуальній машині);
- роздільна здатність екрану 1920x1080 пікселів з 16-бітовим кольором;
- мінімум 2 ГБ вільного місця на жорсткому диску.

3.3 Розробка архітектури програми

Розробка архітектури програми є одним із ключових етапів у процесі створення настільного програмного продукту для моніторингу радіаційного фону. Архітектура визначає загальну структуру застосунку, склад його програмних модулів, а також механізми взаємодії між ними, що безпосередньо впливає на стабільність роботи, ефективність обробки даних та можливість подальшого розширення функціоналу.

У даному підрозділі розглядається процес побудови архітектури програмного продукту, визначаються основні компоненти системи, такі як модулі збору та обробки даних з радіаційних датчиків, модуль візуалізації

результатів вимірювань, модуль роботи з базою даних та модуль журналу подій, а також описуються зв'язки між ними. Окрему увагу приділено підходам до організації асинхронного моніторингу, обробки подій перевищення порогових значень та взаємодії з користувацьким інтерфейсом.

Для наочного представлення архітектурних рішень та взаємодії компонентів у процесі проєктування програмного продукту використовуються UML-діаграми (Unified Modeling Language), які дозволяють чітко відобразити структуру системи, логіку її роботи та сценарії взаємодії між основними модулями застосунку.

3.3.1 Вибір шаблону проєктування

Вибір шаблону проєктування є одним з ключових етапів у процесі розробки програмного забезпечення, оскільки він визначає загальну структуру застосунку, принципи організації коду та взаємодію між основними частинами системи. Використання шаблонів проєктування дозволяє підвищити зрозумілість архітектури, спростити подальший супровід, забезпечити розширюваність і зменшити кількість помилок при розвитку функціоналу.

У даній роботі розроблюється Windows-застосунок для моніторингу радіаційного фону, який виконує такі задачі: керування переліком датчиків, отримання та збереження вимірювань у базі даних, відображення показників у реальному часі, перегляд історичних даних за період, формування попереджень при перевищенні порогів та ведення журналу подій. Для коректної реалізації цих задач необхідно застосувати підхід, що дозволяє розділити відповідальності між інтерфейсом, логікою моніторингу та доступом до даних.

Найбільш універсальними та високорівневими є архітектурні шаблони проєктування, оскільки їх можна застосувати практично в будь-якій мові програмування. На відміну від локальних шаблонів (для окремих класів

чи алгоритмів), архітектурні підходи дозволяють проектувати структуру застосунку в цілому.

З огляду на те, що програмний продукт реалізовано мовою C# у вигляді WinForms-застосунку, для побудови архітектури доцільно розглянути такі підходи:

- MVC (Model-View-Controller);
- MVP (Model-View-Presenter);
- MVVM (Model-View-ViewModel);
- багатошарова архітектура.

MVC (Model-View-Controller) – архітектурний шаблон, у якому модель відповідає за дані та правила їх обробки, представлення – за відображення, а контролер – за обробку дій користувача та взаємодію між моделлю і представленням. Цей підхід широко використовується у веброзробці, але в WinForms він не завжди реалізується “чисто”, оскільки форма часто містить обробники подій, які поєднують керування UI та частину логіки [28].

MVP (Model-View-Presenter) – підхід, який добре підходить для настільних застосунків з подієвою моделлю (WinForms). Представлення (View) відображає дані та генерує події, а Presenter містить основну логіку взаємодії: отримує дані з моделі/сервісів, обробляє їх і передає в View для відображення. Такий підхід спрощує тестування і зменшує залежність логіки від UI [29].

MVVM (Model-View-ViewModel) – шаблон, який найкраще підходить для WPF через наявність прив’язки даних (Data Binding), команд (Commands) та розвинених механізмів роботи з інтерфейсом. У WinForms MVVM можливо застосовувати, але це потребує додаткових рішень для прив’язки і зазвичай менш зручно, ніж у WPF [30].

Багатошарова архітектура передбачає розділення на шари:

- UI-шар (форми, елементи керування);
- шар бізнес-логіки (моніторинг датчиків, перевірка порогів,

логування подій);

– шар доступу до даних (робота з MySQL, таблицями sensors, sensor_readings, event_log).

Цей підхід є практичним для настільних застосунків, забезпечує прозору структуру, спрощує розвиток функціоналу та дозволяє замінювати окремі частини системи без повної переробки застосунку (наприклад, заміна емулятора датчика на реальний пристрій або зміна СКБД) [31].

Порівняємо доступні архітектурні підходи для побудови Windows-застосунку моніторингу радіаційного фону (таблиця 3.1).

Таблиця 3.1 – Порівняння архітектурних підходів

Архітектурний підхід	Переваги	Недоліки
MVC	Зрозуміле розділення ролей; популярний шаблон; зручно структурує моделі	У WinForms важко дотримуватися “чистого” MVC, форма часто стає контролером і представленням одночасно
MVP	Добре підходить для WinForms; логіка винесена з форм; легше тестувати	Потрібно проєктувати Presenter і інтерфейси View, більше “службового” коду
MVVM	Зручно для UI-архітектури при наявності data binding (особливо у WPF)	У WinForms менш природний; прив’язка даних і команди потребують додаткових рішень
Багатошарова архітектура (UI–Logic–Data)	Чітке розмежування відповідальностей; легко розширювати; зручно замінювати датчики/БД; природно для WinForms	Частина логіки може залишатися в формах, якщо не контролювати структуру; потрібна дисципліна розділення коду

Порівнявши доступні варіанти, було прийнято рішення використовувати багатошарову архітектуру з елементами MVP-підходу. Це дозволяє забезпечити чітке розділення відповідальностей між інтерфейсом

користувача (форми WinForms), логікою моніторингу (сервіс опитування датчиків і перевірки порогів) та шаром доступу до даних (репозиторій для взаємодії з MySQL). Такий підхід є найбільш практичним для реалізації задач моніторингу радіаційного фону, оскільки підтримує одночасну роботу з кількома датчиками, забезпечує стабільне збереження вимірювань і подій у базі даних та дає можливість розширювати застосунок без суттєвого перепроєктування.

3.3.2 UML-діаграма розгортання

На верхньому рівні система містить (рис.3.1):

- клієнтський ПК оператора з Windows-застосунком;
- сервер бази даних MySQL (radiation_db);
- джерела даних датчиків: реальні або імітаційні.

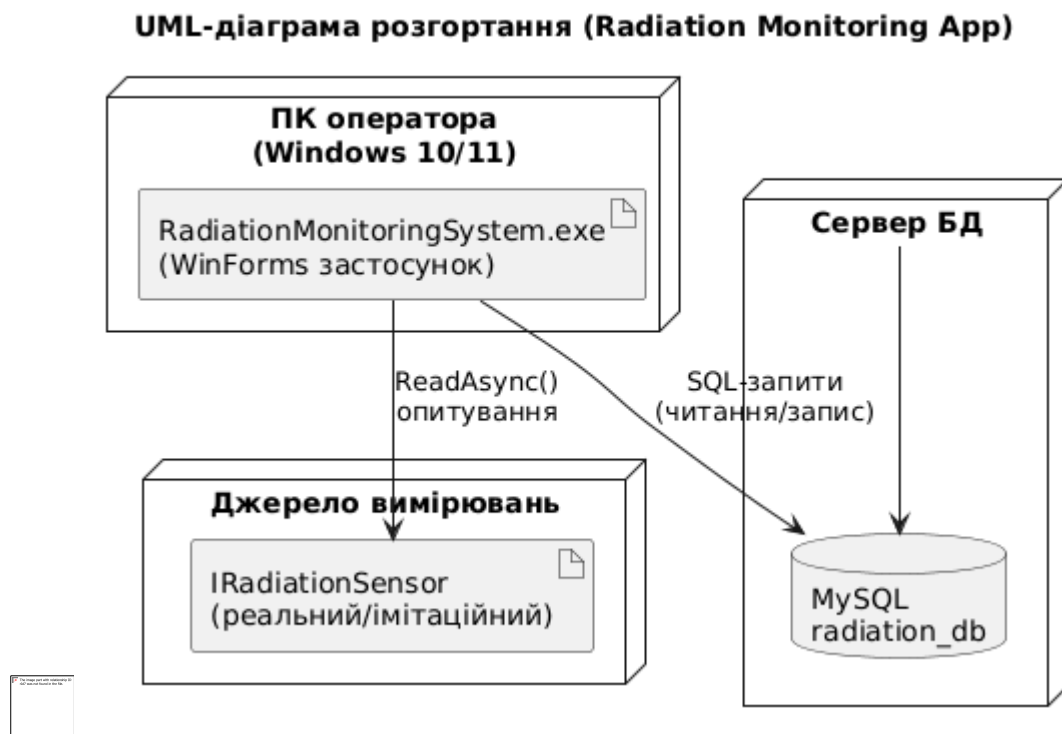


Рисунок 3.1 – UML-діаграма розгортання

3.3.3 UML-діаграми компонентів

Компонентна структура відображає поділ застосунку на модулі(рис. 3.2):

- UI (WinForms): MainForm, FormEventLog;
- Monitoring: SensorMonitoringService (опитування датчиків, події ReadingReceived/ThresholdExceeded);
- Data Access: SensorRepository (робота з sensors, sensor_readings, event_log);
- Models: SensorConfig, SensorReading, EventLogEntry;
- Sensors: IRadiationSensor + FakeRadiationSensor.

Це відповідає реальній логіці: сервіс моніторингу піднімає події та формує вимірювання, а головна форма зчитує/малює дані та перемикає режими monitoring/history.

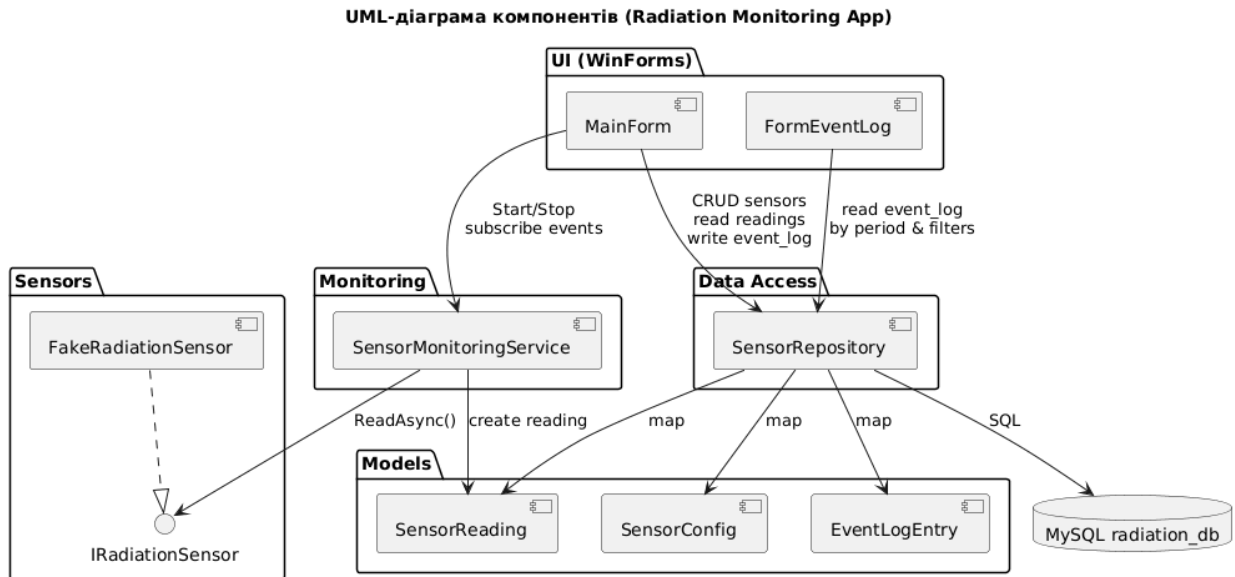


Рисунок 3.2 – UML-діаграми компонентів

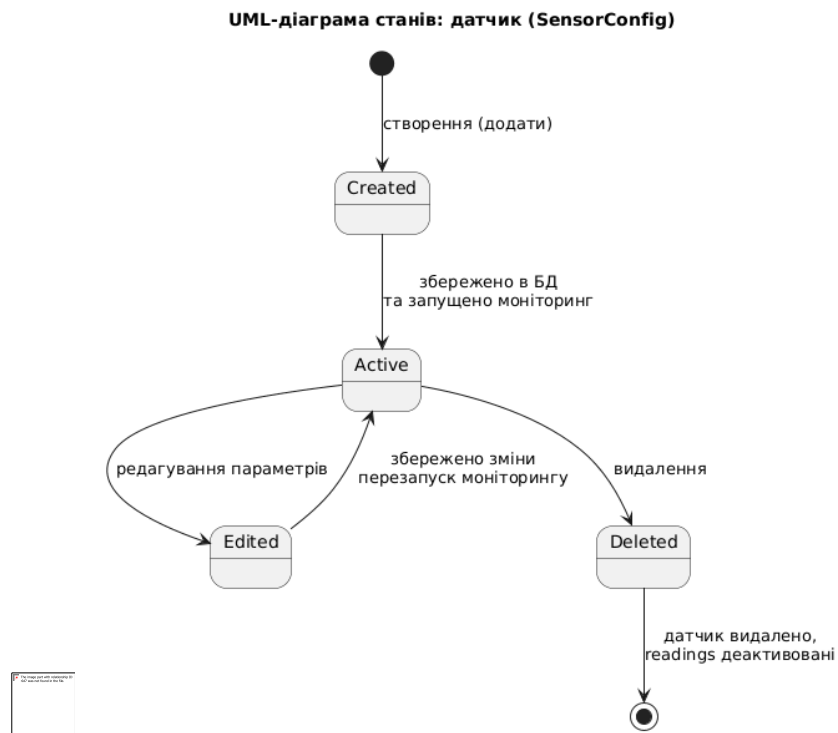


Рисунок 3.4 – UML-діаграм стану датчика

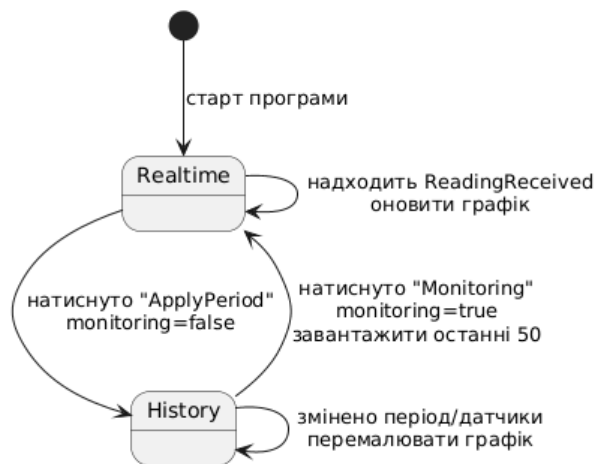
UML-діаграма станів: режим графіка (monitoring)

Рисунок 3.5 – UML-діаграм стану режиму відображення датчика

3.3.6 Проєктування бази даних

Для зберігання конфігурацій датчиків, результатів вимірювань та системних повідомлень у розроблюваному Windows-застосунку використовується реляційна база даних MySQL. Структура БД побудована на основі сутностей предметної області та їх зв'язків. Підхід до опису сутностей

і подальшого подання їх полів у вигляді таблиць відповідає шаблону, де для кожної сутності задається перелік атрибутів і подається їх опис у табличній формі.

Сутність «sensors» (Id, Name, Place, Model, MinThreshold, MaxThreshold, PollingSec, Smh, Start, Status, Comment) – сутність, що описує конфігурацію радіаційного датчика: його ідентифікатор, місце встановлення, модель, порогові значення, частоту опитування, стан та додаткові відомості. Опис полів сутності наведено у таблиці 3.2.

Таблиця 3.2 – Опис полів сутності «sensors»

№	Назва поля	Тип даних
1	Id	varchar
2	Name	varchar
3	Place	varchar
4	Model	varchar
5	MinThreshold	double
6	MaxThreshold	double
7	PollingSec	int
8	Smh	int
9	Start	datetime
10	Status	varchar
11	Comment	text

Сутність «sensor_reading» (Id, SensorId, Timestamp, Value, Active) – сутність, що представляє одне вимірювання рівня радіаційного фону, отримане від датчика. Містить посилання на датчик, дату/час вимірювання, значення та ознаку активності запису (для логічного видалення історичних даних). Опис полів сутності наведено у таблиці 3.3.

Таблиця 3.3 – Опис полів сутності «sensor_reading»

№	Назва поля	Тип даних
1	Id	int
2	SensorId	varchar
3	Timestamp	datetime
4	Value	double
5	Active	tinyint

Сутність «event_log» (Id, SensorId, Message, CreatedAt) – сутність, що описує запис журналу подій системи. Вона використовується для фіксації дій користувача (додавання/редагування/видалення датчика), службових повідомлень (запуск/закриття програми) та попереджень (перевищення порогів). Поле SensorId може бути NULL для загальних подій, які не належать конкретному датчику. Опис полів сутності наведено у таблиці 3.4.

Таблиця 3.4 – Опис полів сутності «event_log»

№	Назва поля	Тип даних
1	Id	int
2	SensorId	varchar (nullable)
3	Message	varchar
4	CreatedAt	datetime

Визначивши сутності можна побудувати концептуальну моделі бази даних що відображає їх зв'язки на рисунку 3.6.

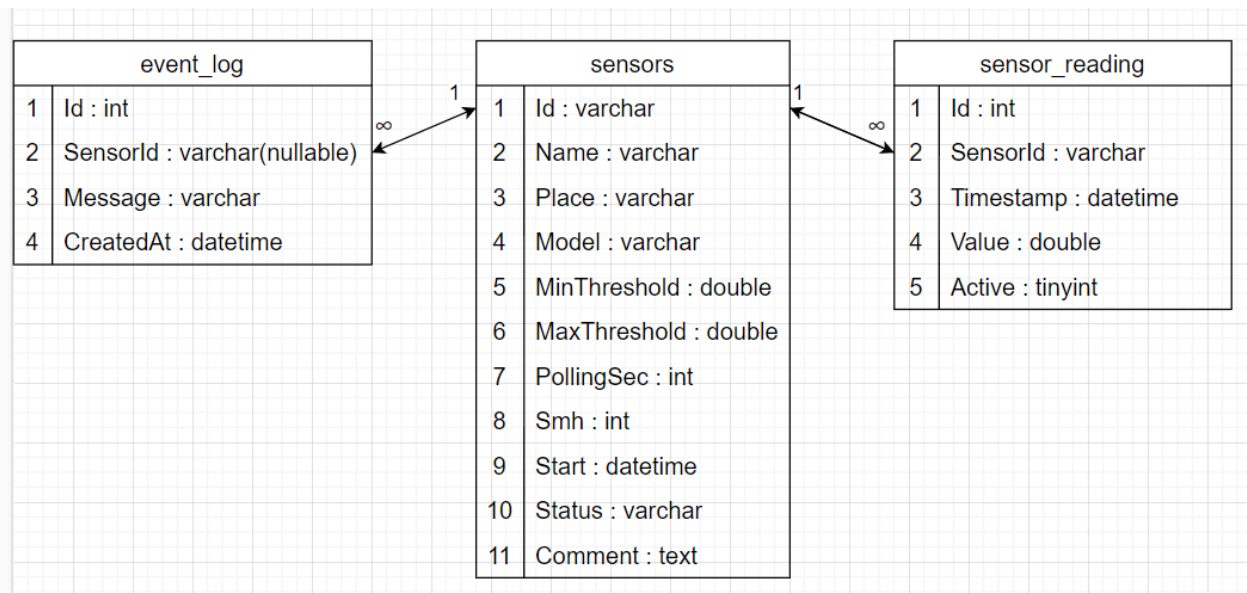


Рисунок 3.6 – Концептуальна модель бази даних

3.3.7 Опис модулів розроблюваної системи

Попередньо у пункті 3.3.3 було визначено основні логічні компоненти, на яких базується архітектура розроблюваного програмного продукту. Дані компоненти формують модульну структуру системи моніторингу радіаційного фону та забезпечують реалізацію основних функціональних можливостей застосунку.

Кожен модуль відповідає за окремий набір функцій, що дозволяє забезпечити чіткий розподіл відповідальностей, підвищити зручність супроводу програмного продукту та спростити його подальше розширення. У таблиці 3.5 наведено опис основних модулів реалізованого програмного продукту із зазначенням їх призначення.

Таблиця 3.5 – Опис модулів реалізованого програмного продукту з визначенням основних функцій

Модуль	Опис
MainForm (UI)	Відповідає за взаємодію з користувачем: ініціалізацію інтерфейсу, відображення показників радіаційного фону в реальному часі, побудову графіків, вибір датчиків, керування режимами моніторингу та перегляд історичних даних.
Sensor Monitoring	Відповідає за асинхронний моніторинг радіаційного фону з використанням датчиків, періодичне зчитування значень, перевірку порогових рівнів та генерацію подій перевищення допустимих значень.
Sensor Management	Забезпечує керування датчиками радіаційного моніторингу: додавання нових датчиків, редагування параметрів існуючих, деактивацію та видалення датчиків із системи.
Event Log	Відповідає за реєстрацію та збереження подій системи, зокрема дій користувача, змін стану датчиків, перевищення порогових значень радіації та системних повідомлень. Забезпечує перегляд журналу подій за обраний період та фільтрацію за датчиками.
Data Access (Repository)	Забезпечує взаємодію з базою даних: збереження, оновлення, видалення та вибірку даних про датчики, результати вимірювань радіаційного фону та записи журналу подій.
Models	Містить класи-моделі даних, які описують структури датчиків, вимірювань та подій. Забезпечує уніфікований формат передавання даних між модулями застосунку.
Sensor Simulation	Відповідає за імітацію роботи реальних радіаційних датчиків шляхом генерації випадкових значень радіаційного фону, що дозволяє тестувати та демонструвати роботу системи без використання фізичних пристроїв.

3.4 Розробка користувацького інтерфейсу

Розробка користувацького інтерфейсу є важливим етапом створення вебзастосунку для керування задачами та нотатками. Саме інтерфейс визначає характер взаємодії користувача з програмою, рівень зручності роботи та загальне враження від використання застосунку. Тому цьому етапу приділяється особлива увага. У межах даного розділу спочатку розглядаються вайрфрейми, після чого інтерфейс детальніше описується за допомогою прототипів.

3.4.1 Базова структура інтерфейсу

Вайрфрейм є схематичним зображенням сторінки або структури вебінтерфейсу, що відображає розташування основних елементів, навігаційних компонентів та принципи їх взаємодії. Як правило, вайрфрейми не містять оформлення у вигляді кольорів, шрифтів чи графічних елементів, оскільки основна увага зосереджена на функціональності, логіці роботи та наповненні інтерфейсу. Таким чином, вайрфрейм показує, які дії виконує екран і як організовано взаємодію з ним, а не його візуальний вигляд.

Вайрфрейми можуть створюватися у вигляді простих ескізів від руки або на дошці, а також за допомогою спеціалізованих безкоштовних чи комерційних програмних засобів.

Головна сторінка програмного продукту (рисунк 3.7) містить основні елементи інтерфейсу, необхідні для відображення, аналізу та керування процесом моніторингу рівня радіаційного фону в режимі реального часу та за вибраний період.

Інтерфейс головного вікна логічно поділений на декілька функціональних областей:

а) верхня панель керування, яка містить:

- 1) кнопку переходу до журналу подій;

2) індикатор поточного значення радіаційного фону для обраного датчика;

3) відображення дати та часу останнього вимірювання;

4) випадаючий список для вибору активного датчика;

5) елемент вибору режиму керування датчиками (додавання, редагування, видалення);

б) ліва інформаційна панель, у якій відображається детальна інформація про вибраний датчик, зокрема:

1) ідентифікатор датчика;

2) назва та місце встановлення;

3) модель;

4) мінімальне та максимальне порогові значення;

5) період опитування;

6) коментар;

7) статус датчика та дата початку роботи;

в) центральна частина вікна, призначена для візуалізації даних у вигляді графіка. Тут відображається графік зміни рівня радіаційного фону в часі для обраного датчика, а також лінії мінімального та максимального допустимих порогів. Графік дозволяє наочно оцінювати динаміку показників та швидко виявляти перевищення нормативних значень;

г) права панель параметрів, яка містить елементи керування для:

1) вибору періоду перегляду даних («за останні хвилини/години/дні» або довільний діапазон);

2) задання початкової та кінцевої дати;

3) налаштування параметрів датчика (назва, місце встановлення, модель, порогові значення, частота опитування, коментар);

4) застосування внесених змін;

г) нижня панель журналу повідомлень, у якій у текстовому вигляді відображаються всі поточні події системи:

- 1) надходження нових вимірювань;
- 2) перевищення порогових значень;
- 3) дії користувача (додавання, редагування, видалення датчиків).

Дана структура інтерфейсу забезпечує зручну навігацію, чіткий розподіл функціональних зон та дозволяє оператору оперативно отримувати інформацію про стан радіаційного фону і реагувати на небезпечні ситуації.

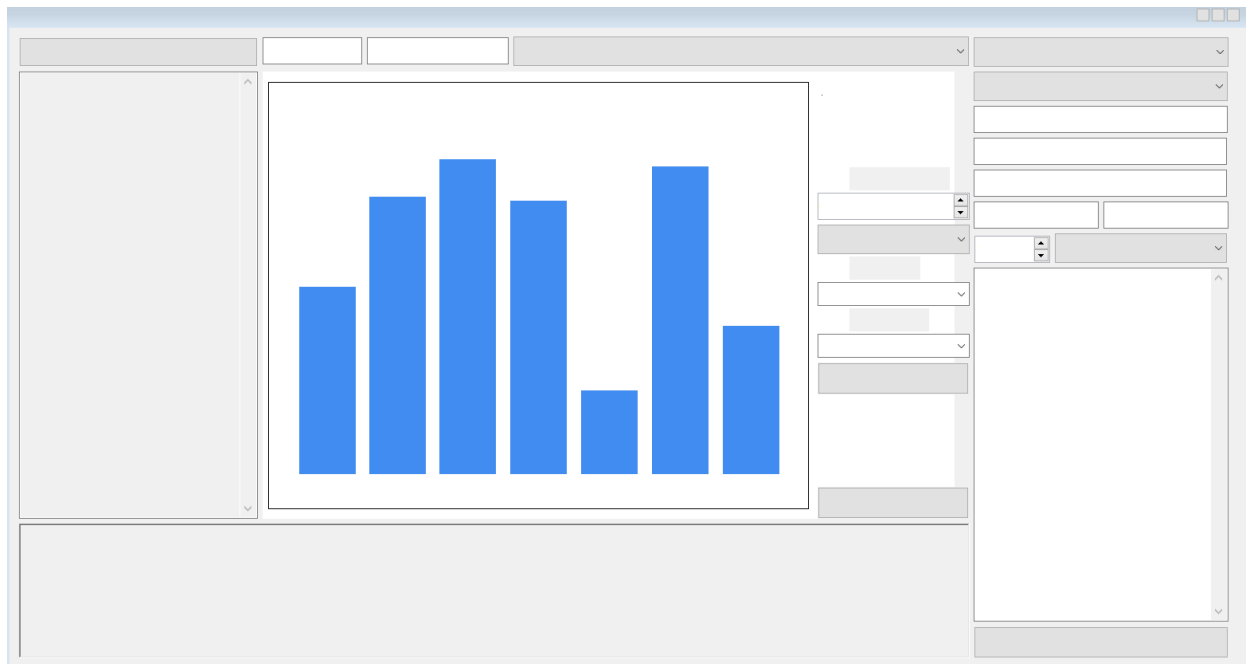


Рисунок 3.7 – Вайрфрейм головного вікна програмного продукту

Окреме вікно журналу подій (рисунок 3.8) призначене для перегляду історії подій системи радіаційного моніторингу за вибраний період часу та з урахуванням вибраних датчиків.

У вікні журналу подій передбачено:

- область виводу текстових повідомлень журналу;
- елементи вибору періоду перегляду (кількість хвилин/годин/днів або довільний інтервал);
- список датчиків з можливістю вибору одного, декількох або всіх датчиків;

- кнопку застосування фільтрів.

Журнал подій дозволяє здійснювати аналіз історичних даних, контролювати перевищення порогових значень та відслідковувати дії користувача в системі.

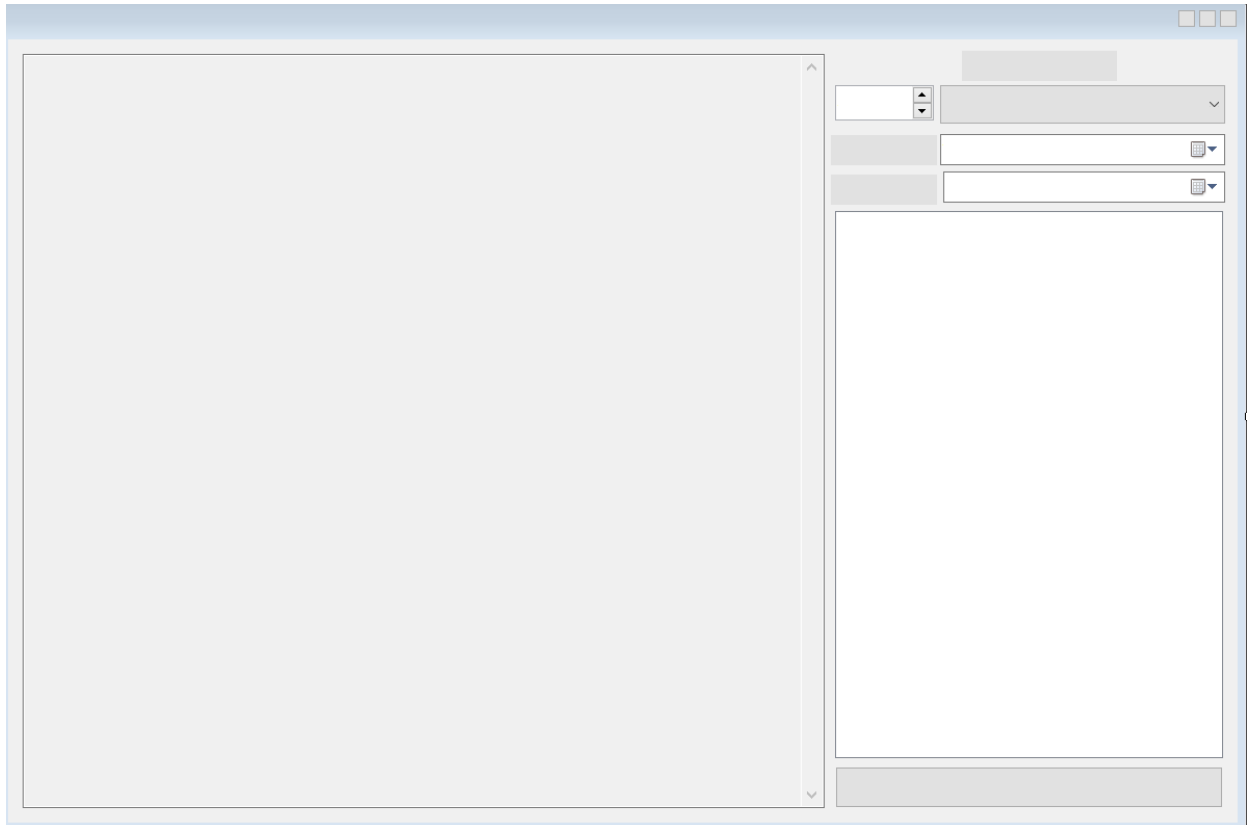


Рисунок 3.8 – Вікно журналу подій

3.4.2 Базова структура інтерфейсу

У пункті 3.4.1 було розглянуто базову структуру інтерфейсу програмного продукту за допомогою вайрфреймів. Було визначено основні типи вікон, що використовуються в системі моніторингу радіаційного фону, а саме:

- головне вікно моніторингу радіаційного фону;
- вікно журналу подій системи.

Розглянемо ці вікна більш детально за допомогою прототипів. Розпочнемо з головного вікна програмного продукту (рис. 3.9) яке

призначене для відображення поточного стану радіаційного фону, керування датчиками та аналізу вимірювань за обраний період.

Порівняно з вайрфреймом, у прототипі головного вікна відображено конкретні елементи інтерфейсу та їх функціональне наповнення, а саме:

- верхню панель, яка містить кнопку відкриття журналу подій, індикатор поточного значення радіаційного фону ($\mu\text{Sv/h}$), дату та час останнього вимірювання, а також випадаючий список вибору активного датчика;
- ліву інформаційну панель, де відображається детальна інформація про обраний датчик (ідентифікатор, назва, місце встановлення, модель, порогові значення, період опитування, статус та час початку роботи);
- центральну область з графіком, який відображає зміну рівня радіаційного фону у часі. На графіку також показані лінії мінімального та максимального порогів;
- праву панель керування, що містить елементи для додавання, редагування та видалення датчиків, а також для вибору періоду перегляду вимірювань (останні хвилини, години, дні або довільний часовий діапазон);
- нижню область журналу повідомлень, де в режимі реального часу відображаються отримані вимірювання та службові повідомлення системи.

Такий підхід забезпечує зручний доступ до всієї ключової інформації та дозволяє оператору оперативно реагувати на зміни радіаційного фону.

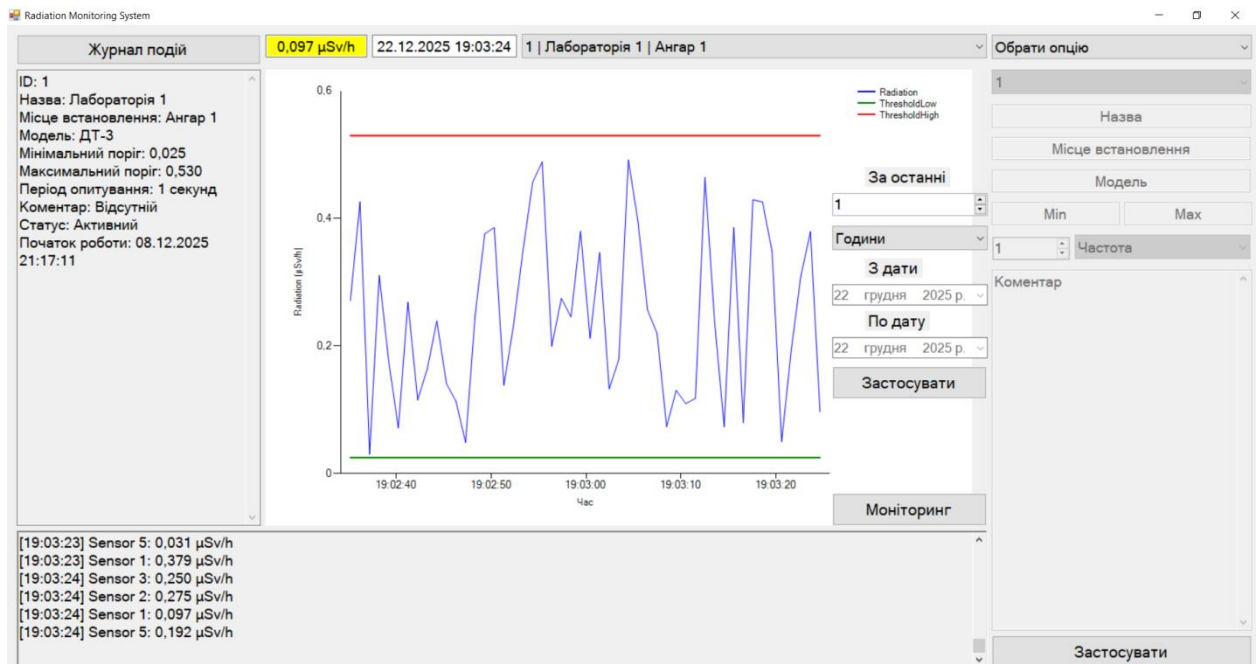


Рисунок 3.9 – Прототип головного вікна системи моніторингу радіаційного фону

Окремим елементом інтерфейсу є вікно журналу подій (рис. 3.10), яке використовується для перегляду історії системних повідомлень та подій, пов'язаних з роботою датчиків.

Порівняно з вайрфреймом, прототип вікна журналу подій містить наступні функціональні елементи:

- центральну текстову область, у якій відображаються записи журналу подій із зазначенням дати, часу, ідентифікатора датчика та тексту повідомлення;
- блок вибору періоду перегляду, який дозволяє відобразити події за останні хвилини, години, дні або за довільний часовий інтервал;
- список датчиків з можливістю множинного вибору, включаючи пункт «Усі», що дозволяє фільтрувати події за конкретними датчиками або переглядати загальні системні події;
- кнопку застосування фільтрів, яка ініціює оновлення списку подій відповідно до обраних параметрів.

Вікно журналу подій забезпечує зручний аналіз історичних даних,

дозволяє швидко виявляти аварійні ситуації та контролювати роботу системи моніторингу в цілому.

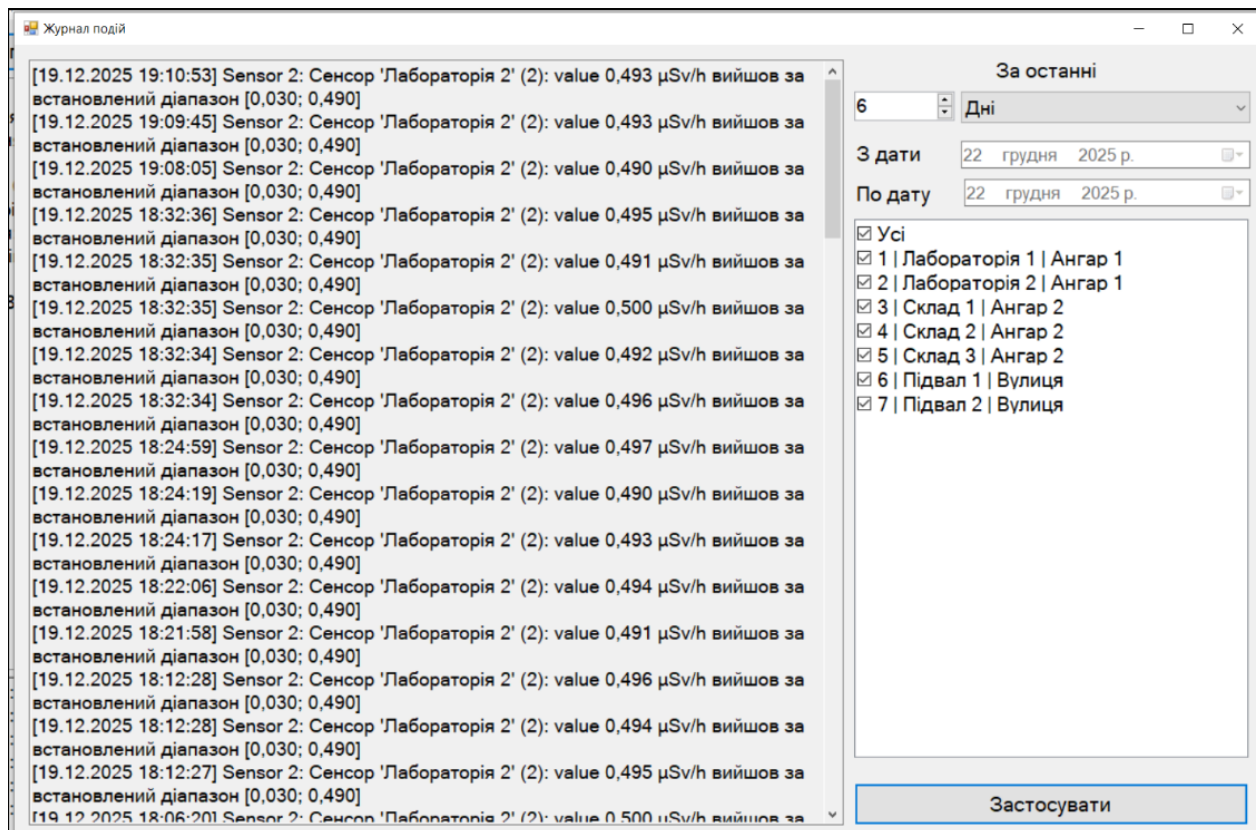


Рисунок 3.10 – Прототип вікна журналу подій

3.5 Висновок

У даному розділі було виконано проектування програмного продукту Radiation Monitoring System, призначеного для моніторингу рівня радіаційного фону з використанням настільного Windows-застосунку. На початковому етапі сформульовано основні функціональні та нефункціональні вимоги до системи, що дозволило чітко визначити її призначення, умови експлуатації та очікувану поведінку в процесі роботи.

Визначено вимоги до технічних і програмних засобів, дотримання яких забезпечує стабільну роботу застосунку, коректну обробку даних у режимі реального часу та зручний користувацький інтерфейс. Розроблено архітектуру програмного продукту на основі багатошарового підходу з

елементами MVP, що забезпечує чітке розмежування відповідальностей між інтерфейсом користувача, логікою моніторингу та доступом до даних.

За допомогою UML-діаграм розгортання, компонентів, прецедентів і станів було наочно відображено структуру системи, взаємодію її основних компонентів, сценарії використання та логіку роботи в різних режимах. Також спроектовано структуру бази даних, яка забезпечує збереження конфігурацій датчиків, результатів вимірювань та журналу подій, що є необхідним для подальшого аналізу та контролю радіаційної обстановки.

Окрему увагу приділено проектуванню користувацького інтерфейсу. Розроблено вайрфрейми та прототипи основних вікон системи, що дозволило визначити логічну структуру інтерфейсу, розташування функціональних елементів і принципи взаємодії користувача із застосунком. Запропоноване інтерфейсне рішення забезпечує зручну навігацію, наочну візуалізацію даних та оперативний доступ до ключових функцій системи.

Таким чином, результати проектування створюють надійну основу для подальшої реалізації програмного продукту, його тестування та впровадження, а також забезпечують можливість розширення функціоналу системи моніторингу радіаційного фону в майбутньому.

4 ІМІТАЦІЯ РОБОТИ РАДІАЦІЙНОГО ДАТЧИКА

4.1 Загальна характеристика імітації радіаційного датчика

Під час розробки програмного продукту для моніторингу радіаційного фону виникає необхідність у постійному надходженні даних вимірювань рівня іонізуючого випромінювання. На етапі проєктування та реалізації програмного забезпечення використання реальних радіаційних датчиків є ускладненим або неможливим через відсутність спеціалізованого обладнання, складність його налаштування, а також обмеження щодо безпеки та доступності.

З цієї причини у розроблюваному програмному продукті було застосовано підхід програмної імітації роботи радіаційного датчика. Імітований датчик дозволяє відтворити процес отримання значень радіаційного фону без підключення фізичних вимірювальних пристроїв, що забезпечує можливість повноцінного тестування та демонстрації функціоналу системи.

Імітація датчика виконує роль джерела вимірювальних даних у системі та забезпечує генерацію значень рівня радіації у реальному часі. Згенеровані показники використовуються для побудови графіків радіаційного фону, перевірки перевищення встановлених порогових значень, формування попереджень, а також для запису результатів вимірювань і подій у базу даних.

Застосування програмної імітації дозволяє перевірити коректність роботи всіх ключових модулів програмного продукту, зокрема:

- сервісу моніторингу радіаційного фону;
- механізму обробки та збереження вимірювань;
- модуля візуалізації даних у вигляді графіків;
- журналу подій, що фіксує перевищення порогових значень та дії користувача.

Важливою особливістю реалізованої імітації є її інтеграція в загальну

архітектуру системи через використання інтерфейсу радіаційного датчика. Такий підхід забезпечує універсальність рішення та дозволяє в майбутньому без суттєвих змін програмного коду замінити імітований датчик на реальний вимірювальний пристрій.

4.2 Обґрунтування вибору програмної імітації радіаційного датчика

Вибір програмної імітації радіаційного датчика в розроблюваному програмному продукті зумовлений низкою технічних, організаційних та практичних чинників. На етапі створення та тестування системи моніторингу радіаційного фону використання реальних вимірювальних пристроїв є недоцільним через складність їх інтеграції, високу вартість обладнання та обмежений доступ до спеціалізованих датчиків.

Реальні радіаційні датчики потребують фізичного підключення, налаштування інтерфейсів обміну даними, калібрування та дотримання вимог радіаційної безпеки. Крім того, у навчальних і демонстраційних умовах використання такого обладнання може бути небезпечним або неможливим. Саме тому програмна імітація дозволяє усунути зазначені обмеження та забезпечити стабільну роботу програмного продукту.

Ще одним важливим аргументом на користь імітації є необхідність безперервного надходження даних для перевірки функціональних можливостей системи. Імітований датчик забезпечує генерацію вимірювань у заданому діапазоні значень та з визначеною періодичністю, що дозволяє моделювати різні сценарії роботи системи, зокрема:

- нормальний рівень радіаційного фону;
- коливання показників у межах допустимих значень;
- перевищення мінімальних або максимальних порогів;
- нестабільні або різко змінні значення вимірювань.

Використання програмної імітації також значно спрощує процес

тестування та налагодження програмного продукту. Розробник має можливість оперативно змінювати параметри генерації даних, відтворювати критичні ситуації та перевіряти реакцію системи на аварійні події без ризику для здоров'я або навколишнього середовища.

Крім того, реалізація імітованого датчика через інтерфейс дозволяє дотримуватись принципів об'єктно-орієнтованого програмування, зокрема принципу інверсії залежностей. Завдяки цьому архітектура системи залишається гнучкою та розширюваною, а заміна імітованого датчика на реальний у майбутньому не потребуватиме суттєвих змін у програмному коді.

4.3 Архітектурний підхід до реалізації датчика

Під час проєктування програмного продукту для моніторингу радіаційного фону одним із ключових завдань було забезпечення гнучкої та розширюваної архітектури, яка дозволяє працювати з різними джерелами радіаційних вимірювань. Для досягнення цієї мети в системі застосовано інтерфейсний підхід, зокрема використання інтерфейсу `IRadiationSensor` як базового контракту для всіх типів датчиків.

Інтерфейс `IRadiationSensor` визначає єдиний набір методів і властивостей, яким повинні відповідати всі реалізації датчиків у системі – як імітовані, так і реальні фізичні пристрої. Такий підхід дозволяє відокремити логіку отримання вимірювань від логіки їх подальшої обробки, аналізу та візуалізації, що відповідає принципам об'єктно-орієнтованого програмування та інверсії залежностей.

Архітектурні взаємозв'язки між основними компонентами підсистеми датчика, зокрема інтерфейсом `IRadiationSensor`, його імітованою реалізацією `FakeRadiationSensor`, сервісом моніторингу `SensorMonitoringService`, а також моделями `SensorConfig` і `SensorReading`, наочно представлені на UML-діаграмі класів (рис. 4.1).

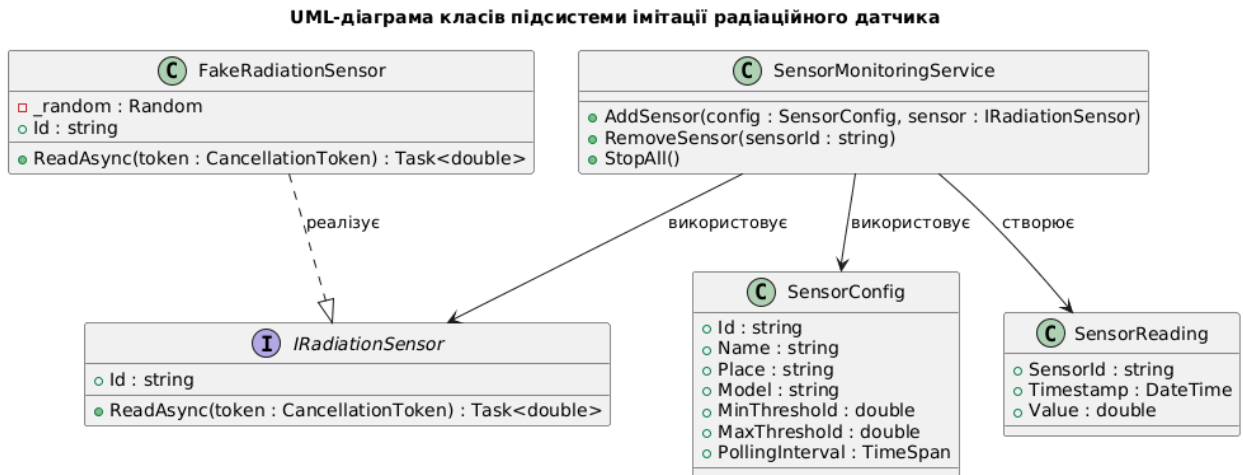


Рисунок 4.1 – UML-діаграма класів підсистеми імітації та моніторингу
радіаційного датчика

Застосування інтерфейсу **IRadiationSensor** забезпечує уніфікований доступ до джерел вимірювань. Незалежно від того, чи дані надходять від імітованого програмного датчика, чи від реального апаратного пристрою, сервіс моніторингу взаємодіє з ними однаковим чином. Це спрощує внутрішню логіку системи та зменшує кількість жорстких залежностей між її компонентами.

Використання інтерфейсного підходу також забезпечує можливість легкої заміни імітованого датчика на реальний. Для цього достатньо реалізувати інтерфейс **IRadiationSensor** у новому класі, який здійснюватиме зчитування показників із фізичного датчика або зовнішнього джерела даних. При цьому інші компоненти системи, зокрема сервіс моніторингу, модуль збереження даних та інтерфейс користувача, не потребуватимуть змін.

Крім того, застосований архітектурний підхід забезпечує високу розширюваність системи. У майбутньому можливе впровадження додаткових типів датчиків, наприклад:

- датчиків з різною точністю вимірювання;
- датчиків з альтернативними одиницями виміру;
- датчиків, що отримують дані з мережевих сервісів або API.

Усі такі реалізації можуть бути інтегровані в систему без зміни існуючої бізнес-логіки, що є важливою перевагою з точки зору супроводу та масштабування програмного продукту.

Інтерфейс `IRadiationSensor` містить ключові елементи, необхідні для роботи з датчиком. Властивість `Id` забезпечує унікальну ідентифікацію датчика та використовується для зв'язування вимірювань із відповідною конфігурацією та записами в базі даних. Асинхронний метод `ReadAsync` призначений для отримання поточного значення рівня радіаційного фону, а підтримка механізму скасування виконання дозволяє коректно зупиняти процес зчитування у разі завершення моніторингу або видалення датчика.

4.4 Принцип роботи імітованого радіаційного датчика

Імітований радіаційний датчик у розроблюваному програмному продукті призначений для відтворення процесу вимірювання рівня радіаційного фону в умовах відсутності реального вимірювального обладнання. Його робота базується на програмній генерації значень радіації з урахуванням заданих параметрів та подальшій передачі цих даних у систему моніторингу.

Основою взаємодії імітованого датчика з іншими компонентами системи є інтерфейс `IRadiationSensor`, який визначає єдиний контракт для отримання вимірювань. Наявність такого інтерфейсу дозволяє уніфікувати роботу з датчиками незалежно від їх типу – імітованими або реальними – та забезпечує гнучкість архітектури програмного продукту.

Принцип роботи імітованого датчика полягає у наступному:

- для кожного датчика створюється окремий об'єкт імітації, який має унікальний ідентифікатор;
- зчитування показників здійснюється асинхронно з використанням методу `ReadAsync`, що дозволяє не блокувати основний потік виконання програми;

- під час кожного циклу зчитування генерується випадкове числове значення рівня радіації у заданому діапазоні (наприклад, від 0.03 до 0.50 $\mu\text{Sv/h}$);
- згенероване значення передається до сервісу моніторингу, де воно обробляється, зберігається в базі даних та відображається в інтерфейсі користувача.

Періодичність зчитування значень визначається параметрами конфігурації датчика, зокрема інтервалом опитування. Завдяки цьому система може імітувати різні режими роботи датчиків – від частого моніторингу до більш рідкісного зчитування показників.

Важливою особливістю реалізації є використання механізму скасування виконання (CancellationToken). Це дозволяє коректно зупинити роботу імітованого датчика у разі вимкнення моніторингу, редагування параметрів датчика або його видалення із системи. Такий підхід запобігає витокам ресурсів та забезпечує стабільність роботи програмного продукту.

Отримані значення рівня радіації додатково перевіряються на відповідність мінімальним та максимальним пороговим значенням. У випадку перевищення допустимого рівня система формує відповідне попередження, яке відображається користувачу та фіксується у журналі подій. Це дозволяє повністю відтворити логіку роботи реальної системи радіаційного контролю.

4.5 Реалізація імітації радіаційного датчика в програмному коді

Реалізація імітації радіаційного датчика у програмному продукті виконана з використанням об'єктно-орієнтованого підходу та асинхронних механізмів мови програмування C#. Основною метою такої реалізації є створення програмного компонента, який максимально наближено відтворює поведінку реального датчика радіаційного фону та може бути безпосередньо інтегрований у загальну систему моніторингу без зміни її архітектури.

Для реалізації імітованого датчика використовується окремий клас `FakeRadiationSensor`, який реалізує інтерфейс `IRadiationSensor`. Застосування інтерфейсного підходу дозволяє чітко відокремити логіку отримання вимірювань від логіки їх подальшої обробки, збереження та візуалізації, а також забезпечує можливість у майбутньому замінити програмну імітацію на реальний апаратний датчик без модифікації інших модулів програмного продукту.

Клас імітованого датчика містить унікальний ідентифікатор, який відповідає ідентифікатору датчика в системі. Це дозволяє коректно пов'язувати отримані значення з відповідною конфігурацією датчика, історією вимірювань та записами у базі даних. Ідентифікація датчика є критично важливою для правильної роботи механізмів моніторингу, фільтрації даних та формування журналу подій.

Основна функціональність імітованого датчика реалізована в асинхронному методі зчитування показників. Під час виклику цього методу відбувається генерація випадкового числового значення рівня радіації у заданому діапазоні. Межі діапазону підібрані таким чином, щоб імітувати як нормальний радіаційний фон, так і потенційно небезпечні значення, які можуть призвести до спрацювання порогових механізмів системи.

Асинхронний характер реалізації забезпечує:

- паралельну роботу кількох датчиків;
- відсутність блокування інтерфейсу користувача;
- стабільну роботу програмного продукту навіть при значній кількості активних датчиків.

Загальний процес взаємодії між імітованим датчиком, сервісом моніторингу, інтерфейсом користувача та модулем збереження даних проілюстровано на рисунку 4.2, де наведено UML-діаграму послідовності. Діаграма відображає повний цикл обробки вимірювання: від виклику методу зчитування значення датчиком, формування об'єкта вимірювання, перевірки порогових значень, відображення результатів у графічному інтерфейсі до

збереження даних у базі та фіксації подій у журналі.

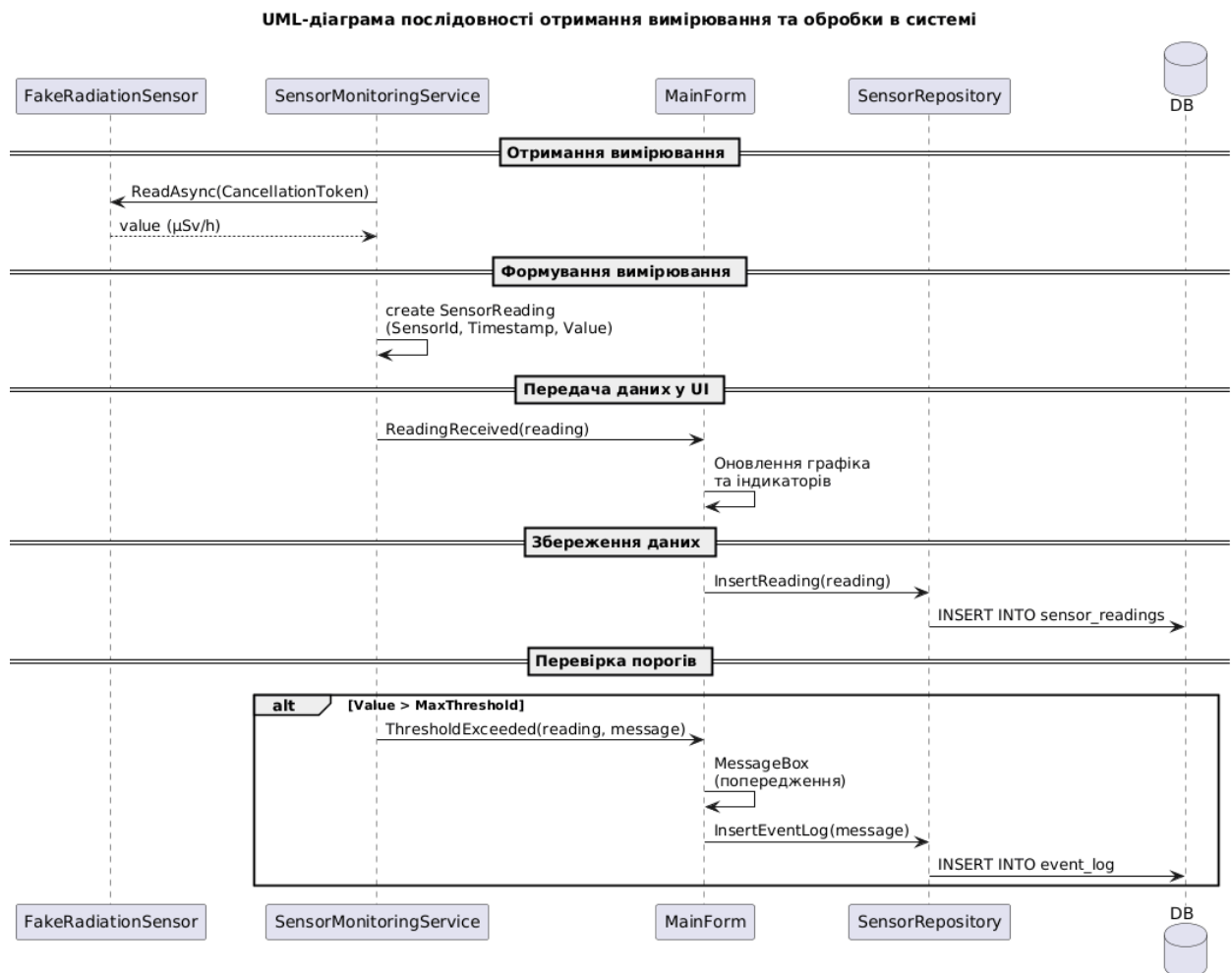


Рисунок 4.2 – UML-діаграма послідовності отримання вимірювання та обробки в системі

Після генерації значення рівня радіації результат передається до сервісу моніторингу, де формується об'єкт вимірювання. Далі отримані дані перевіряються на відповідність заданим пороговим значенням, зберігаються у базі даних та використовуються для побудови графіків і формування повідомлень про можливі перевищення допустимих норм.

Важливою складовою реалізації є підтримка керування життєвим циклом датчика. У разі зупинки моніторингу, редагування параметрів датчика або його видалення асинхронний процес опитування коректно завершується. Це досягається за рахунок використання токенів скасування

виконання, що дозволяє ефективно контролювати фонові процеси та запобігати перевантаженню системи.

4.6 Особливості тестування та демонстрації роботи системи

Імітація радіаційного датчика відіграє ключову роль у процесі тестування та демонстрації розробленого програмного продукту. Завдяки використанню програмного датчика стає можливим повноцінне тестування всіх функціональних можливостей системи без необхідності підключення реального вимірювального обладнання. Це значно спрощує процес налагодження, перевірки працездатності та презентації програмного продукту.

Одним із основних напрямів використання імітованого датчика є тестування роботи користувацького інтерфейсу. Згенеровані значення радіаційного фону дозволяють перевірити коректність відображення показників у режимі реального часу, правильність побудови графіків, оновлення текстових індикаторів та відображення інформації про обраний датчик. Таким чином забезпечується візуальна перевірка роботи інтерфейсу за різних сценаріїв використання.

Імітація датчика також застосовується для перевірки механізмів моніторингу в реальному часі. Завдяки періодичній генерації нових вимірювань система безперервно отримує дані, що дозволяє протестувати асинхронну обробку показників, коректну роботу таймерів і сервісу моніторингу, а також реакцію системи на перевищення встановлених порогових значень радіації. У процесі тестування перевіряється формування попереджувальних повідомлень і запис відповідних подій у журнал подій.

Важливим аспектом є перевірка коректності роботи фільтрів перегляду даних за період. Імітовані вимірювання зберігаються в базі даних із часовими мітками, що дозволяє тестувати вибірку показників за хвилини, години, дні або довільний часовий інтервал. Це дає змогу перевірити

правильність формування історичних графіків та відповідність відображених даних обраному часовому діапазону.

Окрему увагу приділено можливості демонстрації роботи системи без фізичного обладнання. Імітаційний датчик дозволяє наочно продемонструвати функціональні можливості програмного продукту під час захисту роботи або презентації, незалежно від наявності реальних вимірювальних пристроїв. Це забезпечує стабільну та передбачувану поведінку системи, що особливо важливо в навчальних та демонстраційних цілях.

4.7 Висновок

У даному розділі було розглянуто підхід до імітації роботи радіаційного датчика в межах розробленого програмного продукту моніторингу радіаційного фону. Обґрунтовано доцільність використання програмної імітації як ефективного засобу для тестування, налагодження та демонстрації функціональних можливостей системи без необхідності застосування фізичного вимірювального обладнання.

Було описано принципи роботи імітованого датчика, який забезпечує генерацію значень радіаційного фону в заданому діапазоні та імітує поведінку реального пристрою. Застосування архітектурного підходу на основі інтерфейсу IRadiationSensor дозволило уніфікувати доступ до джерел вимірювань, забезпечити модульність системи та створити умови для легкої заміни програмного датчика на реальний у майбутньому без внесення змін до бізнес-логіки застосунку.

Окрему увагу було приділено використанню імітації датчика під час тестування користувацького інтерфейсу, механізмів моніторингу в реальному часі, роботи з історичними даними та формування журналу подій. Такий підхід дозволив перевірити коректність функціонування основних компонентів системи, зокрема асинхронної обробки вимірювань, побудови

графіків і механізмів сповіщення про перевищення порогових значень.

У результаті впровадження імітаційного датчика було досягнуто підвищення гнучкості та надійності розробленого програмного продукту. Запропоноване рішення спрощує процес розробки, налагодження та презентації системи, а також створює передумови для подальшого розширення функціоналу і інтеграції з реальними апаратними засобами вимірювання радіаційного фону.

5 КЕРІВНИЦТВО ПРОГРАМІСТА

5.1 Призначення й умови використання

Розроблюваний програмний продукт призначений для моніторингу рівня радіаційного фону в навколишньому середовищі. Програмний продукт реалізований у вигляді окремого віконного застосунку, який надає можливість переглядати поточні значення радіаційного фону, здійснювати моніторинг показників у реальному часі, аналізувати історичні дані вимірювань за обраний період, а також фіксувати та переглядати події, пов'язані з перевищенням встановлених порогових значень. Крім того, застосунок забезпечує керування радіаційними датчиками, зокрема їх додавання, редагування, видалення та вибір для подальшого моніторингу.

Для коректної роботи ПП необхідні такі мінімальні системні вимоги:

- платформа x64-based;
- операційна система Microsoft Windows 11/10/8/7 (рекомендується встановити всі останні оновлення);
- процесор з мінімальною тактовою частотою 1,8 ГГц, рекомендується використовувати як мінімум двоядерний процесор;
- оперативна пам'ять від 2 ГБ (рекомендується 4 ГБ та мінімум 2,5ГБ при використанні на віртуальній машині);
- роздільна здатність екрану 1920x1080 пікселів з 16-бітовим кольором;
- мінімум 2 ГБ вільного місця на жорсткому диску.

5.2 Звертання до програмного продукту (файли проєкту)

Розроблюваний ПП складається з таких модулів:

MainForm.cs – основний модуль, який відповідає за ініціалізацію програми, елементів інтерфейсу, запуск/зупинку моніторингу, відображення вимірювань радіації на графіку, керування датчиками та перегляд даних за

вибраний період.

Опис структури модулю MainForm.cs:

- MainForm – конструктор головної форми: розгортає вікно на весь екран та викликає ініціалізацію компонентів інтерфейсу;
- Form1_Load – стартова ініціалізація: створює репозиторій і сервіс моніторингу, підписується на події отримання показників і перевищення порогів, ініціалізує елементи UI (списки періодів/частот/опцій), будує графік, завантажує датчики з БД, запускає симуляцію для кожного датчика, заповнює комбобокси та відображає останні 50 записів для першого датчика;
- OnReadingReceived – обробник події надходження нового вимірювання: виводить повідомлення в журнал (RichTextBox), записує показник у БД, оновлює “ковзне вікно” з останніх 50 значень для поточного датчика та (якщо режим monitoring увімкнено) перемальовує графік і оновлює індикатори стану;
- OnThresholdExceeded – обробник події перевищення порогу: додає червоне повідомлення в журнал, показує попереджувальне вікно та записує подію в event_log;
- InitChart – підготовка графіка: створює ChartArea та серії Radiation, ThresholdLow, ThresholdHigh, налаштовує осі часу та сітки, готує графік до відображення даних;
- UpdateThresholdLines – оновлює порогові лінії (зелена/червона) на графіку для поточного датчика на проміжку часу від xMin до xMax;
- DrawArray – відмальовка графіка за списком вимірювань: очищує серію, додає точки (X=Timestamp, Y=Value), після чого підлаштовує порогові лінії під часовий діапазон даних;
- AppendColoredText – службовий метод форматowanego додавання тексту в RichTextBox із заданим кольором і автопрокруткою вниз;
- cmbPeriodType_SelectedIndexChanged – керування логікою вибору періоду: при “Довільний діапазон” активує DateTimePicker-и та блокує NumericUpDown, в інших режимах – навпаки;

- `Form1_FormClosing` – завершення роботи: записує подію про закриття програми в `event_log` та зупиняє всі сенсори в сервісі моніторингу;
- `btnSensorApply_Click` – обробник кнопки “Застосувати” для керування датчиками:
 - 1) додати – формує новий `SensorConfig`, підбирає наступний вільний `Id`, зберігає в БД, додає в симуляцію і в локальний список, оновлює комбобокси та пише подію в `event_log`;
 - 2) редагувати – оновлює параметри обраного датчика (крім `Id`), зберігає в БД, перезапускає симуляцію датчика, логгує подію;
 - 3) видалити – деактивує показники (в `readings`), видаляє датчик із `sensors`, прибирає з локального списку, оновлює UI та логгує подію;
- `GetNextSensorId` – знаходить найменший вільний числовий `Id` датчика (заповнює “дірки” 1..N);
- `UpdateSensorComboBox` – перевстановлює `DataSource` для `cmbSensors` та `cmbSensorId` (відсортовано за `Id`), щоб відображати датчики у форматі “`Id | Name | Place`”;
- `LoadSensorToEditForm` – завантажує дані датчика у поля редагування та коректно відображає частоту опитування з урахуванням `Smh` (сек/хв/год);
- `cmbSensorId_SelectedIndexChanged` – при режимі “Редагувати” підвантажує у форму вводу параметри вибраного датчика (або першого, якщо вибору ще нема);
- `cmbSensors_SelectedIndexChanged` – при виборі датчика для перегляду: переходить у `monitoring=true`, завантажує останні 50 вимірювань цього датчика та оновлює інформаційний блок `AboutSensor`;
- `LoadHistoryToBuffer` – зчитує останні 50 показників із БД для заданого датчика, записує у `_currentReadings` та відмальовує на графіку;
- `btnMonitoring_Click` – вмикає режим реального часу (`monitoring=true`) і перезавантажує останні 50 значень для вибраного датчика,

щоб графік знову оновлювався “наживо”;

- `btnEventLog_Click` – відкриває форму журналу подій (`FormEventLog`) в одному екземплярі: якщо форма вже відкрита – активує її, якщо закрита – створює заново;

- `btnApplyPeriod_Click` – режим перегляду даних за період (`monitoring=false`): визначає межі часу за обраним типом (хв/год/дні або довільний діапазон), зчитує показники з БД за період і відображає їх на графіку з урахуванням порогів поточного датчика.

`FormEventLog.cs` – модуль форми журналу подій, відповідає за завантаження списку датчиків, вибір періоду перегляду, фільтрацію подій за датчиками та відображення записів із таблиці `event_log` у текстовому вигляді.

Опис структури модулю `FormEventLog.cs`:

- `FormEventLog` – конструктор форми, ініціалізує компоненти інтерфейсу та створює об’єкт для роботи з базою даних;

- `FormEventLog_Load` – метод початкового завантаження форми: зчитує датчики з БД, заповнює `checkedListBox1`, налаштовує елементи вибору періоду (`NumericUpDown`, `ComboBox`, `DateTimePicker`) та встановлює значення за замовчуванням;

- `LoadSensorsFromDb` – метод зчитує всі датчики з БД, сортує їх за `Id` та записує у локальний список;

- `FillCheckedListBox` – метод заповнює `checkedListBox1` пунктом «Усі» та списком датчиків у форматі `Id | Name | Place`;

- `button1_Click` – метод застосування фільтрів: визначає часовий інтервал (хв/год/дні або довільний діапазон), враховує вибрані датчики та зчитує відповідні записи з таблиці `event_log`, після чого виводить їх у `textBox1`;

- `checkedListBox1_ItemCheck` – метод керування логікою вибору датчиків: при виборі «Усі» автоматично відмічає/знімає всі датчики; при виборі/знятті окремих датчиків коректно оновлює позначку «Усі»;

- `cmbPeriodType_SelectedIndexChanged` – метод обробки зміни

типу періоду, викликає налаштування доступності елементів вводу;

- `ApplyPeriodUiState` – метод керує станом UI: при «Довільний діапазон» активує `dtFrom/dtTo` і блокує `numPeriodValue`, в інших режимах – навпаки.

`ForDesign.cs` є допоміжним модулем основної форми програмного продукту. Він відповідає за керування елементами користувацького інтерфейсу, перевірку коректності введених даних під час додавання, редагування та видалення датчиків радіаційного моніторингу, а також за формування текстового опису вибраного датчика.

Опис структури модуля `ForDesign.cs`:

- `ClearOption` – метод призначений для скидання елементів керування форми до початкового стану. Використовується після виконання операцій додавання, редагування або видалення датчика. Забезпечує очищення текстових полів, встановлення значень за замовчуванням та коректне блокування або розблокування елементів інтерфейсу;

- `ValidateSensorForm` – метод виконує загальну перевірку обраної користувачем операції (додавання, редагування або видалення датчика). На основі вибраної опції визначає подальший сценарій обробки даних та викликає додаткові методи перевірки. Повертає числове значення, яке використовується для керування логікою виконання операції;

- `ValidateAddEdit` – метод призначений для перевірки коректності введених даних під час додавання або редагування датчика радіаційного моніторингу і перевіряє:

- 1) заповненість обов'язкових полів (назва, місце встановлення, модель);
- 2) коректність числових значень мінімального та максимального порогів радіації;
- 3) логічну узгодженість порогових значень (мінімальне значення повинно бути меншим за максимальне);
- 4) вибір періоду опитування датчика (секунди, хвилини або

години);

- `AboutSensor` – метод формує текстовий опис обраного датчика радіаційного моніторингу. Інформація використовується для відображення у відповідному текстовому полі інтерфейсу. Метод враховує параметр періоду опитування (smh) і автоматично перетворює його у зручний для користувача формат (секунди, хвилини або години).

`EventLogEntry.cs` – допоміжний модуль (клас-модель), який описує структуру одного запису журналу подій (event log) та використовується для передачі/зберігання даних між базою даних і програмою.

Опис структури модуля `EventLogEntry.cs`:

- `EventLogEntry` – клас сутності (модель даних) для представлення одного запису журналу подій і містить такі поля:
 - `Id` – унікальний ідентифікатор запису (як правило, автоінкремент у БД);
 - `SensorId` – ідентифікатор датчика, до якого відноситься подія; може бути null для загальних системних подій (наприклад, запуск/закриття програми);
 - `Message` – текст повідомлення/опис події (наприклад, “датчик додано”, “перевищено поріг”);
 - `Timestamp` – дата та час фіксації події.

`FakeRadiationSensor.cs` призначений для імітації роботи реального радіаційного датчика. Він використовується у програмному продукті для тестування та демонстрації процесу моніторингу радіаційного фону без необхідності підключення фізичних вимірювальних пристроїв.

Модуль реалізує інтерфейс `IRadiationSensor` та забезпечує генерацію випадкових значень рівня радіації у заданому діапазоні.

Опис структури модуля `FakeRadiationSensor.cs`:

- `FakeRadiationSensor` – конструктор класу, який ініціалізує ідентифікатор датчика. Ідентифікатор використовується для однозначного розпізнавання датчика в системі моніторингу;

- `ReadAsync` – асинхронний метод, призначений для зчитування значення радіаційного фону.

Метод імітує роботу реального радіаційного датчика шляхом генерації випадкового числового значення в діапазоні від 0.03 до 0.50 $\mu\text{Sv/h}$. Отримане значення повертається у вигляді результату асинхронного завдання та використовується іншими модулями системи для:

- побудови графіків радіаційного фону;
- порівняння з мінімальними та максимальними допустимими порогоми;
- формування подій перевищення допустимого рівня радіації;
- збереження результатів вимірювань у базі даних.

`IRadiationSensor.cs` визначає інтерфейс радіаційного датчика, який використовується в програмному продукті для уніфікації роботи з різними джерелами радіаційних вимірювань. Інтерфейс задає базовий контракт, якого повинні дотримуватись усі реалізації датчиків (реальні або імітаційні).

Опис структури модулю `IRadiationSensor.cs`:

- `Id` – властивість, що містить унікальний ідентифікатор радіаційного датчика. Використовується для зв'язування датчика з конфігурацією сенсора та записами в базі даних;
- `ReadAsync` – асинхронний метод, призначений для отримання поточного значення рівня радіації з датчика. Метод підтримує механізм скасування виконання за допомогою токена `CancellationToken`, що дозволяє коректно зупинити процес зчитування під час вимкнення моніторингу або видалення датчика.

`SensorConfig.cs` призначений для опису конфігурації радіаційного датчика та зберігання його основних параметрів. Даний модуль використовується для передавання налаштувань датчика між різними частинами програмного продукту, а також для збереження та відображення інформації про датчики у базі даних і користувацькому інтерфейсі.

Опис структури модуля `SensorConfig.cs`:

- Id – унікальний ідентифікатор датчика, який використовується для ідентифікації датчика в системі моніторингу та базі даних;
- Name – назва датчика, що відображається користувачеві в інтерфейсі програми;
- Place – місце встановлення датчика (наприклад, приміщення або зона контролю);
- Model – модель радіаційного датчика;
- MinThreshold – мінімальне порогове значення рівня радіації, при якому спрацьовує нижній поріг контролю;
- MaxThreshold – максимальне порогове значення рівня радіації, при перевищенні якого формується аварійне повідомлення;
- PollingInterval – інтервал опитування датчика, що визначає періодичність отримання нових значень рівня радіації;
- Smh – параметр, що визначає одиницю виміру періоду опитування (секунди, хвилини або години);
- Comment – текстовий коментар до датчика, який може містити додаткову інформацію;
- Status – поточний стан датчика (активний, неактивний тощо);
- LastValue – останнє зафіксоване значення рівня радіації для даного датчика;
- LastUpdate – дата та час останнього оновлення показників датчика;
- Start – дата та час початку роботи датчика в системі;
- ToString – перевизначений метод, який формує текстове представлення датчика у вигляді рядка, що містить ідентифікатор, назву та місце встановлення датчика. Даний метод використовується для коректного відображення датчиків у списках та випадających елементах інтерфейсу.

SensorMonitoringService.cs відповідає за організацію та керування процесом моніторингу рівня радіаційного випромінювання з використанням датчиків. Даний модуль реалізує асинхронне опитування датчиків, обробку

отриманих значень та перевірку їх на відповідність заданим пороговим значенням.

Опис структури модуля `SensorMonitoringService.cs`:

- `AddSensor` – метод додає новий датчик до сервісу моніторингу. Під час додавання перевіряється відповідність ідентифікатора датчика конфігурації (`SensorConfig.Id`) та самого датчика (`IRadiationSensor.Id`). Для кожного датчика створюється окремий асинхронний процес моніторингу з використанням `CancellationTokenSource`.

- `RemoveSensor` – метод видаляє датчик із сервісу моніторингу за його ідентифікатором. Видалення відбувається шляхом зупинки асинхронного процесу опитування датчика та видалення його з внутрішньої колекції.

- `StopAll` – метод повністю зупиняє моніторинг усіх датчиків. Для кожного активного датчика скасовується токен виконання, після чого внутрішній список датчиків очищується.

- `MonitorSensorAsync` – приватний асинхронний метод, що виконує безперервний моніторинг конкретного датчика. Метод циклічно зчитує значення рівня радіації з датчика з заданим інтервалом опитування, формує об'єкт `SensorReading` та викликає подію `ReadingReceived`. У випадку перевищення максимального порогового значення викликається подія `ThresholdExceeded` із відповідним попереджувальним повідомленням.

- `ReadingReceived` – подія, яка спрацьовує при отриманні нового значення рівня радіації від будь-якого датчика. Використовується для передачі даних до інтерфейсу користувача та подальшого збереження в базі даних.

- `ThresholdExceeded` – подія, яка спрацьовує при перевищенні допустимого рівня радіації. Подія передає інформацію про вимірювання та текстове повідомлення для формування попереджень і запису в журнал подій.

`SensorReading.cs` призначений для опису структури одного

вимірювання радіаційного фону, отриманого від датчика. Даний модуль використовується як базова модель даних для збереження, обробки та візуалізації результатів радіаційних вимірювань у програмному продукті.

Опис структури модуля `SensorReading.cs`:

- `SensorId` – ідентифікатор датчика, з якого було отримано вимірювання. Використовується для зв'язку показника з конкретним датчиком радіації;
- `Timestamp` – дата та час отримання вимірювання. Дозволяє відслідковувати динаміку змін радіаційного фону у часі та здійснювати аналіз за обрані періоди;
- `Value` – числове значення рівня радіації, виражене в мікрозівертах на годину ($\mu\text{Sv/h}$). Використовується для контролю допустимих порогів, побудови графіків та формування повідомлень про перевищення норм.

Модуль `SensorReading.cs` є універсальним контейнером даних і застосовується:

- під час реального моніторингу радіації;
- при збереженні результатів вимірювань у базі даних;
- для побудови графіків радіаційного фону;
- при формуванні журналу подій та сповіщень.

`SensorRepository.cs` відповідає за взаємодію програмного продукту з базою даних. Він реалізує доступ до таблиць датчиків, зчитувань радіації та журналу подій, а також забезпечує збереження, оновлення, видалення та вибірку даних, необхідних для роботи системи моніторингу радіаційного фону.

Опис структури модуля `SensorRepository.cs`:

- `InsertSensor` – метод призначений для додавання нового датчика радіаційного моніторингу до бази даних. Додавання відбувається шляхом створення нового запису в таблиці `sensors` з усіма параметрами датчика (ідентифікатор, місце встановлення, модель, порогові значення, частота

опитування тощо).

- GetAllSensors – метод призначений для отримання списку всіх датчиків радіації з бази даних. Використовується під час ініціалізації програми, формування списків датчиків у графічному інтерфейсі та для подальшої роботи з ними.
- GetSensorById – метод здійснює пошук датчика в базі даних за унікальним ідентифікатором Id та повертає повну конфігурацію обраного датчика.
- UpdateSensor – метод призначений для редагування параметрів існуючого датчика радіаційного моніторингу. Оновлення відбувається шляхом зміни відповідного запису в таблиці sensors.
- DeleteSensor – метод видаляє датчик з таблиці sensors за його ідентифікатором. Застосовується при повному видаленні датчика з системи.
- InsertReading – метод призначений для запису результатів вимірювання радіаційного фону в базу даних. Кожне зчитування містить ідентифікатор датчика, дату та час вимірювання, а також значення рівня радіації.
- GetLastReadings – метод здійснює отримання останніх n активних зчитувань радіації для конкретного датчика. Використовується для побудови графіків у реальному часі та під час переходу в режим моніторингу.
- GetReadingsByPeriod – метод призначений для вибірки зчитувань радіації за заданий період часу для конкретного датчика. Використовується при аналізі історичних даних та побудові графіків за вибраний часовий інтервал.
- DeactivateSensorReadings – метод виконує логічне видалення зчитувань датчика шляхом встановлення ознаки Active = false. Це дозволяє зберігати історію вимірювань без змішування даних старих і нових датчиків з однаковими ідентифікаторами.
- InsertEventLog – метод призначений для запису подій у журнал подій системи. До журналу записуються повідомлення про дії користувача,

зміни стану датчиків, помилки та попередження, пов'язані з перевищенням порогових значень радіації.

– `GetEventLogByPeriod` – метод здійснює отримання записів журналу подій за заданий період часу з урахуванням обраних датчиків. Підтримує фільтрацію подій за конкретними датчиками або відображення загальних системних подій без прив'язки до датчика.

5.3 Вхідні і вихідні дані

5.3.1 Вхідні дані

Вхідними даними програмного продукту для моніторингу радіаційного фону є сукупність параметрів, що надходять від користувача, з бази даних та від імітованих радіаційних датчиків.

Дані від радіаційних датчиків (емуляція). До вхідних даних належать результати вимірювання рівня іонізуючого випромінювання, які генеруються радіаційними датчиками (або їх програмною імітацією):

- `sensorId` – ідентифікатор датчика;
- `value` – значення рівня радіаційного фону ($\mu\text{Sv/h}$);
- `timestamp` – дата та час зчитування показника.

Ці дані використовуються для оперативного моніторингу радіаційного фону, побудови графіків та виявлення перевищень допустимих порогових значень.

На рисунку 5.1 наведено приклад вхідних даних в базі даних в таблиці `sensor_readings`.

Id	SensorId	Timestamp	Value
13018	5	2025-12-15 21:23:06	0.2819866943741155
13017	2	2025-12-15 21:23:06	0.06643173945901532
13016	1	2025-12-15 21:23:06	0.2819866943741155
13015	1	2025-12-15 21:23:05	0.06643173945901532
13014	5	2025-12-15 21:23:05	0.06643173945901532
13013	7	2025-12-15 21:23:04	0.2386456797219094
13012	3	2025-12-15 21:23:04	0.2386456797219094
13011	4	2025-12-15 21:23:04	0.2386456797219094

Рисунок 5.1 – Вхідні дані в базі даних в таблиці sensor_readings

Дані, що вводяться користувачем. Користувач вводить або обирає такі вхідні параметри:

- name – назва датчика;
- place – місце встановлення датчика;
- model – модель датчика;
- minThreshold – мінімальне допустиме порогове значення рівня радіації;
- maxThreshold – максимальне допустиме порогове значення рівня радіації;
- pollingInterval – період опитування датчика (у секундах, хвилинах або годинах);
- comment – коментар до датчика;
- вибір часових інтервалів для перегляду даних (хвилини, години, дні або довільний період).

На рисунку 5.2 наведено приклад вхідних даних в базі даних в таблиці sensors.

Id	Name	Place	Model	MinThreshold	MaxThreshold	PollingSec	Comment	Status	Start	smh
1	Лабораторія 1	Ангар 1	ДТ-3	0.025	0.53	1	Відсутній	Активний	2025-12-08 21:17:11	1
2	Лабораторія 2	Ангар 1	ДТ-3	0.03	0.49	2	Відсутній	Активний	2025-12-08 21:53:44	1
3	Склад 1	Ангар 2	ДТ-3	0.2	0.52	3	Відсутній	Активний	2025-12-08 22:37:00	1
4	Склад 2	Ангар 2	ДТ-3	0.01	0.65	4	Відсутній	Активний	2025-12-08 21:55:55	1
5	Склад 3	Ангар 2	ДТ-3	0.1	0.5	1	Відсутній	Активний	2025-12-14 21:05:54	1
6	Підвал 1	Вулиця	ДТ-3	0.015	0.51	8	Вулиця задня сторона	Активний	2025-12-08 22:52:10	1
7	Підвал 2	Вулиця	ДТ-3	0.02	0.52	60	Відсутній	Активний	2025-12-09 16:30:35	2

Рисунок 5.2 – Вхідні дані в базі даних в таблиці sensors

Дані з бази даних. До вхідних даних також належать дані, що зчитуються з бази даних:

- список зареєстрованих датчиків радіаційного моніторингу;
- параметри конфігурації кожного датчика;

- історичні дані зчитувань рівня радіації;
- записи журналу подій системи;
- інформація про активність або деактивацію зчитувань датчиків.

Службові та системні параметри. До вхідних даних належать також службові параметри, які використовуються для керування логікою роботи застосунку:

- поточна дата та час системи;
- обраний режим роботи (реальний моніторинг або перегляд архівних даних);
- параметри фільтрації та відображення інформації.

5.3.2 Вихідні дані

Вихідними даними програмного продукту є результати обробки, аналізу та візуалізації інформації про рівень іонізуючого випромінювання, які надаються користувачу у зручному для сприйняття вигляді.

Візуалізовані дані моніторингу. До основних вихідних даних належать графічні представлення результатів вимірювань радіаційного фону:

- графік зміни рівня радіації в часі;
- значення рівня радіації у вибраному часовому діапазоні;
- мінімальні та максимальні порогові лінії на графіку;
- історія вимірювань за заданий період (хвилини, години, дні або довільний інтервал).

Поточні показники датчиків. ПП відображає актуальні значення для обраного датчика:

- поточне значення рівня радіації ($\mu\text{Sv/h}$);
- дата та час останнього вимірювання;
- візуальну індикацію стану (нормальний рівень, попередження, перевищення порогу).

Інформація про датчики. До вихідних даних належить довідкова та

технічна інформація про датчики:

- ідентифікатор датчика;
- назва та місце встановлення;
- модель датчика;
- встановлені порогові значення;
- період опитування;
- поточний статус датчика;
- коментар до датчика;
- статус;
- дата початку роботи датчика.

Повідомлення та сповіщення. ПП формує та відображає:

- інформаційні повідомлення про дії користувача (додавання, редагування, видалення датчиків);
- попередження про перевищення допустимих рівнів радіаційного фону;
- системні повідомлення про запуск і завершення роботи застосунку;
- повідомлення про помилки та некоректні дії.

На рисунку 5.3 наведено приклад вихідних даних в базі даних в таблиці event_log.

Id	SensorId	Message	CreatedAt
15	NULL	Програму моніторингу закрито	2025-12-14 21:02:40
16	NULL	Програму моніторингу запущено	2025-12-14 21:05:31
17	1	Сенсор 'Лабораторія 1' (1): value 0,481 μ Sv/h вийш...	2025-12-14 21:05:43
18	2	Сенсор 'Лабораторія 2' (2): value 0,481 μ Sv/h вийш...	2025-12-14 21:05:51
19	5	Датчик №5: а успішно додано!	2025-12-14 21:05:54
20	5	Додано: ID: 5 Назва: а Місце встановлення: а Модел...	2025-12-14 21:05:54
21	3	Сенсор 'Склад 1' (3): value 0,481 μ Sv/h вийшов за ...	2025-12-14 21:06:01
22	NULL	Програму моніторингу закрито	2025-12-14 21:06:01
23	NULL	Програму моніторингу запущено	2025-12-14 21:09:44
24	1	Сенсор 'Лабораторія 1' (1): value 0,489 μ Sv/h вийш...	2025-12-14 21:09:59
25	2	Сенсор 'Лабораторія 2' (2): value 0,489 μ Sv/h вийш...	2025-12-14 21:10:14

Рисунок 5.3 – Вихідні дані в базі даних в таблиці event_log

Журнал подій системи. До вихідних даних також належить журнал подій, який містить:

- записи про всі важливі події системи;
- дату та час виникнення події;
- ідентифікатор відповідного датчика (за наявності);
- текстовий опис події.

Журнал подій може фільтруватися за датчиками та часовими інтервалами.

Дані, що зберігаються у базі даних. Результати роботи ПЗ зберігаються у базі даних у вигляді:

- записів зчитувань рівня радіації;
- журналу подій;
- оновлених параметрів датчиків.

Ці дані можуть повторно використовуватися для аналізу та перегляду історії.

5.4 Повідомлення

Під час роботи створений ПП виводить текстові повідомлення, які описані у табл. 5.1.

Таблиця 5.1 – Повідомлення під час роботи ПЗ

Повідомлення	Опис
Будь ласка, оберіть опцію.	Не обрано опцію(додати, редагувати, видалити).
Поле 'Назва' не може бути порожнім.	Не заповнено поле “Назва”.
Поле 'Місце встановлення' не може бути порожнім.	Не заповнено поле “Місце встановлення”.

Кінець таблиці 5.1

Повідомлення	Опис
Поле 'Модель' не може бути порожнім.	Не заповнено поле “Модель”.
Поле 'Min' повинно містити число.	Не заповнено поле “Min”, або введено не число.
Поле 'Max' повинно містити число.	Не заповнено поле “Max”, або введено не число.
Значення 'Min' повинно бути меншим за 'Max'.	Значення “Min” більше за “Max”.
Будь ласка, оберіть тип інтервалу (секунди/хвилини/години).	Не обрано тип інтервалу.
За обраний період подій немає.	Записів за обраний період немає.
Дата 'З' не може бути більшою за дату 'По'.	Початкова дата більша за кінцеву.
SensorConfig.Id must match sensor.Id	SensorConfig.Id має збігатися з sensor.Id.
Sensor with ID N already exists	Датчик з ідентифікатором(ID) N вже існує.
Сенсор N: value M $\mu\text{Sv/h}$ вийшов за встановлений діапазон О:Р	Значення М сенсора N вийшов за встановлений діапазон О:Р.
01.01.2025 12:12 Сенсор N: М $\mu\text{Sv/h}$	Актуальне значення М $\mu\text{Sv/h}$ Сенсора N.
Помилка запису в БД	Помилка запису даних в БД.
Датчик N успішно додано.	Датчик N успішно додано в систему.
Датчик N успішно відредаговано.	Успішно відредаговано дані датчика N.
Датчик N успішно видалено.	Датчик N успішно видалено з системи.

5.5 Запити до бази даних

Для отримання інформації про всі наявні датчики радіаційного моніторингу в системі використовується запит вибірки даних з таблиці *sensors*, який має вигляд (3.1):

```
SELECT Id, Name, Place, Model, MinThreshold,  
MaxThreshold, PollingSec, Smh, Comment,  
Status, Start FROM sensors, (3.1)
```

де *Id* – унікальний номер присвоєний кожному датчику;

Name – ім'я датчика;

Place – місце встановлення;

Model – модель датчика;

MinThreshold – мінімальне порогове значення;

MaxThreshold – максимальне порогове значення;

PollingSec – кількість інтервалу часу опитування;

Smh – показник часу(секунди/хвилини/години);

Comment – коментар;

Status – статус активності датчика;

Start – дата початку роботи датчика.

Для видалення датчика реалізовано запит, який визначає запис про датчик в базі даних за параметром *id* та видаляє запис за запитом (3.2):

```
DELETE FROM sensors WHERE Id = @SensorId, (3.2)
```

де *Id* – унікальний номер присвоєний кожному датчику.

Для редагування даних датчика реалізовано запит, який визначає запис за параметром *id* та перезаписує введені дані за запитом (3.3):

```

UPDATE sensors SET
    Name = @Name,
    Place = @Place,
    Model = @Model,
    MinThreshold = @MinThreshold,
    MaxThreshold = @MaxThreshold,
    PollingSec = @PollingSec,
    Smh = @Smh,
    Comment = @Comment,
    Status = @Status,
    Start = @Start
WHERE Id = @Id,

```

(3.3)

де Name – ім'я датчика;

Place – місце встановлення;

Model – модель датчика;

MinThreshold – мінімальне порогове значення;

MaxThreshold – максимальне порогове значення;

PollingSec – кількість інтервалу часу опитування;

Smh – показник часу(секунди/хвилини/години);

Comment – коментар;

Status – статус активності датчика;

Start – дата початку роботи датчика;

Id – унікальний номер присвоєний кожному датчику.

Для додавання нового датчика реалізовано запит, який записує в новий рядок бази даних введені дані за запитом (3.4):

```

INSERT INTO sensors(Id,Name,Place,Model,MinThreshold,
MaxThreshold,PollingSec,Smh,Comment,Status,Start)VALUES(@Id,
    @Name,@Place,@Model,@MinThreshold,@MaxThreshold,
    @PollingSec,@Smh,@Comment,@Status,@Start),

```

(3.4)

де Id – унікальний номер присвоєний кожному датчику;

Name – ім'я датчика;

Place – місце встановлення;

Model – модель датчика;

MinThreshold – мінімальне порогове значення;

MaxThreshold – максимальне порогове значення;

PollingSec – кількість інтервалу часу опитування;

Smh – показник часу(секунди/хвилини/години);

Comment – коментар;

Status – статус активності датчика;

Start – дата початку роботи датчика.

Для пошуку даних за вказаний діапазон, реалізовано запит, який визначає запис про виміри датчика в базі даних за параметром id та порівнює з початковою та кінцевою вказаними датами за запитом (3.5):

```
SELECT SensorId, Timestamp, Value FROM sensor_reading
WHERE SensorId = @SensorId AND Active = 1
AND Timestamp BETWEEN @From
AND @To ORDER BY Timestamp ASC, (3.5)
```

де SensorId – унікальний номер присвоєний кожному датчику;

From – початкова дата;

To – кінцева дата.

Для запису подій при роботі програми реалізовано запит, який додає тестове повідомлення до бази даних за запитом (3.6):

```
INSERT INTO event_log(SensorId, Message, CreatedAt)
VALUES (@SensorId, @Message, @CreatedAt) (3.6)
```

де SensorId – унікальний номер присвоєний кожному датчику;

Message – повідомлення, створене програмою;

CreatAt – дата та час створення повідомлення.

Для збереження результатів вимірювання радіаційного фону в базі даних використовується запит додавання нового запису до таблиці `sensor_readings`, який має вигляд (3.7):

$$\begin{aligned} &INSERT INTO sensor_readings (SensorId, Timestamp, Value) \\ &VALUES (@SensorId, @Timestamp, @Value), \end{aligned} \quad (3.7)$$

де `SensorId` – ідентифікатор датчика, з якого було отримано вимірювання;

`Timestamp` – дата та час отримання значення радіаційного фону;

`Value` – числове значення рівня радіаційного фону, виражене в мікрозівертах на годину ($\mu\text{Sv/h}$).

Для деактивації результатів вимірювань радіаційного фону конкретного датчика використовується SQL-запит оновлення записів у таблиці `sensor_readings`, який має вигляд (3.8):

$$\begin{aligned} &UPDATE sensor_readings SET Active = FALSE \\ &WHERE SensorId = @SensorId, \end{aligned} \quad (3.8)$$

де `Active` – логічне поле, яке визначає актуальність запису вимірювання;

`SensorId` – ідентифікатор датчика, вимірювання якого необхідно деактивувати.

5.6 Повідомлення оператору

Під час роботи оператора з розробленим ПП виводяться різні текстові повідомлення, вони інформують оператора про помилки або його неправильні дії, а також про успішність виконання різних маніпуляцій.

Список всіх текстових повідомлень, описання їх змісту та дії оператора, які він повинен виконати при отриманні цих повідомлень представлені у розділі “Керівництво програміста” в таблиці 5.1.

5.7 Висновок

У цьому розділі було представлено призначення та функції розробленого програмного продукту ПП, а також мінімальні системні вимоги для його коректної роботи. Докладно описані модулі, з яких складається ПП. Створений ПП має відкритий вихідний код, що дозволяє іншим розробникам переглядати, вивчати та змінювати його. Також у цьому розділі міститься опис вхідних і вихідних даних, а також список усіх текстових повідомлень, які можуть з'являтися під час роботи користувача з ПП. Описано структуру бази даних та її таблиць.

6 КЕРІВНИЦТВО ОПЕРАТОРА

6.1 Призначення й умови виконання програмного продукту

Розроблюваний програмний продукт призначений для моніторингу рівня радіаційного фону за допомогою системи датчиків радіаційного вимірювання. Програмний продукт реалізований у вигляді окремого віконного застосунку (Windows-застосунку), який дозволяє в режимі реального часу переглядати показники радіаційного фону, аналізувати історію вимірювань та здійснювати контроль перевищення допустимих порогових значень.

Застосунок забезпечує можливість керування датчиками радіаційного контролю, зокрема додавання, редагування та видалення датчиків, а також налаштування параметрів їх опитування. Окрему увагу приділено журналу подій, у якому фіксуються системні повідомлення, попередження про перевищення рівня радіації та дії користувача. Це дозволяє оперативно реагувати на небезпечні зміни радіаційного фону та здійснювати подальший аналіз роботи системи.

Для коректної роботи ПП необхідні такі мінімальні системні вимоги:

- платформа x64-based;
- операційна система Microsoft Windows 11/10/8/7 (рекомендується встановити всі останні оновлення);
- процесор з мінімальною тактовою частотою 1,8 ГГц, рекомендується використовувати як мінімум двоядерний процесор;
- оперативна пам'ять від 2 ГБ (рекомендується 4 ГБ та мінімум 2,5ГБ при використанні на віртуальній машині);
- роздільна здатність екрану 1920x1080 пікселів з 16-бітовим кольором;
- мінімум 2 ГБ вільного місця на жорсткому диску.

6.2 Виконання програмного продукту

Після вдалого запуску ПП відкривається головне вікно, яке представлено на рис. 6.1.

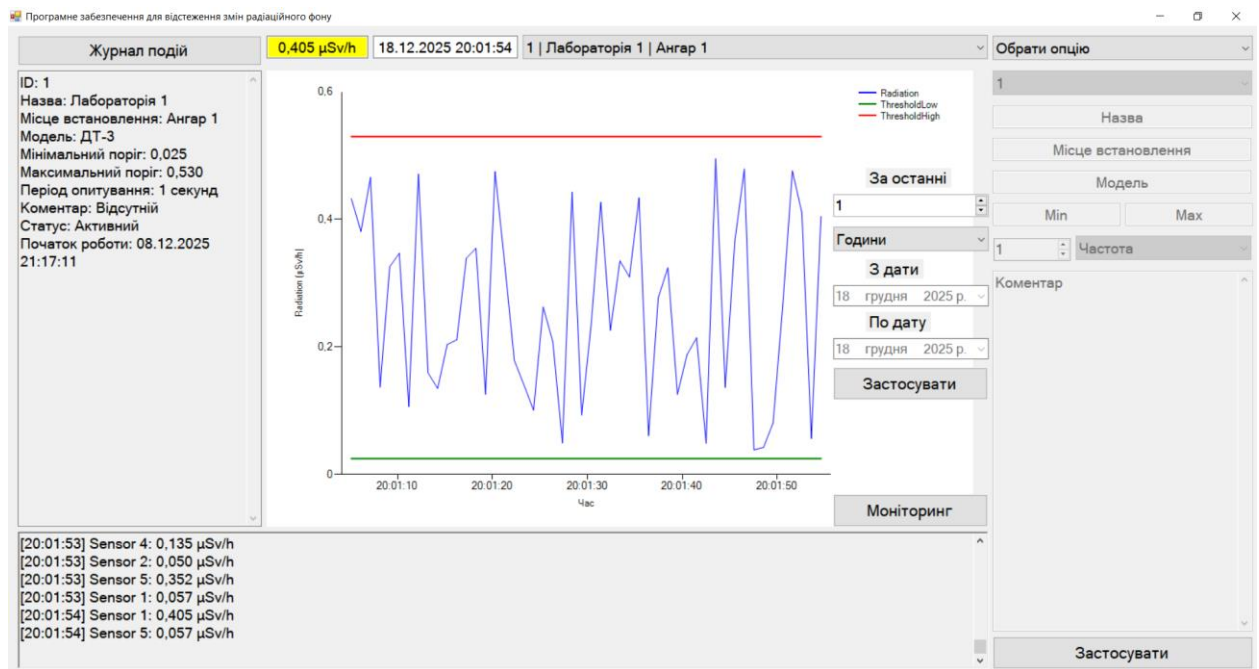


Рисунок 6.1 – Головне вікно ПП

На рисунку 6.2 зображено блок, де виводиться вся інформація про поточний датчик. В цьому блоці виводиться така інформація як ідентифікаційний номер датчика, його назва, місце розташування, модель, мінімальний та максисальний пороги, період опитування, коментар, статус та дата початку роботи. Також в самому верху розташовано кнопку “Журнал подій”. При натисканні на неї відкривається паралельно з основним вікном додаткове вікно “Журнал подій”(рис. 6.3). На цій формі ліворуч розташоване поле для виведення зафіксованих повідомлень з журналу подій. Праворуч параметри журналу подій. Тут поле для введення числа, вибору періоду(хвилини, години, дні та довільний діапазон) та введення початкової та кінцевої дати. При діапазонах хвилини, години, дні дата не вводиться, а при довільному діапазоні оператор може вводити тільки початкову і кінцеву

дати. Під цими полями список з датчиків. Якщо треба вивести повідомлення, які стосуються конкретних датчиків, оператор обирає потрібні датчики з цього списку. Після встановлення всіх налаштувань оператор натискає кнопку “Застосувати” і програма виводить інформацію з журналу подій, опираючись на задані фільтри.

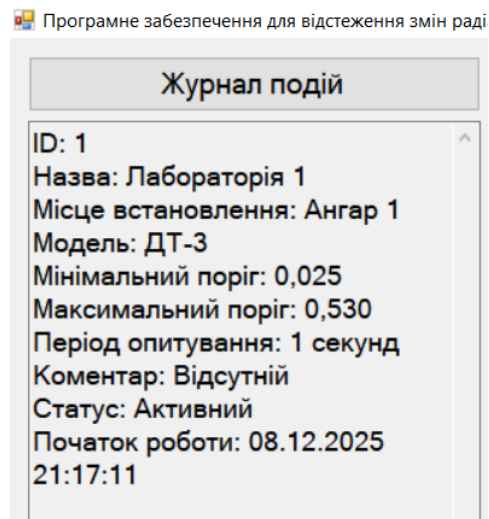


Рисунок 6.2 – Блок з основною інформацією про датчик

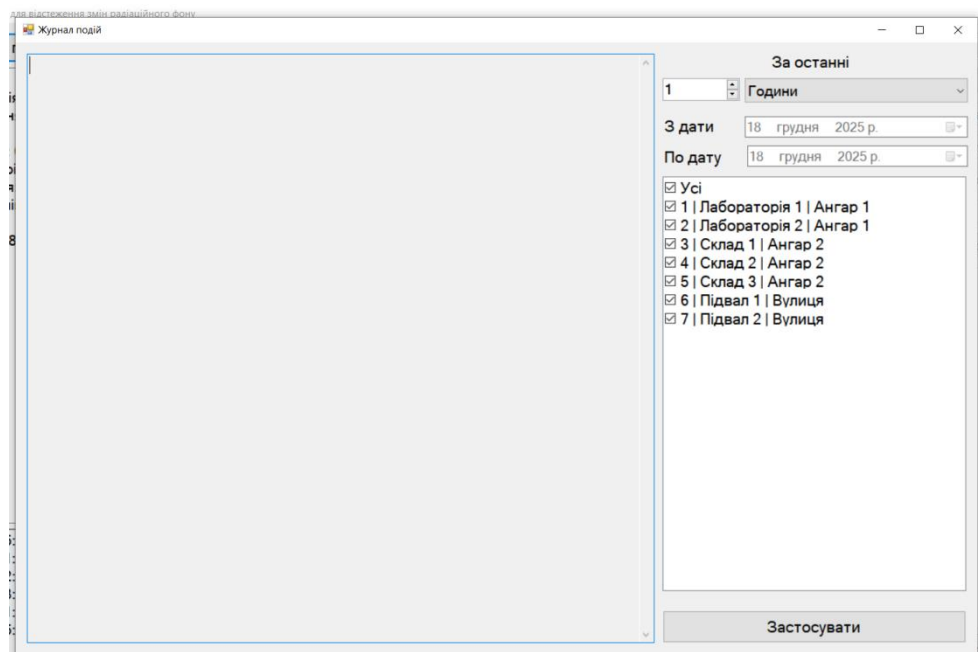


Рисунок 6.3 – Вікно “Журнал подій”

На рисунку 6.4 зображено блок моніторингу показників датчика.

Зверху відображається останнє отримане значення від сенсора(в залежності від заданих мінімального і максимального порогів колір може бути зеленим, жовтим та червоним), дата отримання цього значення та випадаючий список для перемикання між всіма датчиками. В центрі блоку графік, де виводяться значення з датчика. Зеленою лінією позначається мінімальний поріг, а червоною відповідно максимальний поріг. Синя лінія відображає дані з датчика. Вісь X – час, вісь Y – значення радіації від датчика. Правоч в блоці розташовано параметри записів. Тут поле для введення числа, вибору періоду(хвилини, години, дні та довільний діапазон) та введення початкової та кінцевої дати. При діапазонах хвилини, години, дні дата не вводиться, а при довільному діапазоні оператор може вводити тільки початкову і кінцеву дати. Дані виводяться для обраного датчика зверху в блоці у випадаючому списку. Після встановлення всіх налаштувань оператор натискає кнопку “Застосувати” і програма будує графік з даних, опираючись на задані фільтри. Щоб повернутися до моніторингу в реальному часі, треба натиснути на кнопку “Моніторинг” внизу блоку.

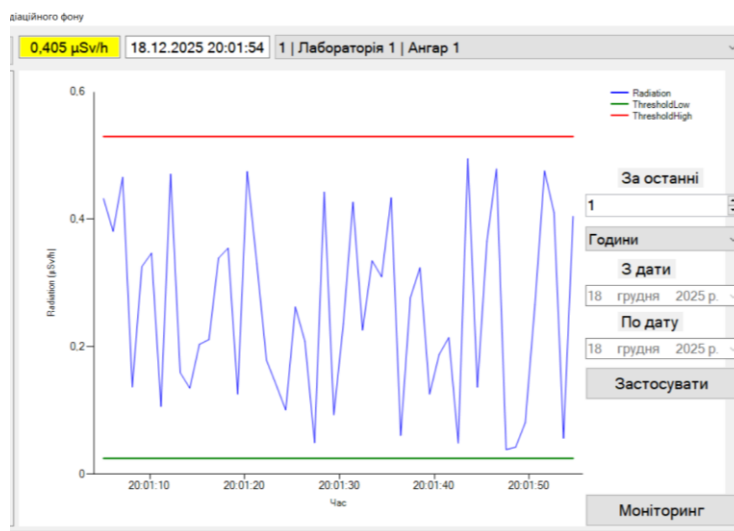


Рисунок 6.4 – Блок моніторингу показників датчика

На рисунку 6.5 нведено поле, де виводяться всі актуальні виміри датчиків у вигляді текстового повідомлення. При виявленні значення від

любого датчика, яке вийшло за встановлені межі в реадньому часі оператору надходить повідомлення про цю подію, незалежно, який датчик відкрито для перегляду а також в полі дублюється повідомлення і виводиться червоним кольором.

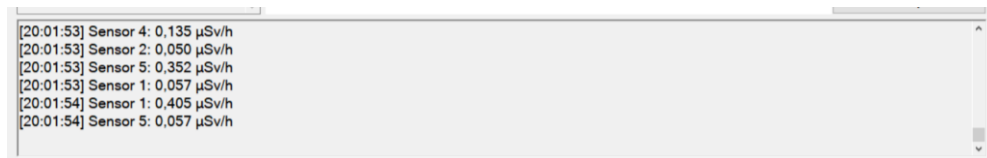


Рисунок 6.5 – Поле для виведення актуальних значень всіх датчиків

На рисунку 6.6 наведено блок додавання, редагування та видалення датчика. В самому верхі оператор обирає режим роботи(додати, редагувати або видалити).

Режим “Додати”. Оператор не може обирати датчик, адже програма сама генерує новий унікальний Id для нового датчикаю оператор вводить назву, місце розташування, модель датчика, мінімальне і максимальне порогові значення, обирає тип періода(секунди, хвилини, години) та вводить бажаний інтервал. Далі є необов’язкове поле “Коментар”. Якщо оператор нічого не вводить, програма присвоює запис “Відсутній”. Решта полів є обов’язковими для введення. Після завершення введення всіх даних треба натиснути кнопку “Застосувати”.

Режим “Редагувати”. При виборі цього режиму у поля назва, місце розташування, модель датчика, мінімальне і максимальне порогові значення, інтервал і тип інтервалу та коментар завантажуються дані з обраного датчика у випадяючому списку в самому верху блока. За замовчуванням обрано перший в списку датчик і завантажуються його дані. Далі оператор може вносити зміни в кожне поле. По завершення редагування треба натиснути кнопку “Застосувати”.

Режим “Видалити”. Оператору дається можливість тільки обирати номер датчика з випадяючого списку зверху блока. Після вибору датчика

треба натиснути кнопку “Застосувати”.

Рисунок 6.6 – Блок для додавання, редагування та видалення датчика

6.3 Повідомлення оператору

Під час роботи оператора з розробленим ПП виводяться різні текстові повідомлення, вони інформують оператора про помилки або його неправильні дії, а також про успішність виконання різних маніпуляцій.

Список всіх текстових повідомлень, описання їх змісту та дії оператора, які він повинен виконати при отриманні цих повідомлень представлені у розділі “Керівництво програміста” в таблиці 5.1.

6.4 Висновок

У цьому розділі було представлено призначення та функціональні можливості розробленого ПП, а також мінімальні системні вимоги, необхідні для його коректної роботи. Наведено покрокову інструкцію для користувача щодо роботи з ПП, включаючи основні режими моніторингу та аналізу

радіаційних показників.

Описано всі текстові повідомлення та сповіщення, які можуть з'являтися під час роботи користувача із системою, зокрема інформаційні повідомлення, попередження та повідомлення про перевищення порогових значень радіаційного фону. Надано детальну інструкцію з використання ПП, включаючи приклади додавання, редагування та видалення датчиків радіаційного контролю, перегляд історії вимірювань та аналіз журналу подій.

Інтерфейс ПП є інтуїтивно зрозумілим і зручним у використанні, що не потребує спеціальної підготовки користувача. У разі виникнення питань щодо роботи із системою передбачено довідкову інформацію, яка забезпечує швидке ознайомлення з функціональними можливостями ПП.

7 ЕКСПЕРИМЕНТИ ТА РЕЗУЛЬТАТИ

7.1 Проведення експериментів

Спроектувавши та створивши ПП було проведено випробування функціоналу. Було перевірено та детально описано дії при перевірці працездатності таких методів як додавання, редагування, видалення датчика. Також було перевірено перемикання між датчиками для моніторингу в реальному часі, перегляд вимірювань обраного датчика за встановлений період, та перегляд системних повідомлень з журналу подій за встановлений період.

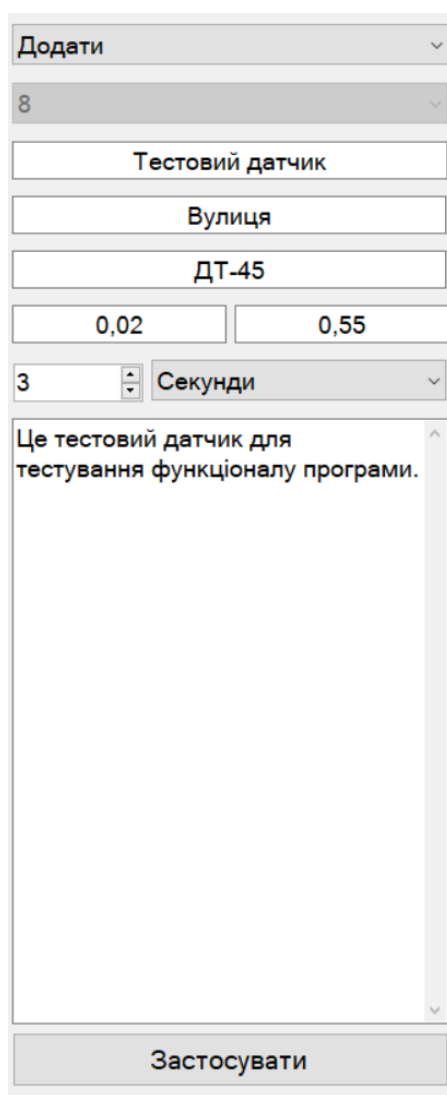
7.1.1 Додавання датчика

На рисунку 7.1 наведено таблицю “sensors” з бази даних до виконання додавання датчика.

Id	Name	Place	Model	MinThreshold	MaxThreshold	PollingSec	Comment	Status	Start	smh
1	Лабораторія 1	Ангар 1	ДТ-3	0.025	0.53	1	Відсутній	Активний	2025-12-08 21:17:11	1
2	Лабораторія 2	Ангар 1	ДТ-3	0.03	0.49	2	Відсутній	Активний	2025-12-08 21:53:44	1
3	Склад 1	Ангар 2	ДТ-3	0.2	0.52	3	Відсутній	Активний	2025-12-08 22:37:00	1
4	Склад 2	Ангар 2	ДТ-3	0.01	0.65	4	Відсутній	Активний	2025-12-08 21:55:55	1
5	Склад 3	Ангар 2	ДТ-3	0.1	0.5	1	Відсутній	Активний	2025-12-14 21:05:54	1
6	Підвал 1	Вулиця	ДТ-3	0.015	0.51	8	Вулиця задня сторона	Активний	2025-12-08 22:52:10	1
7	Підвал 2	Вулиця	ДТ-3	0.02	0.52	60	Відсутній	Активний	2025-12-09 16:30:35	2

Рисунок 7.1 – Таблиця “sensors” з бази даних до виконання додавання датчика

На рисунках 7.2 – 7.4 наведено процес додавання нового датчика.



Додати

8

Тестовий датчик

Вулиця

ДТ-45

0,02 0,55

3 Секунди

Це тестовий датчик для тестування функціоналу програми.

Застосувати

Рисунок 7.2 – Заповнені поля для додавання датчика

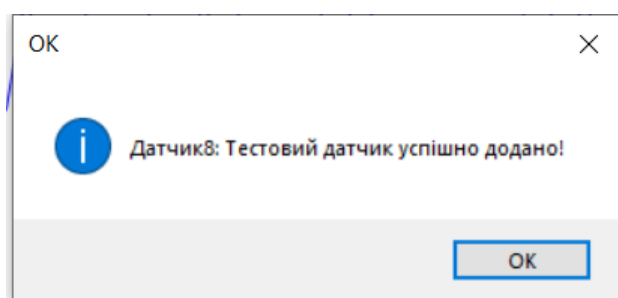


Рисунок 7.3 – Повідомлення про успішне додавання датчика

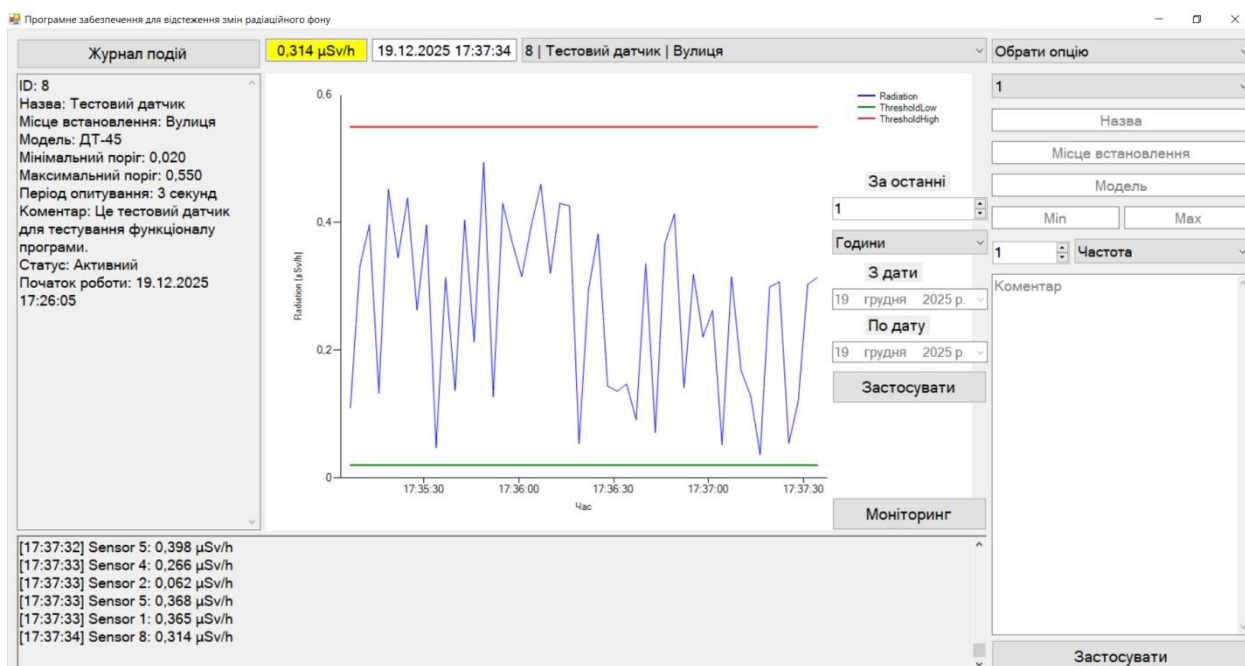


Рисунок 7.4 – Новий датчик вже заведено в систему

На рисунку 7.5 показано новий запис в таблиці “sensors” в базі даних.

ID	Name	Place	Model	MinThreshold	MaxThreshold	PollingSec	Comment	Status	Start	smh
1	Лабораторія 1	Ангар 1	ДТ-3	0.025	0.53	1	Відсутній	Активний	2025-12-08 21:17:11	1
2	Лабораторія 2	Ангар 1	ДТ-3	0.03	0.49	2	Відсутній	Активний	2025-12-08 21:53:44	1
3	Склад 1	Ангар 2	ДТ-3	0.2	0.52	3	Відсутній	Активний	2025-12-08 22:37:00	1
4	Склад 2	Ангар 2	ДТ-3	0.01	0.65	4	Відсутній	Активний	2025-12-08 21:55:55	1
5	Склад 3	Ангар 2	ДТ-3	0.1	0.5	1	Відсутній	Активний	2025-12-14 21:05:54	1
6	Підвал 1	Вулиця	ДТ-3	0.015	0.51	8	Вулиця задня сторона	Активний	2025-12-08 22:52:10	1
7	Підвал 2	Вулиця	ДТ-3	0.02	0.52	60	Відсутній	Активний	2025-12-09 16:30:35	2
8	Тестовий датчик	Вулиця	ДТ-45	0.02	0.55	3	Це тестовий датчик для тестування функціоналу прог...	Активний	2025-12-19 17:26:05	1

Рисунок 7.5 – Новий запис в таблиці “sensors” в базі даних

На рисунку 7.6 наведено таблицю “sensor_readings”, яку йде запис показів датчиків, де помітно запис датчика 8.

Id	SensorId	Timestamp	Value	Active
22598	4	2025-12-19 18:08:10	0.3898175004403188	1
22597	8	2025-12-19 18:08:10	0.49995014438403307	1
22596	1	2025-12-19 18:08:10	0.32245015451798686	1
22595	5	2025-12-19 18:08:10	0.1293267042233267	1

Рисунок 7.6 – Записи показів з датчика 8

7.1.2 Редагування датчика

На рисунках 7.7 – 7.9 показано процес редагування датчика. Було змінено такі параметри:

- модель: ДТ-38;
- мінімальний поріг: 0,05;
- максимальний поріг: 0,51;
- період опитування: 5 хвилин.

Редагувати

8

Тестовий датчик

Вулиця

ДТ-38

0,05 0,51

5 Хвилини

Це тестовий датчик для тестування функціоналу програми.

Застосувати

Рисунок 7.7 – Записано нові дані для датчика 8

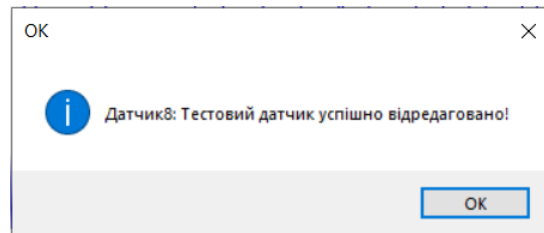


Рисунок 7.8 – Повідомлення про успішне редагування датчика

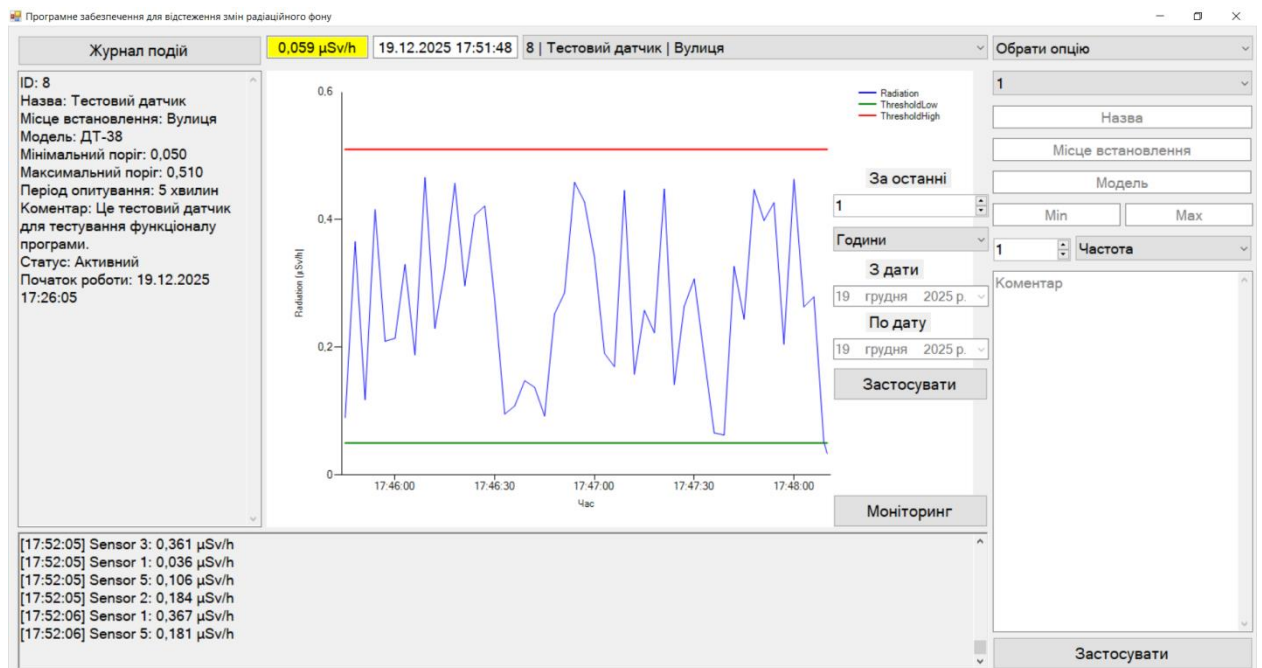


Рисунок 7.9 – Демонстрація відредагованого датчика в системі

На рисунку 7.10 показано зміну даних в записі в таблиці “sensors” в базі даних.

Id	Name	Place	Model	MinThreshold	MaxThreshold	PollingSec	Comment	Status	Start	smh
1	Лабораторія 1	Ангар 1	ДТ-3	0.025	0.53	1	Відсутній	Активний	2025-12-08 21:17:11	1
2	Лабораторія 2	Ангар 1	ДТ-3	0.03	0.49	2	Відсутній	Активний	2025-12-08 21:53:44	1
3	Склад 1	Ангар 2	ДТ-3	0.2	0.52	3	Відсутній	Активний	2025-12-08 22:37:00	1
4	Склад 2	Ангар 2	ДТ-3	0.01	0.65	4	Відсутній	Активний	2025-12-08 21:55:55	1
5	Склад 3	Ангар 2	ДТ-3	0.1	0.5	1	Відсутній	Активний	2025-12-14 21:05:54	1
6	Підвал 1	Вулиця	ДТ-3	0.015	0.51	8	Вулиця задня сторона	Активний	2025-12-08 22:52:10	1
7	Підвал 2	Вулиця	ДТ-3	0.02	0.52	60	Відсутній	Активний	2025-12-09 16:30:35	2
8	Тестовий датчик	Вулиця	ДТ-38	0.05	0.51	300	Це тестовий датчик для тестування функціоналу прог...	Активний	2025-12-19 17:26:05	2

Рисунок 7.10 – Відредагований запис в таблиці “sensors” в базі даних

7.1.3 Видалення датчика

На рисунках 7.11 – 7.13 показано процес видалення датчика.



Рисунок 7.11 – В програмі обрано опцію видалити і номер датчика

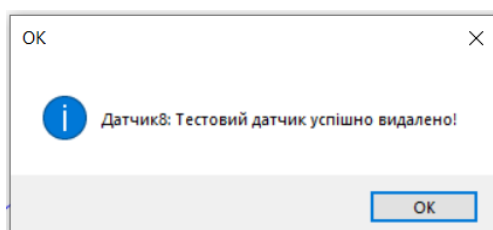


Рисунок 7.12 – Повідомлення про успішне видалення датчика



Рисунок 7.13 – В програмі тепер неможливо переглянути датчик 8

На рисунку 7.14 наведено зміни в таблиці “sensors” в базі даних.

Id	Name	Place	Model	MinThreshold	MaxThreshold	PollingSec	Comment	Status	Start	smh
1	Лабораторія 1	Ангар 1	ДТ-3	0.025	0.53	1	Відсутній	Активний	2025-12-08 21:17:11	1
2	Лабораторія 2	Ангар 1	ДТ-3	0.03	0.49	2	Відсутній	Активний	2025-12-08 21:53:44	1
3	Склад 1	Ангар 2	ДТ-3	0.2	0.52	3	Відсутній	Активний	2025-12-08 22:37:00	1
4	Склад 2	Ангар 2	ДТ-3	0.01	0.65	4	Відсутній	Активний	2025-12-08 21:55:55	1
5	Склад 3	Ангар 2	ДТ-3	0.1	0.5	1	Відсутній	Активний	2025-12-14 21:05:54	1
6	Підвал 1	Вулиця	ДТ-3	0.015	0.51	8	Вулиця задня сторона	Активний	2025-12-08 22:52:10	1
7	Підвал 2	Вулиця	ДТ-3	0.02	0.52	60	Відсутній	Активний	2025-12-09 16:30:35	2

Рисунок 7.14 – Зміни в таблиці “sensors” в базі даних після видалення датчика

7.1.4 Перегляд записів за встановлений проміжок

На рисунках 7.15 – 7.16 наведено перегляд записів за встановленими діапазонами.

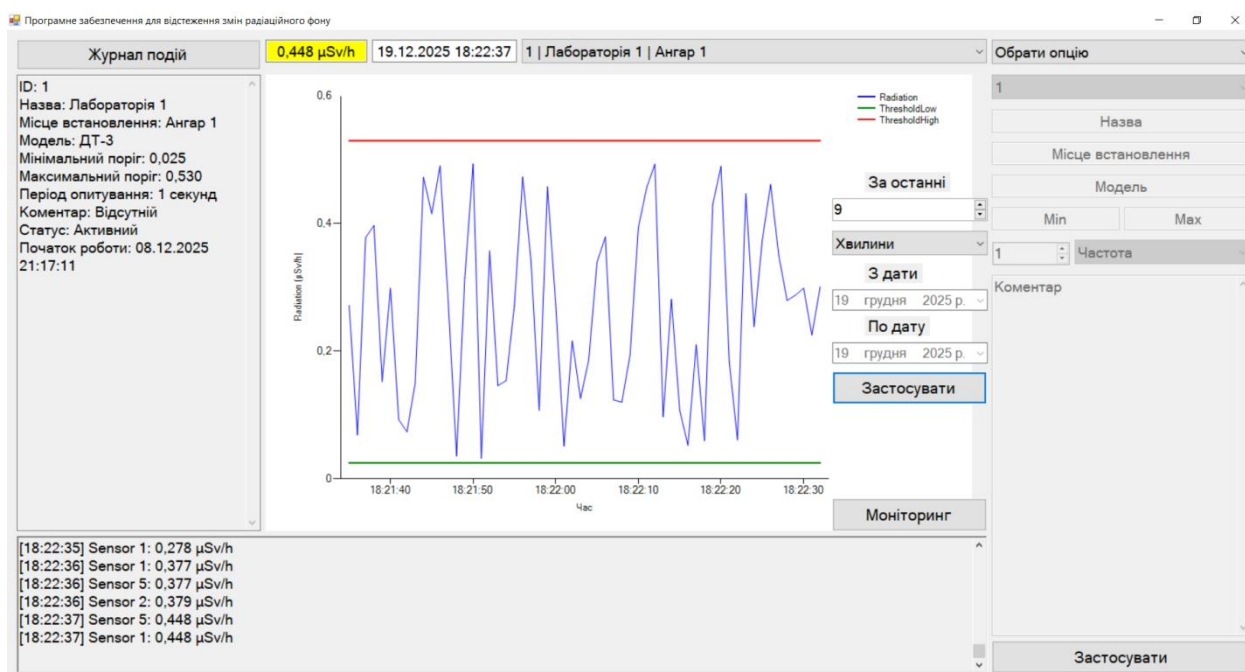


Рисунок 7.15 – Перегляд показів за останні 9 хвилин для датчика 1

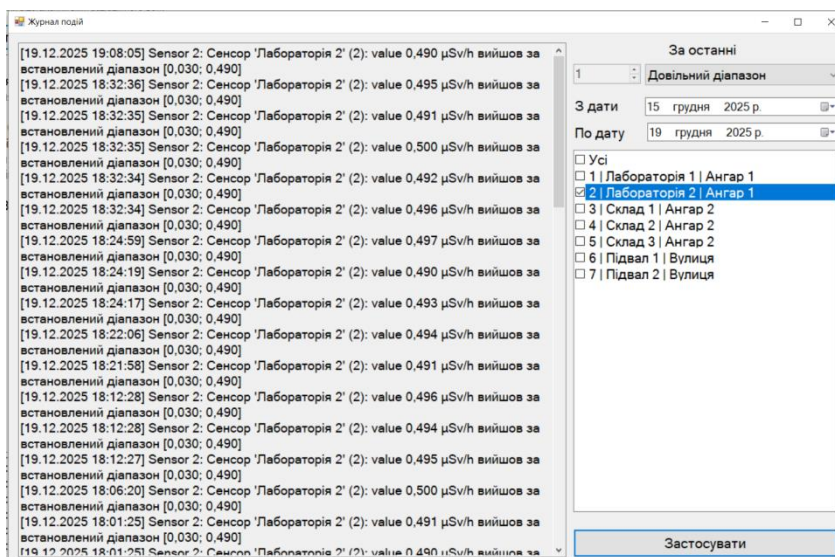


Рисунок 7.18 – Перегляд журналу подій з 15.12.2025 по 19.12.2025 для датчика 2

7.2 Висновок

У цьому розділі було проведено тестування основного функціоналу системи, що підтвердило її коректну та стабільну роботу. Було перевірено працездатність ключових методів керування датчиками, зокрема додавання, редагування та видалення датчиків радіаційного контролю. Також протестовано механізм перемикання між датчиками з метою здійснення моніторингу в реальному часі, перегляд історії вимірювань обраного датчика за заданий період, а також аналіз системних повідомлень, що зберігаються в журналі подій.

ВИСНОВКИ

Під час виконання дипломної роботи було проведено ґрунтовний аналіз предметної області радіаційного контролю та моніторингу, визначено основні поняття, пов'язані з об'єктом дослідження і розробки, а також сформульовано проблематику поставленого завдання. Обґрунтовано актуальність і доцільність створення програмного засобу для автоматизованого контролю рівня радіаційного випромінювання з можливістю оперативного реагування на перевищення допустимих порогових значень.

У межах роботи було досліджено основні підходи та методи моніторингу радіаційних показників, проаналізовано їх переваги та недоліки, а також проведено порівняльний аналіз існуючих рішень у сфері збору, обробки та візуалізації даних радіаційних вимірювань.

Практична цінність роботи полягає у розробці програмного продукту, який поєднує в єдиній системі автоматизований збір радіаційних даних, моніторинг показників у реальному часі, збереження історії вимірювань, ведення журналу подій та керування параметрами датчиків. Запропоноване рішення спрощує процес контролю радіаційної обстановки та забезпечує своєчасне виявлення потенційно небезпечних ситуацій, що особливо актуально в умовах підвищеного техногенного навантаження та нестабільних зовнішніх факторів.

Розроблений програмний продукт призначений для моніторингу рівня радіаційного випромінювання за допомогою мережі датчиків. Програмний продукт реалізовано у вигляді окремого віконного застосунку (Windows-застосунку), який надає користувачеві можливість переглядати поточні радіаційні показники, аналізувати дані за обраний період, відслідковувати перевищення порогових значень, переглядати журнал подій та керувати параметрами датчиків радіаційного контролю.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Що таке радіаційний фон [Електрон. ресурс]. – Режим доступу : <https://zp.cdc.gov.ua/news/shho-take-radiatsijnyj-fon-2/>.
2. Гандзюк М. П. Основи охорони праці : підручник. 5-е вид. / М. П. Гандзюк, Є. П. Желібо, М. О. Халімовський ; за ред. М. П. Гандзюка. – К. : Каравела, 2011. – 384 с.
3. Можливі наслідки радіаційного опромінення та забруднення для здоров'я [Електрон. ресурс]. – Режим доступу: <https://phc.org.ua/reaguvannya-na-nadzvichayni-situacii/radiaciyna-nadzvichayna-situaciya/mozhlivi-naslidki-radiaciynogo-oprominennya-ta-zabrudnennya-dlya-zdorovya>.
4. Дозиметр – що це таке, принцип роботи [Електрон. ресурс]. – Режим доступу : <https://www.0352.ua/list/332019>.
5. Системи екологічного управління : сучасні тенденції та міжнародні стандарти : посібник / [С. В. Берзіна, І. І. Яреськовська та ін.]. – К : Інститут екологічного управління та збалансованого природокористування, 2017. – 134 с.
6. International Atomic Energy Agency (IAEA). Generic Procedures for Monitoring in a Nuclear or Radiological Emergency (IAEA-TECDOC-1092). – Vienna : IAEA, 1999. – 306 с.
7. Ministry of Health (New Zealand). Guidance for Radiation Monitoring and Instrument Selection. – Wellington : Ministry of Health, March 2024. – PDF, 184.72 KB.
8. Real-Time Monitoring Method for Radioactive Substances Using Monolithic Active Pixel Sensors (MAPS) [Electronic resource]. – Access mode : <https://www.mdpi.com/1424-8220/22/10/3919>.
9. State-of-the-Art Mobile Radiation Detection Systems for Different Scenarios [Electronic resource]. – Access mode : <https://www.mdpi.com/1424-8220/21/4/1051>.
10. Radiation early warning system [Electronic resource]. – Access

mode : <https://www.bmluk.gv.at/en/topics/climate-environment/radiation-protection/radiation-early-warning-system>.

11. Гудков І. М. Радіоекологічний моніторинг : навчальний посібник / І. М. Гудков, В. О. Кашпаров, О. Ю. Паренюк. – Київ, 2019. – 188 с.

12. RadonEye RD200 Radon Gas Detector | BESTSELLER 2025 [Electronic resource]. – Access mode : <https://www.radonshop.com/ftlab-radoneye-rd200-radon-gas-detector-r222-testing>.

13. Safecast is an international volunteer driven non-profit organization [Electronic resource]. – Access mode : <https://safecast.org/about/>.

14. GammaGuard [Electronic resource]. – Access mode : <https://gammawatch.com/>.

15. Найпопулярніші мови програмування у 2024 році [Електрон. ресурс]. – Режим доступу : <https://dan-it.com.ua/uk/blog/samyje-populjarnye-jazyki-programmirovanija-v-2024-godu/>.

16. Кофанов О. Є. Програмування із використанням R у статистичних та маркетингових дослідженнях / О. Є. Кофанов, С. О. Солнцев, О. В. Зозульов. – Київ : КПІ ім. Ігоря Сікорського, 2023. – 204 с.

17. Як вивчити мову програмування C# та стати .NET розробником [Електрон. ресурс]. – Режим доступу : <https://edu.cbsystematics.com/ua/blog/learn-csharp-become-dnet>.

18. Lee, W. Python machine learning / W. Lee. – London : Wiley, 2019. – 320 p.

19. Swift (мова програмування) : нова епоха універсального коду та неймовірних шляхів до інновацій [Електрон. ресурс]. – Режим доступу : <https://bdut.co.ua/pro-nas/swift-mova-programuvannya/>.

20. Чому варто програмувати на мові Swift? [Електрон. ресурс]. – Режим доступу : <https://novograd.city/articles/268916/chomu-varto-programuvati-na-movi-sift>.

21. Нова мова програмування swift : коментарі розробників [Електрон. ресурс]. – Режим доступу : <https://www.cn.ua/news/apple/16717->

nova-mova-programyvannia-swift-komentari-rozrobnikiv.html.

22. Сьерра, К. Изучаем Java. Лекции и упражнения / К. Сьерра, Б. Бэйтс. – М. : Эксмо, 2017. – 720 с.

23. Найкращі IDE для програмування на C# [Електрон. ресурс]. – Режим доступу : <https://deveducation.com/blog/luchshie-ide-dlya-razrabotki-na-c/>.

24. Microsoft Visual Studio Reviews [Electronic resource]. – Access mode : <https://www.softwareadvice.com/game-development/microsoft-visual-studio-profile/reviews/>.

25. Stetsyuk, V. Method of choosing the programming environment for software / V. Stetsyuk, T. Hovorushchenko // International Scientific Journal Computer Systems and Information Technologies. – 2021. – Vol. 2. – P. 6–10.

26. Rider [Electronic resource]. – Access mode : <https://www.trustradius.com/products/jetbrains-rider/reviews?qs=pros-and-cons#overview>.

27. Xamarin vs. Android Studio – A Comparison Guide with Pros and Cons [Electronic resource]. – Access mode : <https://medium.com/nerd-for-tech/xamarin-vs-android-studio-a-comparison-guide-with-pros-and-cons-dab30cb11008>.

28. Overview of ASP.NET Core MVC [Electronic resource]. – Access mode : <https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-9.0>.

29. Model-View-Presenter Pattern in Java : Enhancing UI Logic Separation for Cleaner Code [Electronic resource]. – Access mode : <https://java-design-patterns.com/patterns/model-view-presenter/>

30. Model-View-ViewModel (MVVM) [Electronic resource]. – Access mode : <https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm>

31. N-tier architecture style [Electronic resource]. – Access mode : <https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier>

ДОДАТОК А
Технічне завдання

Вступ

Розроблюваний програмний продукт використовується для моніторингу рівня радіаційного фону, фіксації та аналізу змін показників іонізуючого випромінювання, а також для управління датчиками радіаційного контролю. Програмний продукт «Radiation Monitoring System» представлений у вигляді окремого віконного додатку (Windows-застосунку), за допомогою якого користувач може в зручній формі здійснювати спостереження за поточними значеннями радіаційного фону та переглядати історію вимірювань.

Застосунок також надає можливість виконувати такі функції, як додавання та видалення датчиків радіації, редагування їх параметрів, перегляд показників у реальному часі, перегляд вимірювань за обраний період часу та ведення журналу подій, що забезпечує ефективний контроль радіаційної безпеки.

A.1 Мета розробки і її призначення

Об'єкт дослідження – процес інженерії програмного забезпечення для моніторингу радіаційного фону.

Предмет дослідження – методи, алгоритми та програмні засоби вимірювання, аналізу та оперативного контролю змін радіаційного фону.

Мета роботи – збільшення точності збору та аналізу даних вимірювання радіаційного випромінювання та швидкості оперативного інформування користувача про його небезпечні зміни.

Метод дослідження – метод порівняння значень з еталонними.

Актуальність роботи полягає в тому, що розробка застосунку для моніторингу рівня радіаційного фону та відстеження його змін у навколишньому середовищі має важливе значення з кількох ключових причин. По-перше, такий застосунок сприяє забезпеченню радіаційної

безпеки, оскільки постійний контроль показників іонізуючого випромінювання дозволяє своєчасно виявляти небезпечні відхилення від нормативних значень та оперативно реагувати на потенційні загрози для здоров'я людей і довкілля. Він також допомагає зменшити ризики негативного впливу радіації, забезпечуючи стабільні та контрольовані умови перебування людей на контрольованих об'єктах.

По-друге, застосунок дозволяє дотримуватися вимог чинних норм і стандартів у сфері радіаційної безпеки, що є критично важливим для промислових підприємств, лабораторій, медичних установ та об'єктів екологічного контролю. Він спрощує процес виконання регуляторних вимог, підвищує ефективність під час перевірок, аудитів та інспекцій, а також забезпечує збереження та аналіз результатів вимірювань у вигляді журналу подій.

A.2 Стислий зміст основних теоретичних посилок до розробки на основі проведених пошуків

Потреба у постійному контролі рівня радіаційного фону та своєчасному виявленні його змін виникає на регулярній основі, особливо в умовах підвищених техногенних і природних ризиків. Унаслідок різних непередбачуваних ситуацій, таких як воєнні дії, аварії на промислових або енергетичних об'єктах, природні катастрофи чи порушення в роботі систем контролю, можливі різкі коливання рівня іонізуючого випромінювання.

Для забезпечення безпеки населення, персоналу підприємств та навколишнього середовища вкрай важливо здійснювати безперервний моніторинг радіаційного фону у контрольованих зонах. Своєчасне отримання та аналіз даних вимірювань дозволяє оперативно реагувати на небезпечні відхилення від нормативних значень, мінімізувати негативний вплив радіації та запобігти можливим наслідкам для здоров'я людей і довкілля. Саме тому розробка програмного забезпечення для автоматизованого моніторингу

радіаційного фону є актуальною та доцільною.

A.3 Підстави для розробки

Підставою для виконання розробки є завдання на дипломний проект на тему “Програмне забезпечення для відстеження змін радіаційного фону”, затверджене наказом по НУ «Запорізька політехніка» № 447 від 30 вересня 2025 року.

A.4 Основні вимоги до програми, що розробляється, початковим даним і результату

A.4.1 Вимоги до функціональних характеристик

Вимоги до функціональних характеристик програмного продукту:

- повинна бути реалізована операція моніторингу рівня радіаційного фону з використанням даних, отриманих від радіаційних датчиків (у тому числі з імітацією їх роботи);
- повинна бути реалізована операція безперервного контролю змін рівня радіації в реальному часі для обраного датчика;
- повинна бути реалізована можливість перегляду історії радіаційних вимірювань за заданий період часу (хвилини, години, дні або довільний часовий інтервал);
- повинна бути реалізована операція порівняння отриманих значень радіаційного фону з мінімальними та максимальними пороговими значеннями, заданими для кожного датчика;
- повинна бути реалізована можливість формування попереджень і повідомлень у разі перевищення допустимих рівнів радіаційного фону;
- повинна бути реалізована можливість додавання, редагування та видалення радіаційних датчиків із збереженням відповідних даних у базі даних;

- повинна бути реалізована можливість перемикання між датчиками для перегляду поточних та історичних даних моніторингу;
- повинна бути реалізована операція ведення журналу подій, у якому фіксуються системні повідомлення, попередження та дії користувача.

A.4.2 Вимоги до надійності

Програмне забезпечення має відповідати певним вимогам щодо надійності, серед яких такі:

- захист від спроб завантаження файлів, тип яких не відповідає вимогам системи;
- захист від помилкових дій користувача, наприклад, у випадку введення невірних даних.

Додатково, програмне забезпечення повинно надійно працювати при безперервній роботі комп'ютера. У разі виникнення неполадок у роботі апаратної частини, відновлення нормальної функціональності програмного забезпечення має здійснюватися після перезавантаження системи.

A.4.3 Умови експлуатації

Умови експлуатації:

- програмне забезпечення не потребує обслуговуючого персоналу;
- для користування програмним забезпеченням не потрібна додаткова кваліфікація.

A.4.4 Вимоги до складу та параметрів технічних засобів

Вимоги до складу та параметрів технічних засобів:

- платформа x64-based;
- операційна система Microsoft Windows 11/10/8/7 (рекомендується

встановити всі останні оновлення);

- процесор з мінімальною тактовою частотою 1,8 ГГц, рекомендується використовувати як мінімум двоядерний процесор;
- оперативна пам'ять від 2 ГБ (рекомендується 4 ГБ та мінімум 2,5 ГБ при використанні на віртуальній машині);
- роздільна здатність екрану 1920x1080 пікселів з 16-бітовим кольором;
- мінімум 2 ГБ вільного місця на жорсткому диску.

A.4.5 Вимоги до маркування та упакування

Програмне забезпечення не потребує додаткових дій при маркуванні та упакуванні.

A.4.6 Вимоги до транспортування та збереження

Вимоги до транспортування й зберігання аналогічні вимогам, які пропонуються до носія, на якому записана програма.

A.5 Вимоги до програмної документації

Склад програмної документації повинен включати в себе:

- технічне завдання;
- керівництво програміста;
- керівництво оператора.

А.6 Основні обмеження на установку і використання програмного продукту

Розроблюваний програмний продукт потребує інсталяції. Програмний продукт повинен використовуватись на комп'ютерах які задовільняють вимогам та параметрам технічних засобів. Програмний продукт може використовуватись користувачем у будь-який момент часу, не конфліктуючи з іншими програмами.

ДОДАТОК Б
Текст програми

Б.1 Файл MainForm.cs

```

using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;
using RadiationMonitoringConsole;

namespace diploma
{
    public partial class MainForm : Form
    {
        private SensorMonitoringService _monitoringService;
        private const int MaxPoints = 50;
        private List<SensorReading> _currentReadings = new
List<SensorReading>();
        SensorConfig _sensor = new SensorConfig { };
        string connectionString =
"Server=localhost;Port=3306;Database=radiation_db;Uid=root;Pwd=root;";
        private SensorRepository _sensorRepo;
        Series greenLine = new Series("ThresholdLow");
        Series redLine = new Series("ThresholdHigh");
        bool monitoring = true;
        private List<SensorConfig> _sensors = new List<SensorConfig>();
        private FormEventLog _eventLogForm;

        public MainForm()

```

```

{
    this.WindowState = FormWindowState.Maximized;
    InitializeComponent();
}

private void Form1_Load(object sender, EventArgs e)
{
    _sensorRepo = new SensorRepository(connectionString);
    _monitoringService = new SensorMonitoringService();
    _sensorRepo.InsertEventLog("Програму моніторингу запущено");

    // Підписуємося на події
    _monitoringService.ReadingReceived += OnReadingReceived;
    _monitoringService.ThresholdExceeded += OnThresholdExceeded;

    numPeriodValue.Value = 1;
    numPeriodValue.Minimum = 1;
    numPeriodValue.Maximum = 100;
    cmbPeriodType.Items.Add("Хвилини");
    cmbPeriodType.Items.Add("Години");
    cmbPeriodType.Items.Add("Дні");
    cmbPeriodType.Items.Add("Довільний діапазон");
    cmbPeriodType.SelectedIndex = 1; // За замовчуванням "Години"
    dtFrom.Enabled = false;
    dtTo.Enabled = false;

    cmbSensorId.Items.Add("Датчик №");
    numPollingFrequency.Value = 1;
    numPollingFrequency.Minimum = 1;
    numPollingFrequency.Maximum = 100;

```

```

cmbPollingFrequency.Items.Add("Частота");
cmbPollingFrequency.Items.Add("Секунди");
cmbPollingFrequency.Items.Add("Хвилини");
cmbPollingFrequency.Items.Add("Години");

cmbOptions.Items.Add("Обрати опцію");
cmbOptions.Items.Add("Додати");
cmbOptions.Items.Add("Редагувати");
cmbOptions.Items.Add("Видалити");
ClearOption();
cmbSensorId.Enabled = false;
tbSensorName.Enabled = false;
tbSensorPlace.Enabled = false;
tbSensorModel.Enabled = false;
tbSensorMin.Enabled = false;
tbSensorMax.Enabled = false;
cmbPollingFrequency.Enabled = false;
numPollingFrequency.Enabled = false;
tbSensorComment.Enabled = false;

InitChart();

_sensors = _sensorRepo.GetAllSensors();
_sensor = _sensors[0];
textBox1.Text = AboutSensor(_sensor);
foreach (var sensor in _sensors)
{
    // Створюємо емулятор датчика
    var s1 = new FakeRadiationSensor(sensor.Id);

```

```

        // Додаємо у сервіс моніторингу
        _monitoringService.AddSensor(sensor, s1);
    }

    UpdateSensorComboBox();

    if (_sensors.Count > 0)
    {
        // вибираємо перший по Id
        _sensor = _sensors
            .OrderBy(s => int.Parse(s.Id))
            .First();

        // якщо треба – виставляємо вибір у комбобоксі
        cmbSensors.SelectedItem = _sensor;

        // завантажуюємо історію в буфер + малюємо
        LoadHistoryToBuffer(_sensor.Id);
    }
    else
    {
        InitChart(); // графік порожній, але підготовлений
    }
}

private void OnReadingReceived(SensorReading reading)
{
    if (InvokeRequired)
    {
        BeginInvoke(new Action(() => OnReadingReceived(reading)));
    }
}

```

```

        return;
    }

    string message = $"[{reading.Timestamp:HH:mm:ss}] Sensor
{reading.SensorId}: {reading.Value:F3} μSv/h";

    AppendColoredText(richTextBox1, message + Environment.NewLine,
Color.Black);

    _sensorRepo.InsertReading(reading);
    if (_sensor.Id == reading.SensorId)
    {
        if (_currentReadings == null)
            _currentReadings = new List<SensorReading>();

        if (_currentReadings.Count < MaxPoints)
        {
            _currentReadings.Add(reading);
        }
        else
        {
            // зсуваємо "вікно": прибираємо найстаріше, додаємо нове
            _currentReadings.RemoveAt(0);
            _currentReadings.Add(reading);
        }

        if(monitoring) DrawArray(_currentReadings);

        if (reading.Value <= _sensor.MinThreshold)
            textBox2.BackColor = Color.Green;
        else if (reading.Value > _sensor.MinThreshold && reading.Value

```

```

< _sensor.MaxThreshold)
    textBox2.BackColor = Color.Yellow;
else
    textBox2.BackColor = Color.Red;

    textBox2.Text = string.Format("{0:F3} µSv/h", reading.Value);
    textBox3.Text = string.Format("{0:dd.MM.yyyy HH:mm:ss}",
reading.Timestamp);
    }
}

private void OnThresholdExceeded(SensorReading reading, string
message)
{
    if (InvokeRequired)
    {
        BeginInvoke(new Action(() => OnThresholdExceeded(reading,
message)));
        return;
    }

    AppendColoredText(richTextBox1, message + Environment.NewLine,
Color.Red);

    // Вікно попередження:
    MessageBox.Show(this, message, "Warning",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
    _sensorRepo.InsertEventLog(message, reading.SensorId);
}

```

```

private void InitChart()
{
    chart.Series.Clear();
    chart.ChartAreas.Clear();

    var area = new ChartArea("MainArea");
    area.AxisX.Title = "Час";
    area.AxisY.Title = "Radiation ( $\mu\text{Sv/h}$ )";

    // Підпис по осі X як час
    area.AxisX.LabelStyle.Format = "HH:mm:ss";

    area.AxisX.MajorGrid.Enabled = false;
    area.AxisX.MinorGrid.Enabled = false;
    area.AxisY.MajorGrid.Enabled = false;
    area.AxisY.MinorGrid.Enabled = false;

    chart.ChartAreas.Add(area);

    // ===== Основний графік =====
    var mainSeries = new Series("Radiation");
    mainSeries.ChartType = SeriesChartType.Line;
    mainSeries.ChartArea = "MainArea";
    mainSeries.Color = Color.Blue;
    mainSeries.XValueType = ChartValueType.DateTime;
    chart.Series.Add(mainSeries);

    // ===== Зелена лінія порогу =====
    greenLine = new Series("ThresholdLow");
    greenLine.ChartType = SeriesChartType.Line;

```

```

greenLine.ChartArea = "MainArea";
greenLine.Color = Color.Green;
greenLine.BorderWidth = 2;
greenLine.XValueType = ChartValueType.DateTime;

chart.Series.Add(greenLine);

// ===== Червона лінія порогу =====
redLine = new Series("ThresholdHigh");
redLine.ChartType = SeriesChartType.Line;
redLine.ChartArea = "MainArea";
redLine.Color = Color.Red;
redLine.BorderWidth = 2;
redLine.XValueType = ChartValueType.DateTime;

chart.Series.Add(redLine);

chart.ChartAreas["MainArea"].RecalculateAxesScale();
}

private void UpdateThresholdLines(DateTime xMin, DateTime xMax)
{
    greenLine.Points.Clear();
    greenLine.Points.AddXY(xMin, _sensor.MinThreshold);
    greenLine.Points.AddXY(xMax, _sensor.MinThreshold);

    redLine.Points.Clear();
    redLine.Points.AddXY(xMin, _sensor.MaxThreshold);
    redLine.Points.AddXY(xMax, _sensor.MaxThreshold);
}

```

```
private void DrawArray(List<SensorReading> data)
{
    // захист: якщо серії ще нема – ініціалізуємо графік
    if (chart.Series.IndexOf("Radiation") == -1)
    {
        InitChart();
    }

    var s = chart.Series["Radiation"];
    s.Points.Clear();

    if (data == null || data.Count == 0)
    {
        chart.ChartAreas["MainArea"].RecalculateAxesScale();
        return;
    }

    // додаємо точки: X = Timestamp, Y = Value
    foreach (var r in data)
    {
        s.Points.AddXY(r.Timestamp, r.Value);
    }

    // порогові лінії по часу
    DateTime xMin = data.First().Timestamp;
    DateTime xMax = data.Last().Timestamp;

    UpdateThresholdLines(xMin, xMax);
}
```

```

        chart.ChartAreas["MainArea"].RecalculateAxesScale();
    }

```

```

private void AppendColoredText(RichTextBox box, string text, Color
color)
{
    box.SelectionStart = box.TextLength;
    box.SelectionLength = 0;

    box.SelectionColor = color;
    box.AppendText(text);
    box.SelectionColor = box.ForeColor;

    // Автопрокрутка вниз
    box.SelectionStart = box.TextLength;
    box.ScrollToCaret();
}

```

```

private void cmbPeriodType_SelectedIndexChanged(object sender,
EventArgs e)
{
    bool isCustomRange = cmbPeriodType.SelectedItem != null &&
        cmbPeriodType.SelectedItem.ToString() == "Довільний
діапазон";

    numPeriodValue.Enabled = !isCustomRange; // для
хв/год/днів/місяців
    dtFrom.Enabled = isCustomRange;
    dtTo.Enabled = isCustomRange;
}

```

```
}
```

```
private void Form1_FormClosing(object sender, FormClosingEventArgs
e)
```

```
{
    _sensorRepo.InsertEventLog("Програму моніторингу закрито");

    if (_monitoringService != null)
    {
        _monitoringService.StopAll();
    }
}
```

```
private void btnSensorApply_Click(object sender, EventArgs e)
```

```
{
    if (ValidateSensorForm() == 0)
        return;
    if (ValidateSensorForm() == 1)
    {
        var newSensor = new SensorConfig();

        int newId = GetNextSensorId(_sensors);
        newSensor.Id = newId.ToString();
        newSensor.Name = tbSensorName.Text;
        newSensor.Place = tbSensorPlace.Text;
        newSensor.Model = tbSensorModel.Text;
        newSensor.MinThreshold =
Convert.ToDouble(tbSensorMin.Text);
        newSensor.MaxThreshold =
Convert.ToDouble(tbSensorMax.Text);
    }
}
```

```

newSensor.Status = "Активний";
newSensor.LastValue = 0;
newSensor.LastUpdate = DateTime.Now;
newSensor.Start = DateTime.Now;
newSensor.Smh = cmbPollingFrequency.SelectedIndex;

newSensor.Comment = (tbSensorComment.Text == "" ||
tbSensorComment.Text == "Коментар")
    ? "Відсутній"
    : tbSensorComment.Text;

double pollingFrequency =
Convert.ToDouble(numPollingFrequency.Value);

if (cmbPollingFrequency.SelectedIndex == 1)
    newSensor.PollingInterval =
(TimeSpan.FromSeconds(pollingFrequency));
else if (cmbPollingFrequency.SelectedIndex == 2)
    newSensor.PollingInterval =
(TimeSpan.FromMinutes(pollingFrequency));
else
    newSensor.PollingInterval =
(TimeSpan.FromHours(pollingFrequency));

MessageBox.Show("Датчик" + newSensor.Id + ": " +
newSensor.Name + " успішно додано!", "OK", MessageBoxButtons.OK,
MessageBoxIcon.Information);

_sensorRepo.InsertEventLog($"Датчик №{newSensor.Id}:
{newSensor.Name} успішно додано!", newSensor.Id);

_sensorRepo.InsertEventLog("Додано:\r\n" +

```

```

AboutSensor(newSensor), newSensor.Id);

    try
    {
        _sensorRepo.InsertSensor(newSensor);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Помилка запису в БД:\n" + ex.Message,
            "Помилка",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        _sensorRepo.InsertEventLog("Помилка запису в БД при створенні.", newSensor.Id);
        return;
    }

    // симуляція
    var fake = new FakeRadiationSensor(newSensor.Id);
    _monitoringService.AddSensor(newSensor, fake);

    // додаємо в локальний список
    _sensors.Add(newSensor);
    _sensors = _sensors.OrderBy(s => int.Parse(s.Id)).ToList();

    // оновлюємо комбобокси (без Items.Add!)
    UpdateSensorComboBox();

    ClearOption();
    //monitoring = true;
    return;
}

```

```

if (ValidateSensorForm() == 2)
{
    if (ValidateAddEdit())
    {
        SensorConfig          sensor          =
(SensorConfig)cmbSensorId.SelectedValue;

        sensor.Name = tbSensorName.Text;
        sensor.Place = tbSensorPlace.Text;
        sensor.Model = tbSensorModel.Text;
        sensor.MinThreshold = Convert.ToDouble(tbSensorMin.Text);
        sensor.MaxThreshold          =
Convert.ToDouble(tbSensorMax.Text);
        sensor.Comment = tbSensorComment.Text;

        // Обробляємо smh + PollingInterval
        string          freqType          =
cmbPollingFrequency.SelectedItem.ToString();
        int value = (int)numPollingFrequency.Value;

        int smh;
        int totalSeconds;

        switch (freqType)
        {
            case "Секунди":
                smh = 1;
                totalSeconds = value;
                break;
            case "Хвилини":
                smh = 2;

```

```

        totalSeconds = value * 60;
        break;
    case "Години":
        smh = 3;
        totalSeconds = value * 3600;
        break;
    default:
        MessageBox.Show("Невідомий тип інтервалу.");
        return;
    }

    sensor.Smh = smh;
    sensor.PollingInterval =
    TimeSpan.FromSeconds(totalSeconds);

    // Оновлюємо в БД
    _sensorRepo.UpdateSensor(sensor);

    // Оновлюємо відображення в комбобоксі (якщо там
    показується Name/Place)
    UpdateSensorComboBox();

    // перезапускаємо датчик у сервісі моніторингу
    _monitoringService.RemoveSensor(sensor.Id);
    var newFakeSensor = new FakeRadiationSensor(sensor.Id);
    _monitoringService.AddSensor(sensor, newFakeSensor);

    MessageBox.Show("Датчик" + sensor.Id + ": " + sensor.Name
    + " успішно відредаговано!", "OK", MessageBoxButtons.OK,
    MessageBoxIcon.Information);

```

```

        _sensorRepo.InsertEventLog("Датчик №" + sensor.Id + ": " +
sensor.Name + " успішно відредаговано!", sensor.Id);
        _sensorRepo.InsertEventLog("Відредаговано:\r\n"
AboutSensor(sensor), sensor.Id);

        if (_sensor.Id == sensor.Id) textBox1.Text =
AboutSensor(sensor);
        ClearOption();
        //monitoring = true;
    }
    return;
}
if (ValidateSensorForm() == 3)
{
    if (cmbSensorId.SelectedValue == null)
    {
        MessageBox.Show("Виберіть датчик для видалення.");
        return;
    }
    SensorConfig sensor =
(SensorConfig)cmbSensorId.SelectedValue;
    string id = sensor.Id;

    // 1. Деактивуємо записи зчитувань
    _sensorRepo.DeactivateSensorReadings(id);
    // 2. Видаляємо датчик з таблиці sensors
    _sensorRepo.DeleteSensor(id);
    // 3. Видаляємо датчик із локального списку
    _sensors.Remove(sensor);

```

```

        MessageBox.Show("Датчик" + sensor.Id + ": " + sensor.Name + "
успішно видалено!", "OK", MessageBoxButtons.OK,
MessageBoxIcon.Information);

        _sensorRepo.InsertEventLog("Датчик №" + sensor.Id + ": " +
sensor.Name + " успішно видалено!", sensor.Id);

        _sensorRepo.InsertEventLog("Видалено:\r\n" +
AboutSensor(sensor), sensor.Id);

        ClearOption();
        UpdateSensorComboBox();
        //monitoring = true;
        return;
    }
}

```

```

private int GetNextSensorId(List<SensorConfig> sensors)
{
    // Збираємо всі існуючі числові ID
    var ids = sensors
        .Select(s => int.TryParse(s.Id, out int x) ? x : -1)
        .Where(x => x > 0)
        .OrderBy(x => x)
        .ToList();

    // Якщо датчиків немає → повертаємо 1
    if (ids.Count == 0)
        return 1;

    // Знаходимо першу "дірку"
    int expected = 1;
    foreach (int id in ids)

```

```

{
    if (id != expected)
        return expected;
    expected++;
}

```

```

// Якщо дірок немає → повертаємо наступний ID
return expected;
}

```

```

private void UpdateSensorComboBox()
{
    cmbSensors.DataSource = null; // скидуємо
    cmbSensors.DataSource = _sensors
        .OrderBy(s => int.Parse(s.Id))
        .Select(s => new { Display = $"{s.Id} | {s.Name} | {s.Place}",
Sensor = s })
        .ToList();

    cmbSensors.DisplayMember = "Display";
    cmbSensors.ValueMember = "Sensor";

    cmbSensorId.DataSource = null; // скидуємо
    cmbSensorId.DataSource = _sensors
        .OrderBy(s => int.Parse(s.Id))
        .Select(s => new { Display = $"{s.Id}", Sensor = s })
        .ToList();

    cmbSensorId.DisplayMember = "Display";
    cmbSensorId.ValueMember = "Sensor";
}

```

```
}
```

```
private void LoadSensorToEditForm(SensorConfig sensor)
```

```
{
```

```
    if (sensor == null) return;
```

```
    tbSensorName.Text = sensor.Name;
```

```
    tbSensorPlace.Text = sensor.Place;
```

```
    tbSensorModel.Text = sensor.Model;
```

```
    tbSensorMin.Text = sensor.MinThreshold.ToString();
```

```
    tbSensorMax.Text = sensor.MaxThreshold.ToString();
```

```
    tbSensorComment.Text = sensor.Comment;
```

```
    // обробка частоти опитування залежно від smh
```

```
    double seconds = sensor.PollingInterval.TotalSeconds;
```

```
    switch (sensor.Smh)
```

```
    {
```

```
        case 1: // секунди
```

```
            numPollingFrequency.Value = (decimal)seconds;
```

```
            cmbPollingFrequency.SelectedItem = "Секунди";
```

```
            break;
```

```
        case 2: // хвилини
```

```
            numPollingFrequency.Value = (decimal)(seconds / 60.0);
```

```
            cmbPollingFrequency.SelectedItem = "Хвилини";
```

```
            break;
```

```
        case 3: // години
```

```
            numPollingFrequency.Value = (decimal)(seconds / 3600.0);
```

```

        cmbPollingFrequency.SelectedItem = "Години";
        break;

    default:
        // якщо щось дивне – вважаємо секунди
        numPollingFrequency.Value = (decimal)seconds;
        cmbPollingFrequency.SelectedItem = "Секунди";
        break;
    }
}

private void cmbSensorId_SelectedIndexChanged(object sender,
EventArgs e)
{
    if (cmbOptions.SelectedItem.ToString() == "Редагувати")
    {
        if (cmbSensorId.SelectedItem == null)
        {
            SensorConfig sensor = _sensors[0];
            LoadSensorToEditForm(sensor);
        }
        else
        {
            SensorConfig sensor =
(SensorConfig)cmbSensorId.SelectedValue;
            LoadSensorToEditForm(sensor);
        }
    }
}

```

e)

```

private void cmbSensors_SelectedIndexChanged(object sender, EventArgs
{
    monitoring = true;
    // якщо нічого не вибрано – виходимо
    if (cmbSensors.SelectedValue == null)
        return;

    // пробуємо привести до SensorConfig
    var selectedSensor = cmbSensors.SelectedValue as SensorConfig;
    if (selectedSensor == null)
        return;

    _sensor = selectedSensor;

    // завантажуюмо історію саме цього сенсора в той самий масив
    LoadHistoryToBuffer(_sensor.Id);
    textBox1.Text = AboutSensor(_sensor);
}

private void LoadHistoryToBuffer(string sensorId)
{
    var readings = _sensorRepo
        .GetLastReadings(sensorId, MaxPoints)
        .OrderBy(r => r.Timestamp)
        .ToList();

    _currentReadings = readings;
}

```

```

        DrawArray(_currentReadings);
    }

    private void btnMonitoring_Click(object sender, EventArgs e)
    {
        // Повертаємось в режим реального часу
        monitoring = true;

        // Беремо обраний сенсор з комбобокса (бо
        ValueMember="Sensor")
        var selected = cmbSensors.SelectedValue as SensorConfig;
        if (selected == null)
        {
            MessageBox.Show("Оберіть датчик.", "Увага",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }

        _sensor = selected; // оновлюємо поточний сенсор

        // Завантажуємо останні 50 (активні) з БД і малюємо
        LoadHistoryToBuffer(_sensor.Id);
    }

    private void btnEventLog_Click(object sender, EventArgs e)
    {
        // якщо ще не створювали або вже закрили – створюємо заново
        if (_eventLogForm == null || _eventLogForm.IsDisposed)
        {
            _eventLogForm = new FormEventLog();
        }
    }

```

```

        _eventLogForm.FormClosed += (s, args) => _eventLogForm =
null; // щоб поле очистилось
        _eventLogForm.Show(this); // this = власник
    }
    else
    {
        // якщо згорнута – розгортаємо
        if (_eventLogForm.WindowState ==
FormWindowState.Minimized)
            _eventLogForm.WindowState = FormWindowState.Normal;

        // виводимо на передній план
        _eventLogForm.Activate();
        _eventLogForm.BringToFront();
    }
}

private void btnApplyPeriod_Click(object sender, EventArgs e)
{
    monitoring = false; // зупиняємо realtime-оновлення графіка

    // 1) беремо обраний датчик
    if (cmbSensors.SelectedValue == null)
    {
        MessageBox.Show("Оберіть датчик.", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }

    var selectedSensor = cmbSensors.SelectedValue as SensorConfig;

```

```

if (selectedSensor == null)
{
    MessageBox.Show("Не вдалося визначити датчик.",
"Помилка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    return;
}

_sensor = selectedSensor; // щоб пороги малювались від
правильного датчика

// 2) визначаємо період
DateTime to = DateTime.Now;
DateTime from;

string periodType = cmbPeriodType.SelectedItem?.ToString() ?? "";

if (periodType == "Довільний діапазон")
{
    from = dtFrom.Value;
    to = dtTo.Value;

    // перевірка: "з" не більше "по"
    if (from > to)
    {
        MessageBox.Show("Дата 'З' не може бути більшою за дату
'По'." , "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }
}
else

```

```

{
    int n = (int)numPeriodValue.Value;

    switch (periodType)
    {
        case "Хвилини":
            from = DateTime.Now.AddMinutes(-n);
            to = DateTime.Now;
            break;

        case "Години":
            from = DateTime.Now.AddHours(-n);
            to = DateTime.Now;
            break;

        case "Дні":
            from = DateTime.Now.AddDays(-n);
            to = DateTime.Now;
            break;

        default:
            MessageBox.Show("Оберіть тип періоду.", "Помилка",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
    }
}

// 3) зчитуємо дані з БД (тільки Active = true)
var readings = _sensorRepo.GetReadingsByPeriod(_sensor.Id, from,
to);

```

// 4) якщо нічого нема – очищаємо графік, але пороги можна не малювати

```
if (readings.Count == 0)
{
    _currentReadings = new List<SensorReading>();
    DrawArray(_currentReadings);
    MessageBox.Show("За обраний період даних немає.",
"Інформація", MessageBoxButtons.OK, MessageBoxIcon.Information);
    return;
}

// 5) підставляємо у джерело графіка і малюємо
_currentReadings = readings;
DrawArray(_currentReadings);
}
}
}
```

B.2 FormEventLog.cs

```

using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using RadiationMonitoringConsole;

namespace diploma
{
    public partial class FormEventLog : Form
    {
        private List<SensorConfig> _sensors = new List<SensorConfig>();
        string connectionString =
"Server=localhost;Port=3306;Database=radiation_db;Uid=root;Pwd=root;";
        private SensorRepository _repo;
        private bool _internalCheckChange = false; // щоб уникнути зациклення

        public FormEventLog()
        {
            InitializeComponent();
            _repo = new SensorRepository(connectionString);
        }

        private void FormEventLog_Load(object sender, EventArgs e)
        {
            LoadSensorsFromDb();
            FillCheckedListBox();
        }
    }

```

```

checkedListBox1.SetItemChecked(0, true);
// NumericUpDown
numPeriodValue.Minimum = 1;
numPeriodValue.Maximum = 100;
numPeriodValue.Value = 1;

// ComboBox: період
cmbPeriodType.Items.Clear();
cmbPeriodType.Items.Add("Хвилини");
cmbPeriodType.Items.Add("Години");
cmbPeriodType.Items.Add("Дні");
cmbPeriodType.Items.Add("Довільний діапазон");

cmbPeriodType.SelectedIndexChanged                                     +=
cmbPeriodType_SelectedIndexChanged;

// Дефолт: "Годин"
cmbPeriodType.SelectedIndex = 1;

// На старті застосовуємо правильне блокування
ApplyPeriodUiState();
}

private void LoadSensorsFromDb()
{
    _sensors = _repo.GetAllSensors()
        .OrderBy(s => int.Parse(s.Id))
        .ToList();
}

```

```

private void FillCheckedListBox()
{
    checkedListBox1.Items.Clear();

    // 0-й елемент – "Усі"
    checkedListBox1.Items.Add("Усі", false);

    // далі всі датчики
    foreach (var s in _sensors)
    {
        checkedListBox1.Items.Add(s, false);
    }
}

private void btnApply_Click(object sender, EventArgs e)
{
    // 1) період (датчики враховуємо завжди)
    DateTime from;
    DateTime to;

    string periodType = cmbPeriodType.SelectedItem?.ToString() ?? "";

    if (periodType == "Довільний діапазон")
    {
        from = dtFrom.Value;
        to = dtTo.Value;

        if (from > to)
        {

```

```

        MessageBox.Show("Дата 'З' не може бути більшою за дату
        'По'.", "Помилка",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }
}
else
{
    int n = (int)numPeriodValue.Value;
    to = DateTime.Now;

    switch (periodType)
    {
        case "Хвилини":
            from = to.AddMinutes(-n);
            break;
        case "Години":
            from = to.AddHours(-n);
            break;
        case "Дні":
            from = to.AddDays(-n);
            break;
        default:
            MessageBox.Show("Оберіть тип періоду.", "Помилка",
                MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
    }
}

// 2) датчики

```

```

bool allChecked = checkedListBox1.Items.Count > 0 &&
checkedListBox1.GetItemChecked(0);

```

```

var selectedSensorIds = checkedListBox1.CheckedItems
    .Cast<object>()
    .OfType<SensorConfig>()
    .Select(s => s.Id)
    .Distinct()
    .ToList();

```

```

// Правила:
// - 0 обрано -> всі + NULL
// - "Усі" -> всі без NULL
// - деякі -> деякі без NULL

```

```

List<string> filterIds;
bool includeNull;

```

```

if (selectedSensorIds.Count == 0 && !allChecked)
{
    // нічого не обрано
    filterIds = null;           // всі
    includeNull = true;        // + NULL
}
else if (allChecked)
{
    // обрано "Усі"
    filterIds = null;           // всі
    includeNull = false;       // без NULL
}

```

```

else
{
    // обрані деякі
    filterIds = selectedSensorIds; // тільки вони
    includeNull = false;          // без NULL
}

// 3) читаємо з БД
var events = _repo.GetEventLogByPeriod(from, to, filterIds,
includeNull);

// 4) вивід
if (events.Count == 0)
{
    textBox1.Text = "За обраний період подій немає.";
    return;
}

var sb = new StringBuilder();
foreach (var ev in events)
{
    string sid = string.IsNullOrEmpty(ev.SensorId) ? "-" :
ev.SensorId;
    sb.AppendLine($"[{ev.Timestamp:dd.MM.yyyy HH:mm:ss}]
Sensor {sid}: {ev.Message}");
}

textBox1.Text = sb.ToString();
}

```

```
private void checkedListBox1_ItemCheck(object sender,
ItemCheckEventArgs e)
{
    if (_internalCheckChange)
        return;

    _internalCheckChange = true;

    try
    {
        // ===== КЛІК ПО "УСІ" =====
        if (e.Index == 0)
        {
            bool checkAll = e.NewValue == CheckState.Checked;

            // якщо "Усі" → ставимо/знімаємо всі інші
            for (int i = 1; i < checkedListBox1.Items.Count; i++)
            {
                checkedListBox1.SetItemChecked(i, checkAll);
            }
        }
        else
        {
            // ===== КЛІК ПО ОКРЕМОМУ ДАТЧИКУ =====

            if (e.NewValue == CheckState.Unchecked)
            {
                // якщо зняли хоча б один – "Усі" знімаємо
                checkedListBox1.SetItemChecked(0, false);
            }
        }
    }
}
```

```

else
{
    // перевіряємо: чи тепер ВСІ датчики відмічені
    bool allChecked = true;

    for (int i = 1; i < checkedListBox1.Items.Count; i++)
    {
        if (i == e.Index)
            continue;

        if (!checkedListBox1.GetItemChecked(i))
        {
            allChecked = false;
            break;
        }
    }

    if (allChecked)
    {
        checkedListBox1.SetItemChecked(0, true);
    }
    }
}

finally
{
    _internalCheckChange = false;
}
}

```

```

private void cmbPeriodType_SelectedIndexChanged(object sender,
EventArgs e)
{
    ApplyPeriodUiState();
}

private void ApplyPeriodUiState()
{
    bool isCustomRange = cmbPeriodType.SelectedItem != null &&
        cmbPeriodType.SelectedItem.ToString() ==
        "Довільний діапазон";

    // Якщо "Довільний діапазон" → вмикаємо DateTimePicker-и,
    вимикаємо число
    dtFrom.Enabled = isCustomRange;
    dtTo.Enabled = isCustomRange;

    numPeriodValue.Enabled = !isCustomRange;
}
}
}

```

Б.3 EventLogEntry.cs

```
using System;

namespace diploma
{
    public class EventLogEntry
    {
        public int Id { get; set; }
        public string SensorId { get; set; } // може бути null
        public string Message { get; set; }
        public DateTime Timestamp { get; set; }
    }
}
```

Б.4 FakeRadiationSensor.cs

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace RadiationMonitoringConsole
{
    public class FakeRadiationSensor : IRadiationSensor
    {
        private readonly Random _random = new Random();

        public string Id { get; }

        public FakeRadiationSensor(string id)
        {
            Id = id;
        }
    }
}
```

```

    {
        Id = id;
    }

    public Task<double> ReadAsync(CancellationTokен cancellationToken =
default)
    {
        // Емуляція: випадкове значення від 0.03 до 0.50  $\mu\text{Sv/h}$ 
        double value = 0.03 + _random.NextDouble() * 0.47;
        return Task.FromResult(value);
    }
}

```

B.5 ForDesign.cs

```

using RadiationMonitoringConsole;
using System;
using System.Drawing;
using System.Windows.Forms;

namespace diploma
{
    public partial class MainForm : Form
    {
        private void ClearOption()
        {
            cmbPollingFrequency.SelectedIndex = 0;
            cmbOptions.SelectedIndex = 0;
            cmbSensorId.SelectedIndex = 0;

```

```

tbSensorName.Text = "Назва";
tbSensorName.ForeColor = Color.Gray;
tbSensorPlace.Text = "Місце встановлення";
tbSensorPlace.ForeColor = Color.Gray;
tbSensorModel.Text = "Модель";
tbSensorModel.ForeColor = Color.Gray;
tbSensorMin.Text = "Min";
tbSensorMin.ForeColor = Color.Gray;
tbSensorMax.Text = "Max";
tbSensorMax.ForeColor = Color.Gray;
tbSensorComment.Text = "Коментар";
tbSensorComment.ForeColor = Color.Gray;
numPollingFrequency.Value = 1;
cmbSensorId.Enabled = true;
tbSensorName.Enabled = true;
tbSensorPlace.Enabled = true;
tbSensorModel.Enabled = true;
tbSensorMin.Enabled = true;
tbSensorMax.Enabled = true;
cmbPollingFrequency.Enabled = true;
numPollingFrequency.Enabled = true;
tbSensorComment.Enabled = true;
}

private void tbSensorName_Enter(object sender, EventArgs e)
{
    if (tbSensorName.ForeColor == Color.Gray)
    {
        tbSensorName.Text = "";
        tbSensorName.ForeColor = Color.Black;
    }
}

```

```
    }  
}  
  
private void tbSensorName_Leave(object sender, EventArgs e)  
{  
    if (string.IsNullOrEmpty(tbSensorName.Text))  
    {  
        tbSensorName.Text = "Назва";  
        tbSensorName.ForeColor = Color.Gray;  
    }  
}  
  
private void tbSensorPlace_Enter(object sender, EventArgs e)  
{  
    if (tbSensorPlace.ForeColor == Color.Gray)  
    {  
        tbSensorPlace.Text = "";  
        tbSensorPlace.ForeColor = Color.Black;  
    }  
}  
  
private void tbSensorPlace_Leave(object sender, EventArgs e)  
{  
    if (string.IsNullOrEmpty(tbSensorPlace.Text))  
    {  
        tbSensorPlace.Text = "Місце встановлення";  
        tbSensorPlace.ForeColor = Color.Gray;  
    }  
}
```

```
private void tbSensorModel_Enter(object sender, EventArgs e)
{
    if (tbSensorModel.ForeColor == Color.Gray)
    {
        tbSensorModel.Text = "";
        tbSensorModel.ForeColor = Color.Black;
    }
}
```

```
private void tbSensorModel_Leave(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(tbSensorModel.Text))
    {
        tbSensorModel.Text = "Модель";
        tbSensorModel.ForeColor = Color.Gray;
    }
}
```

```
private void tbSensorMin_Enter(object sender, EventArgs e)
{
    if (tbSensorMin.ForeColor == Color.Gray)
    {
        tbSensorMin.Text = "";
        tbSensorMin.ForeColor = Color.Black;
    }
}
```

```
private void tbSensorMin_Leave(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(tbSensorMin.Text))
```

```
{
    tbSensorMin.Text = "Min";
    tbSensorMin.ForeColor = Color.Gray;
}

private void tbSensorMax_Enter(object sender, EventArgs e)
{
    if (tbSensorMax.ForeColor == Color.Gray)
    {
        tbSensorMax.Text = "";
        tbSensorMax.ForeColor = Color.Black;
    }
}

private void tbSensorMax_Leave(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(tbSensorMax.Text))
    {
        tbSensorMax.Text = "Max";
        tbSensorMax.ForeColor = Color.Gray;
    }
}

private void tbSensorComment_Enter(object sender, EventArgs e)
{
    if (tbSensorComment.ForeColor == Color.Gray)
    {
        tbSensorComment.Text = "";
        tbSensorComment.ForeColor = Color.Black;
    }
}
```

```
    }
}
```

```
private void tbSensorComment_Leave(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(tbSensorComment.Text))
    {
        tbSensorComment.Text = "Коментар";
        tbSensorComment.ForeColor = Color.Gray;
    }
}
```

```
e)
private void cmbOptions_SelectedIndexChanged(object sender, EventArgs
{
    if (cmbOptions.SelectedIndex == 0)
    {
        cmbSensorId.Enabled = false;
        tbSensorName.Enabled = false;
        tbSensorPlace.Enabled = false;
        tbSensorModel.Enabled = false;
        tbSensorMin.Enabled = false;
        tbSensorMax.Enabled = false;
        cmbPollingFrequency.Enabled = false;
        numPollingFrequency.Enabled = false;
        tbSensorComment.Enabled = false;
    }
    if (cmbOptions.SelectedItem.ToString() == "Додати")
    {
        cmbSensorId.Enabled = false;
```

```
tbSensorName.Enabled = true;
tbSensorPlace.Enabled = true;
tbSensorModel.Enabled = true;
tbSensorMin.Enabled = true;
tbSensorMax.Enabled = true;
cmbPollingFrequency.Enabled = true;
numPollingFrequency.Enabled = true;
tbSensorComment.Enabled = true;
}
if (cmbOptions.SelectedItem.ToString() == "Редагувати")
{
    tbSensorName.ForeColor = Color.Black;
    tbSensorPlace.ForeColor = Color.Black;
    tbSensorModel.ForeColor = Color.Black;
    tbSensorMin.ForeColor = Color.Black;
    tbSensorMax.ForeColor = Color.Black;
    tbSensorComment.ForeColor = Color.Black;
    SensorConfig sensor = _sensors[0];
    LoadSensorToEditForm(sensor);

    cmbSensorId.Enabled = true;
    tbSensorName.Enabled = true;
    tbSensorPlace.Enabled = true;
    tbSensorModel.Enabled = true;
    tbSensorMin.Enabled = true;
    tbSensorMax.Enabled = true;
    cmbPollingFrequency.Enabled = true;
    numPollingFrequency.Enabled = true;
    tbSensorComment.Enabled = true;
}
```

```

if (cmbOptions.SelectedItem.ToString() == "Видалити")
{
    cmbSensorId.Enabled = true;
    tbSensorName.Enabled = false;
    tbSensorPlace.Enabled = false;
    tbSensorModel.Enabled = false;
    tbSensorMin.Enabled = false;
    tbSensorMax.Enabled = false;
    cmbPollingFrequency.Enabled = false;
    numPollingFrequency.Enabled = false;
    tbSensorComment.Enabled = false;
}
}

private int ValidateSensorForm()
{
    // Перевірка опції
    if (cmbOptions.SelectedIndex == 0)
    {
        MessageBox.Show("Будь ласка, оберіть опцію.", "Помилка",
        MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return 0;
    }

    else if (cmbOptions.SelectedIndex == 1) // Додати опція
    {
        if (!ValidateAddEdit())
            return 0;
        return 1;
    }
}

```

```

else if (cmbOptions.SelectedIndex == 2) // Редагувати опція
{
    return 2;
}

else // Видалити опція
{
    return 3;
}
}

private bool ValidateAddEdit()
{
    if (string.IsNullOrEmpty(tbSensorName.Text) ||
    tbSensorName.Text == "Назва")
    {
        MessageBox.Show("Поле 'Назва' не може бути порожнім.",
        "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return false;
    }

    if (string.IsNullOrEmpty(tbSensorPlace.Text) ||
    tbSensorPlace.Text == "Місце встановлення")
    {
        MessageBox.Show("Поле 'Місце встановлення' не може бути
        порожнім.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return false;
    }
}

```

```
if (string.IsNullOrEmpty(tbSensorModel.Text) ||
tbSensorModel.Text == "Модель")
{
    MessageBox.Show("Поле 'Модель' не може бути порожнім.",
"Помилка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    return false;
}

// Перевірка Min
if (!double.TryParse(tbSensorMin.Text, out double minValue))
{
    MessageBox.Show("Поле 'Min' повинно містити число.",
"Помилка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    return false;
}

// Перевірка Max
if (!double.TryParse(tbSensorMax.Text, out double maxValue))
{
    MessageBox.Show("Поле 'Max' повинно містити число.",
"Помилка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    return false;
}

if (minValue >= maxValue)
{
    MessageBox.Show("Значення 'Min' повинно бути меншим за
'Max'.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    return false;
}
```

```

// Перевірка частоти
if (cmbPollingFrequency.SelectedIndex == 0)
{
    MessageBox.Show("Будь ласка, оберіть тип інтервалу  
(секунди/хвилини/години).", "Помилка", MessageBoxButtons.OK,  
MessageBoxIcon.Warning);
    return false;
}
return true;
}

private string AboutSensor(SensorConfig sensor)
{
    double sec = sensor.PollingInterval.TotalSeconds;
    string periodText;
    string text;

    switch (sensor.Smh)
    {
        case 1:
            periodText = $"{sec} секунд";
            break;

        case 2:
            periodText = $"{sec / 60.0:0.##} хвилин";
            break;

        case 3:
            periodText = $"{sec / 3600.0:0.##} годин";

```

```

        break;

        default:
            periodText = $"{sec} секунд";
            break;
    }

    text =
        $"ID: {sensor.Id}\r\n" +
        $"Назва: {sensor.Name}\r\n" +
        $"Місце встановлення: {sensor.Place}\r\n" +
        $"Модель: {sensor.Model}\r\n" +
        $"Мінімальний поріг: {sensor.MinThreshold:F3}\r\n" +
        $"Максимальний поріг: {sensor.MaxThreshold:F3}\r\n" +
        $"Період опитування: {periodText}\r\n" +
        $"Коментар: {sensor.Comment}\r\n" +
        $"Статус: {sensor.Status}\r\n" +
        $"Початок роботи: {sensor.Start:dd.MM.yyyy HH:mm:ss}";

    return text;
}
}
}

```

Б.6 IRadiationSensor.cs

```

using System.Threading;
using System.Threading.Tasks;

namespace RadiationMonitoringConsole

```

```

{
    public interface IRadiationSensor
    {
        string Id { get; }
        Task<double> ReadAsync(CancellationToken cancellationToken =
default);
    }
}

```

B.7 SensorConfig.cs

```

using System;

namespace RadiationMonitoringConsole
{
    public class SensorConfig
    {
        public string Id { get; set; } = "";
        public string Name { get; set; } = "";
        public string Place { get; set; } = "";
        public string Model { get; set; } = "";
        public double MinThreshold { get; set; }
        public double MaxThreshold { get; set; }
        public TimeSpan PollingInterval { get; set; } =
TimeSpan.FromSeconds(10);
        public int Smh { get; set; }
        public string Comment { get; set; } = "";
        public string Status { get; set; } = "";
        public double LastValue { get; set; }
        public DateTime LastUpdate { get; set; }
    }
}

```

```

public DateTime Start { get; set; }

public override string ToString()
{
    return $"{Id} | {Name} | {Place}";
}

}

}

```

B.8 SensorMonitoringService.cs

```

using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;

namespace RadiationMonitoringConsole
{
    public class SensorMonitoringService
    {
        // Словник: ID датчика -> (конфіг, сам датчик, токен відміни)
        private readonly Dictionary<string, (SensorConfig config,
        IRadiationSensor sensor, CancellationTokenSource cts)> _sensors
            = new Dictionary<string, (SensorConfig config, IRadiationSensor
        sensor, CancellationTokenSource cts)>();

        // Подія: нове значення від будь-якого датчика
        public event Action<SensorReading> ReadingReceived;
    }
}

```

```

// Подія: перевищення порогу
public event Action<SensorReading, string> ThresholdExceeded;

public void AddSensor(SensorConfig config, IRadiationSensor sensor)
{
    if (config.Id != sensor.Id)
        throw new ArgumentException("SensorConfig.Id must match
sensor.Id");

    if (_sensors.ContainsKey(config.Id))
        throw new InvalidOperationException("Sensor with ID " +
config.Id + " already exists");

    var cts = new CancellationTokensource();
    _sensors[config.Id] = (config, sensor, cts);

    // Запускаємо моніторинг цього датчика в окремому taskу
    Task.Run(() => MonitorSensorAsync(config, sensor, cts.Token));
}

public void RemoveSensor(string sensorId)
{
    (SensorConfig config, IRadiationSensor sensor,
CancellationTokensource cts) entry;
    if (_sensors.TryGetValue(sensorId, out entry))
    {
        entry.cts.Cancel();
        _sensors.Remove(sensorId);
    }
}

```

```

    }

    public void StopAll()
    {
        foreach (var kvp in _sensors)
        {
            kvp.Value.cts.Cancel();
        }
        _sensors.Clear();
    }

    private async Task MonitorSensorAsync(SensorConfig config,
        IRadiationSensor sensor, CancellationToken token)
    {
        while (!token.IsCancellationRequested)
        {
            try
            {
                double value = await sensor.ReadAsync(token);
                var reading = new SensorReading
                {
                    SensorId = sensor.Id,
                    Timestamp = DateTime.Now,
                    Value = value
                };

                // 1) викликаємо подію про нове значення (якщо є
                підписники)

                var readingHandler = ReadingReceived;
                if (readingHandler != null)

```

```

    {
        readingHandler(reading);
    }

    // 2) перевіряємо пороги
    if (value > config.MaxThreshold)
    {
        string msg = string.Format(
            "Сенсор '{0}' ({1}): value {2:F3}  $\mu$ Sv/h вийшов за
встановлений діапазон [{3:F3}; {4:F3}]",
            config.Name, config.Id, value, config.MinThreshold,
            config.MaxThreshold);

        var thresholdHandler = ThresholdExceeded;
        if (thresholdHandler != null)
        {
            thresholdHandler(reading, msg);
        }
    }

    // 3) чекаємо інтервал опитування для цього датчика
    await Task.Delay(config.PollingInterval, token);
}
catch (TaskCanceledException)
{
    // нормальне завершення
    break;
}
catch (Exception ex)
{

```

```

        Console.ForegroundColor = ConsoleColor.Yellow;
        Console.WriteLine("Error in sensor " + sensor.Id + "
monitoring: " + ex.Message);
        Console.ResetColor();
        await Task.Delay(TimeSpan.FromSeconds(2), token);
    }
}
}
}
}
}

```

B.9 SensorReading.cs

```

using System;

namespace RadiationMonitoringConsole
{
    public class SensorReading
    {
        public string SensorId { get; set; } = "";
        public DateTime Timestamp { get; set; }
        public double Value { get; set; } //  $\mu$ Sv/h
    }
}

```

B.10 SensorRepository.cs

```

using System;
using System.Collections.Generic;
using System.Text;

```

```

using MySql.Data.MySqlClient;
using RadiationMonitoringConsole;

public class SensorRepository
{
    private readonly string _connectionString;

    public SensorRepository(string connectionString)
    {
        _connectionString = connectionString;
    }

    public void InsertSensor(SensorConfig cfg)
    {
        using (var conn = new MySqlConnection(_connectionString))
        {
            conn.Open();

            string sql = @"INSERT INTO sensors
                            (Id, Name, Place, Model, MinThreshold, MaxThreshold,
                             PollingSec, Smh, Comment, Status, Start)
                            VALUES
                            (@Id, @Name, @Place, @Model, @MinThreshold,
                             @MaxThreshold,
                             @PollingSec, @Smh, @Comment, @Status, @Start)";

            using (var cmd = new MySqlCommand(sql, conn))
            {
                cmd.Parameters.AddWithValue("@Id", cfg.Id);
                cmd.Parameters.AddWithValue("@Name", cfg.Name);
            }
        }
    }
}

```

```

        cmd.Parameters.AddWithValue("@Place", cfg.Place);
        cmd.Parameters.AddWithValue("@Model", cfg.Model);

        cmd.Parameters.AddWithValue("@MinThreshold",
cfg.MinThreshold);
        cmd.Parameters.AddWithValue("@MaxThreshold",
cfg.MaxThreshold);

        int pollingSec = (int)cfg.PollingInterval.TotalSeconds;
        cmd.Parameters.AddWithValue("@PollingSec", pollingSec);

        cmd.Parameters.AddWithValue("@Smh", cfg.Smh);
        cmd.Parameters.AddWithValue("@Comment", cfg.Comment);
        cmd.Parameters.AddWithValue("@Status", cfg.Status);
        cmd.Parameters.AddWithValue("@Start", cfg.Start);

        cmd.ExecuteNonQuery();
    }
}

// □ Читання BCIX датчиків
public List<SensorConfig> GetAllSensors()
{
    var result = new List<SensorConfig>();

    using (var conn = new MySqlConnection(_connectionString))
    {
        conn.Open();
    }
}

```

```

string sql = @"SELECT Id, Name, Place, Model,
                    MinThreshold, MaxThreshold,
                    PollingSec, Smh, Comment, Status, Start
                    FROM sensors";

using (var cmd = new MySqlCommand(sql, conn))
using (var reader = cmd.ExecuteReader())
{
    while (reader.Read())
    {
        var cfg = new SensorConfig();

        cfg.Id = reader.GetString("Id");
        cfg.Name = reader.GetString("Name");
        cfg.Place = reader.GetString("Place");
        cfg.Model = reader.GetString("Model");

        cfg.MinThreshold = reader.GetDouble("MinThreshold");
        cfg.MaxThreshold = reader.GetDouble("MaxThreshold");

        int pollingSec = reader.GetInt32("PollingSec");
        cfg.PollingInterval = TimeSpan.FromSeconds(pollingSec);

        cfg.Smh = reader.GetInt32("Smh");

        int commentIndex = reader.GetOrdinal("Comment");
        cfg.Comment = reader.IsDBNull(commentIndex)
            ? ""
            : reader.GetString(commentIndex);
    }
}

```

```

        cfg.Status = reader.GetString("Status");
        cfg.Start = reader.GetDateTime("Start");

        result.Add(cfg);
    }
}

return result;
}

public void InsertReading(SensorReading reading)
{
    using (var conn = new MySqlConnection(_connectionString))
    {
        conn.Open();

        string sql = @"INSERT INTO sensor_readings
                        (SensorId, Timestamp, Value)
                        VALUES
                        (@SensorId, @Timestamp, @Value)";

        using (var cmd = new MySqlCommand(sql, conn))
        {
            cmd.Parameters.AddWithValue("@SensorId", reading.SensorId);
            cmd.Parameters.AddWithValue("@Timestamp",
reading.Timestamp);
            cmd.Parameters.AddWithValue("@Value", reading.Value);

            cmd.ExecuteNonQuery();

```

```

        }
    }
}

public void DeleteSensor(string sensorId)
{
    using (var conn = new MySqlConnection(_connectionString))
    {
        conn.Open();

        string sql = @"DELETE FROM sensors WHERE Id = @SensorId";

        using (var cmd = new MySqlCommand(sql, conn))
        {
            cmd.Parameters.AddWithValue("@SensorId", sensorId);
            cmd.ExecuteNonQuery();
        }
    }
}

public void DeactivateSensorReadings(string sensorId)
{
    using (var conn = new MySqlConnection(_connectionString))
    {
        conn.Open();

        string sql = @"UPDATE sensor_readings
                        SET Active = FALSE
                        WHERE SensorId = @SensorId";

        using (var cmd = new MySqlCommand(sql, conn))

```

```

    {
        cmd.Parameters.AddWithValue("@SensorId", sensorId);
        cmd.ExecuteNonQuery();
    }
}

```

```

public void UpdateSensor(SensorConfig cfg)
{
    using (var conn = new MySqlConnection(_connectionString))
    {
        conn.Open();

        string sql = @"UPDATE sensors
                        SET Name = @Name,
                            Place = @Place,
                            Model = @Model,
                            MinThreshold = @MinThreshold,
                            MaxThreshold = @MaxThreshold,
                            PollingSec = @PollingSec,
                            Smh = @Smh,
                            Comment = @Comment,
                            Status = @Status,
                            Start = @Start
                        WHERE Id = @Id";

        using (var cmd = new MySqlCommand(sql, conn))
        {
            cmd.Parameters.AddWithValue("@Id", cfg.Id);
            cmd.Parameters.AddWithValue("@Name", cfg.Name);

```

```

        cmd.Parameters.AddWithValue("@Place", cfg.Place);
        cmd.Parameters.AddWithValue("@Model", cfg.Model);
        cmd.Parameters.AddWithValue("@MinThreshold",
cfg.MinThreshold);
        cmd.Parameters.AddWithValue("@MaxThreshold",
cfg.MaxThreshold);

        int pollingSec = (int)cfg.PollingInterval.TotalSeconds;
        cmd.Parameters.AddWithValue("@PollingSec", pollingSec);
        cmd.Parameters.AddWithValue("@Smh", cfg.Smh);

        cmd.Parameters.AddWithValue("@Comment", cfg.Comment);
        cmd.Parameters.AddWithValue("@Status", cfg.Status);
        cmd.Parameters.AddWithValue("@Start", cfg.Start);

        cmd.ExecuteNonQuery();
    }
}
}

```

```

public List<SensorReading> GetLastReadings(string sensorId, int limit = 50)
{
    var result = new List<SensorReading>();

    using (var conn = new MySqlConnection(_connectionString))
    {
        conn.Open();

        string sql = @"SELECT SensorId, Timestamp, Value, Active
                        FROM sensor_readings

```

```
WHERE SensorId = @SensorId
AND Active = 1
ORDER BY Timestamp DESC
LIMIT @Limit";
```

```
using (var cmd = new MySqlCommand(sql, conn))
{
    cmd.Parameters.AddWithValue("@SensorId", sensorId);
    cmd.Parameters.AddWithValue("@Limit", limit);

    using (var reader = cmd.ExecuteReader())
    {
        while (reader.Read())
        {
            var reading = new SensorReading
            {
                SensorId = reader.GetString("SensorId"),
                Timestamp = reader.GetDateTime("Timestamp"),
                Value = reader.GetDouble("Value")
            };

            result.Add(reading);
        }
    }
}

result.Reverse();
return result;
}
```

```

public List<SensorReading> GetReadingsByPeriod(string sensorId, DateTime
from, DateTime to)
{
    var result = new List<SensorReading>();

    using (var conn = new MySqlConnection(_connectionString))
    {
        conn.Open();

        // Active = 1 (true) – показуємо тільки активні записи
        string sql = @"
SELECT SensorId, Timestamp, Value
FROM sensor_readings
WHERE SensorId = @SensorId
    AND Active = 1
    AND Timestamp BETWEEN @From AND @To
ORDER BY Timestamp ASC;";

        using (var cmd = new MySqlCommand(sql, conn))
        {
            cmd.Parameters.AddWithValue("@SensorId", sensorId);
            cmd.Parameters.AddWithValue("@From", from);
            cmd.Parameters.AddWithValue("@To", to);

            using (var reader = cmd.ExecuteReader())
            {
                while (reader.Read())
                {
                    result.Add(new SensorReading
                    {

```

```

        SensorId = reader.GetString("SensorId"),
        Timestamp = reader.GetDateTime("Timestamp"),
        Value = reader.GetDouble("Value")
    });
    }
}
}

return result;
}

public void InsertEventLog(string message, string sensorId = null)
{
    using (var conn = new MySqlConnection(_connectionString))
    {
        conn.Open();

        string sql = @"INSERT INTO event_log (SensorId, Message,
CreatedAt)
VALUES (@SensorId, @Message, @CreatedAt)";

        using (var cmd = new MySqlCommand(sql, conn))
        {
            cmd.Parameters.AddWithValue("@SensorId",
                string.IsNullOrEmpty(sensorId) ? (object)DBNull.Value :
sensorId);

            cmd.Parameters.AddWithValue("@Message", message);
            cmd.Parameters.AddWithValue("@CreatedAt", DateTime.Now);

```

```

        cmd.ExecuteNonQuery();
    }
}

```

```

public List<diploma.EventLogEntry> GetEventLogByPeriod(
    DateTime from,
    DateTime to,
    List<string> sensorIds,
    bool includeNullSensorEvents)
{
    var result = new List<diploma.EventLogEntry>();

    using (var conn = new MySqlConnection(_connectionString))
    {
        conn.Open();

        var sql = new StringBuilder();
        sql.Append(@"
SELECT Id, SensorId, Message, CreatedAt
FROM event_log
WHERE CreatedAt BETWEEN @From AND @To
");

        // Фільтр по датчиках
        if (sensorIds != null && sensorIds.Count > 0)
        {
            sql.Append(" AND SensorId IN (");
            for (int i = 0; i < sensorIds.Count; i++)

```

```

        {
            if (i > 0) sql.Append(", ");
            sql.Append($"@sid{i}");
        }
        sql.Append(")");
    }
    else
    {
        // sensorIds == null або порожній список -> всі датчики,
        // але NULL беремо/не беремо за прапорцем
        if (!includeNullSensorEvents)
            sql.Append(" AND SensorId IS NOT NULL");
    }

    // якщо sensorIds заданий (обрані деякі) – NULL ми НЕ показуємо
    (по твоїх правилах)
    // тому додатково нічого не треба

    sql.Append(" ORDER BY CreatedAt DESC;");

    using (var cmd = new MySqlCommand(sql.ToString(), conn))
    {
        cmd.Parameters.AddWithValue("@From", from);
        cmd.Parameters.AddWithValue("@To", to);

        if (sensorIds != null && sensorIds.Count > 0)
        {
            for (int i = 0; i < sensorIds.Count; i++)
                cmd.Parameters.AddWithValue($"@sid{i}", sensorIds[i]);
        }
    }

```

```

using (var reader = cmd.ExecuteReader())
{
    while (reader.Read())
    {
        var e = new diploma.EventLogEntry
        {
            Id = reader.GetInt32("Id"),
            SensorId =
reader.IsDBNull(reader.GetOrdinal("SensorId"))
                ? null
                : reader.GetString("SensorId"),
            Message = reader.GetString("Message"),
            Timestamp = reader.GetDateTime("CreatedAt")
        };

        // якщо includeNullSensorEvents=false, а в БД все ж
        прийшло NULL – на всяк випадок відфільтруємо
        if (!includeNullSensorEvents &&
string.IsNullOrEmpty(e.SensorId))
            continue;
        result.Add(e);}}}}return result;}}

```

ДОДАТОК В
Слайди презентації

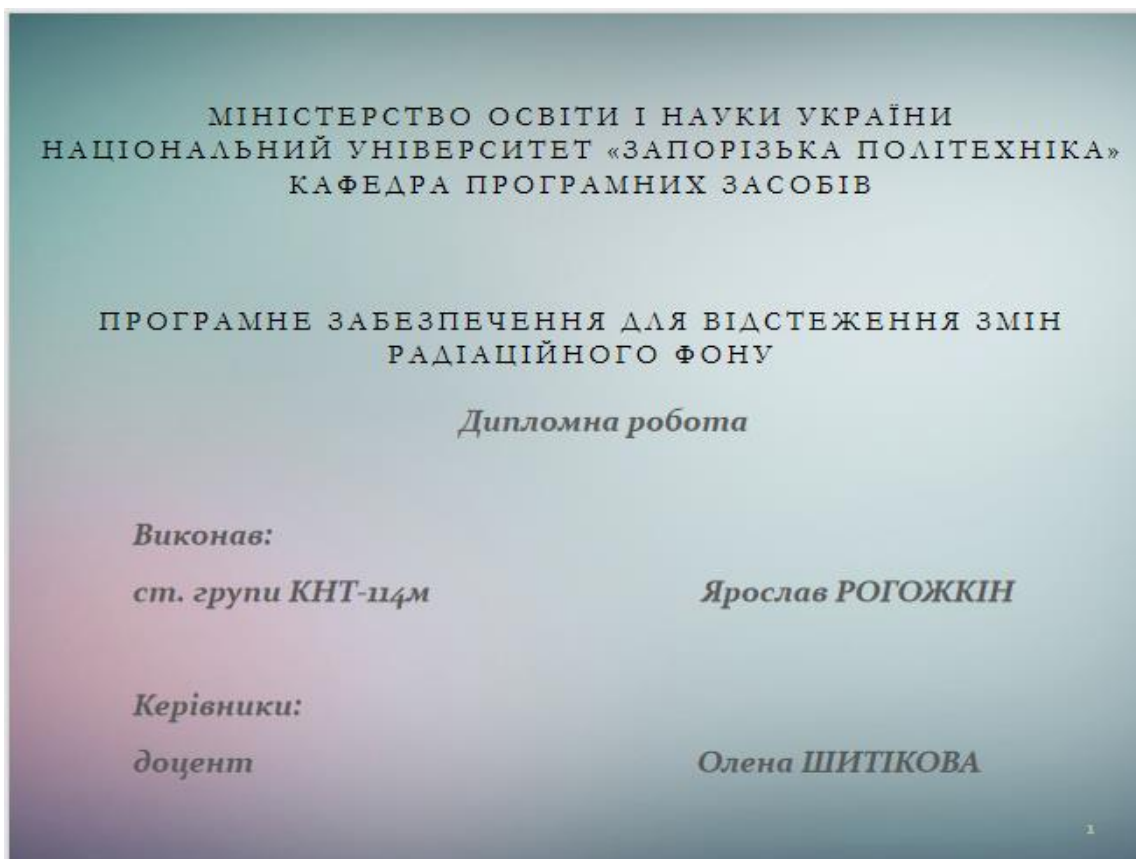


Рисунок В.1 – Слайд №1

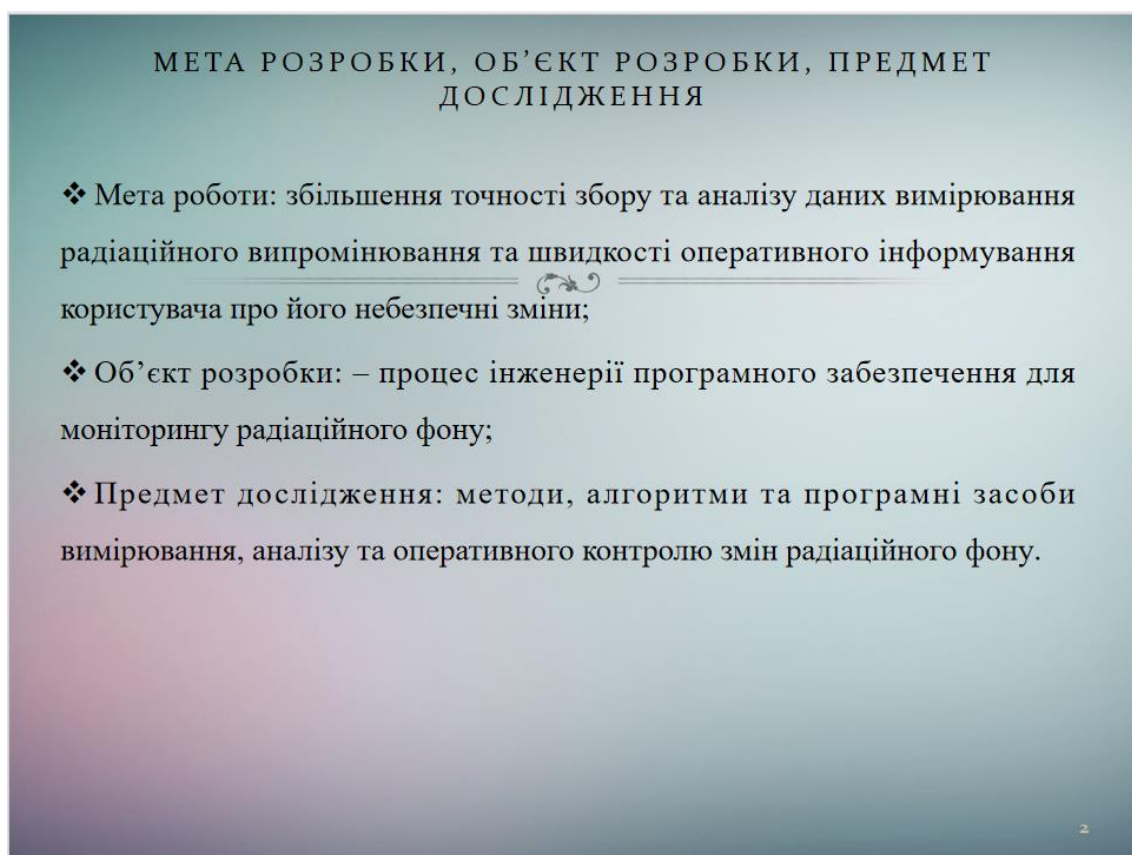


Рисунок В.2 – Слайд №2

ДЛЯ ДОСЯГНЕННЯ ПОСТАВЛЕНОЇ МЕТИ НЕОБХІДНО ВИКОНАТИ НАСТУПНІ ЗАВДАННЯ:

- провести аналіз сучасних тенденцій і технологій у сфері моніторингу радіації;
- провести проєктування архітектури застосунку з урахуванням функціональних та нефункціональних вимог;
- вивчити вимоги користувачів до функціоналу застосунку;
- розробити технічне завдання для створення застосунку для моніторингу змін радіації;
- обрати та обґрунтувати вибір технологій та інструментів для розробки застосунку;
- реалізувати структуру бази даних для зберігання інформації про датчики, їх заміри та системні повідомлення;
- розробити інтерфейс користувача, який забезпечить зручність та інтуїтивність у використанні;
- провести тестування та налагодження програмного продукту для забезпечення його коректної роботи;
- підготувати документацію та інструкції для користувачів і адміністраторів системи.

3

Рисунок В.3 – Слайд №3

ОГЛЯД ІСНУЮЧИХ МЕТОДІВ

Показники/ Застосунки	RadonEye RD200	Safecast SC	GammaGuard
Тип вимірювання	Локальне	Мережеве	Локальне
Моніторинг в реальному часі	Так	Так	Так
Встановлення порогових значень	Ні	Ні	Так
Автоматичне сповіщення про небезпеку	Ні	Ні	Так
Формування історії вимірювань	Так	Так	Так
Кросплатформність	Ні	Так	Ні
Підтримка зовнішніх датчиків	Ні	Так	Так
Аналітика та прогнозування	Ні	Так	Ні
Масштабованість мережі	Ні	Так	Ні

4

Рисунок В.4 – Слайд №4

ПОСТАНОВКА ЗАВДАННЯ

Програмна реалізація системи моніторингу радіаційного фону є актуальним завданням у зв'язку зі зростаючою потребою в оперативному контролі рівня іонізуючого випромінювання та своєчасному інформуванні про його небезпечні зміни. Радіаційний фон відіграє важливу роль у забезпеченні екологічної та техногенної безпеки, а його постійний контроль дозволяє зменшити ризики для здоров'я населення та навколишнього середовища.

Отже, актуальним завданням залишається програмна реалізація системи моніторингу радіаційного фону, яка забезпечує збір, аналіз і візуалізацію даних вимірювань у реальному часі з можливістю фіксації та перегляду історичних показників.

5

Рисунок В.5 – Слайд №5

ВИБІР МОВИ ПРОГРАМУВАННЯ

Критерії порівняння	R	C#	Python	Swift	Java
Ліцензія	GNU GPL	RAND	Python Software Foundation License	GNU GPL	MIT
Простота використання	–	+	+	+	–
Підтримка кросплатформеності	+	+	+	–	+
Продуктивність	+	+	+	+	+
Розвиток та підтримка	+	+	+	+	+
Візуальні інтерфейси (GUI)	+	+	–	+	+
Підтримка великих проєктів	–	+	–	–	+
Спільнота та ресурси	+	+	+	+	+
Інтеграція з іншими технологіями	+	+	+	–	+

6

Рисунок В.6– Слайд №6

ВИБІР СЕРЕДОВИЩА РОЗРОБКИ				
Критерії порівняння	Visual Studio	Visual Studio Code	JetBrain s Rider	Xamarin Studio
Ліцензія	МІТ	МІТ	–	МІТ
Кросплатформність	+	+	+	–
Автозавершення коду	+	+	+	–
Підтримка репозиторіїв	+	+	+	+
Простота встановлення	+	–	+	–
Перевірка та аналіз коду	+	+	+	+
Швидкий та безпечний рефакторинг коду	+	+	–	+
Наявність інтерактивної консолі	+	+	–	+
Наявність зручного користувацького інтерфейсу	+	+	+	+
Швидкий та зручний пошук	+	+	–	–
Підсвічування синтаксису	+	+	+	+
Підтримка звукових пакетів	+	+	+	–
Наявність функції статичного аналізу коду	+	+	+	+

Рисунок В.7 – Слайд №7



Рисунок В.8 – Слайд №8

СТРУКТУРА БАЗИ ДАНИХ

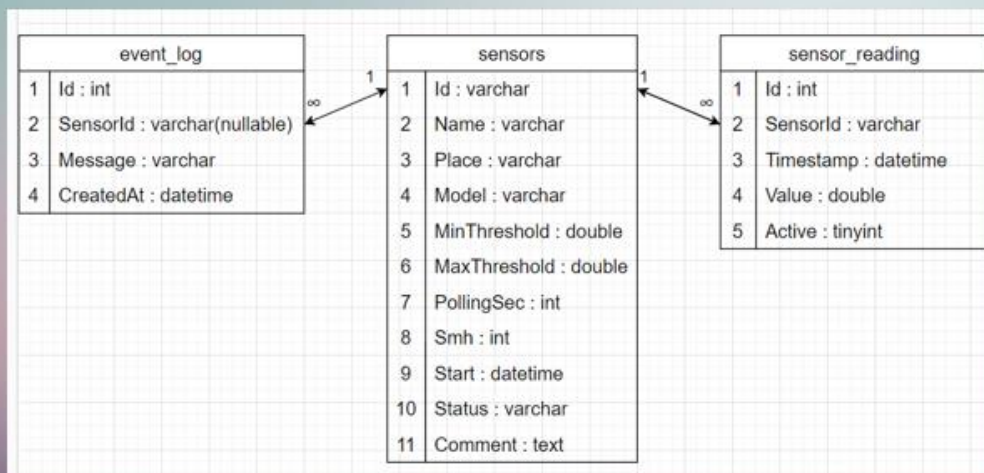


Рисунок В.9– Слайд №9

ІНТЕРФЕЙС ЗАСТОСУНКУ

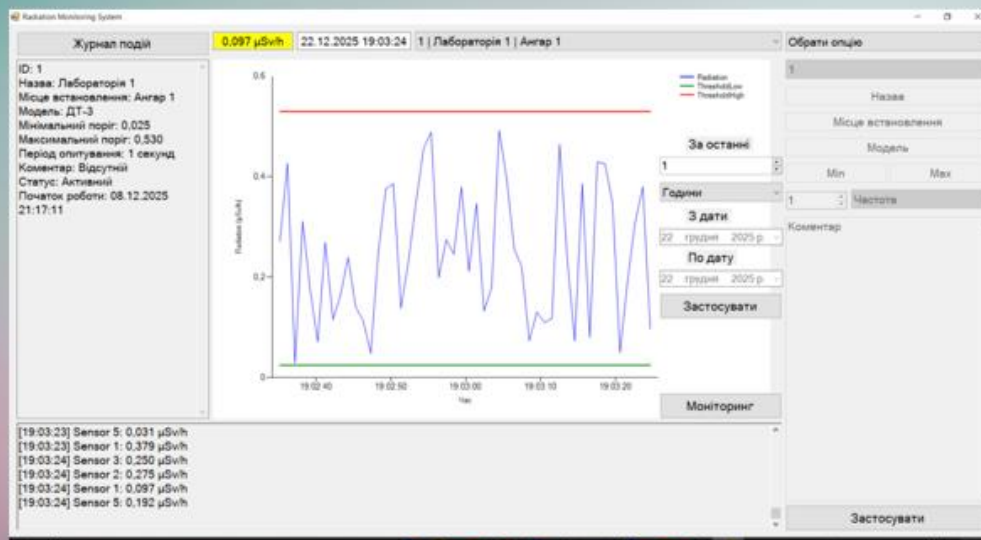


Рисунок В.10 – Слайд №10

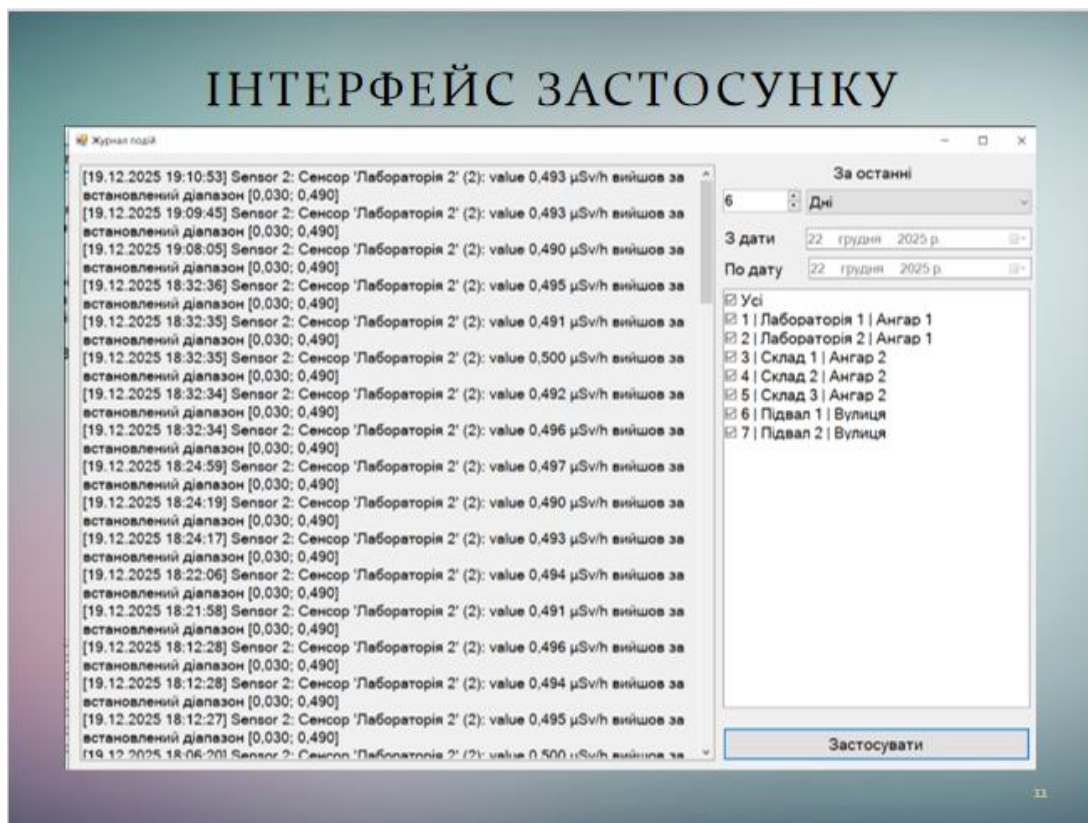


Рисунок В.11 – Слайд №11

ВИСНОВОК

- ❖ Проаналізовано предметну область моніторингу радіаційного фону;
- ❖ Визначено основні поняття та проблематику завдання;
- ❖ Обґрунтовано актуальність і доцільність розв'язання поставленої задачі;
- ❖ Досліджено існуючі методи вимірювання та контролю рівня іонізуючого випромінювання;
- ❖ Проаналізовано переваги та недоліки наявних методів і підходів;
- ❖ Проведено порівняльний аналіз підходів до організації радіаційного моніторингу;
- ❖ Розроблено програмний продукт для моніторингу радіаційного фону в реальному часі;
- ❖ Реалізовано Windows-застосунок для зчитування та аналізу показників радіації;
- ❖ Забезпечено графічне відображення результатів вимірювань;
- ❖ Реалізовано ведення журналу подій та формування попереджень при перевищенні порогових значень;
- ❖ Передбачено можливість керування датчиками та перегляду історії вимірювань;
- ❖ Реалізовано аналіз даних за вибрані часові періоди.

Рисунок В.12 – Слайд №12