Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматизації, комп'ютерно-інтегрованих технологій та систем
(повна назва)

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації, робототехніки та безпекової інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Розроблення додатку автоматизованої системи оповіщення населення при надзвичайних ситуаціях
(тема)

Виконав:
здобувач 4 року навчання,
групи АКТСІ-22-1

Владислав МАКОГОН
(власне ім'я, прізвище)

Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Системна інженерія
(повна назва освітньої програми)

Керівник доц. Артем БРОННІКОВ
(посада, власне ім'я, прізвище)

Допускається до захисту
Зав. кафедри КІТАРБІ _____ Ігор НЕВЛЮДОВ
(підпис) (власне ім'я, прізвище)

2026 р.

Я, Макогон Владислав Олегович, як здобувач вищої освіти ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Я не використовував штучний інтелект для підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на віповідне джерело.

"04" липня 2026 р.  Макогон В.О.

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет АКІТС
Кафедра КІТАРБІ
Рівень вищої освіти перший (бакалаврський)
Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
(код і повна назва)
Тип програми освітньо-професійна
Освітня програма Системна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри КІТАР _____
(підпис)
« ___»_____2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Макогону Владиславу Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення додатку автоматизованої системи оповіщення населення при надзвичайних ситуаціях
затверджена наказом по університету від «04» червня 2026 р. № 473 Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 04 липня 2026
3. Вихідні дані до роботи
3.1 Фреймворк FastAPI;
3.2 Програмна реалізація Telegram-бота;
3.3 Інтерфейс з HTML, JS, Bootstrap 5, Jinja2.
4. Перелік питань, що потрібно опрацювати в роботі
4.1 Аналіз сучасних автоматизованих систем оповіщення;
4.2 Вибір та обґрунтування стеку програмних технологій, розроблення серверної логіки автоматизованої системи оповіщення яка включає інтеграцію Telegram бот;
4.3 Оформлення пояснювальної записки кваліфікаційної роботи.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)
Графічний матеріал у вигляді презентації – 12 арк. ф. А4.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Приміт-ка
1	Аналіз технічного завдання	28.04.2026	виконано
2	Аналіз літератури за темою	30.04.2026	виконано
3	Аналіз сучасних систем оповіщення населення при надзвичайних ситуаціях	04.05.2026	виконано
4	Проектування архітектури автоматизованої системи оповіщення	25.05.2026	виконано
5	Розробка автоматизованої системи оповіщення	20.06.2026	виконано
6	Подання роботи на перевірку Інтернет-сервісом StrikePlagiarism	03.07.2026	виконано
7	Оформлення пояснювальної записки	04.07.2026	виконано
8	Подання роботи на рецензію	05.07.2026	виконано
9	Подання роботи на підпис зав. кафедри	10.07.2026	виконано
10	Подання кваліфікаційної роботи в ЕК	17.07.2026	виконано

Дата видачі завдання 27 квітня 2026 р.

Здобувач _____ Владислав МАКОГОН
(підпис) (власне ім'я, прізвище)

Керівник роботи _____ доц. Артем БРОННІКОВ.
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 117 с; 25 рис; 2 додатків, 27 джерел.

АВТОМАТИЗОВАНА СИСТЕМА ОПОВІЩЕННЯ, НАДЗВИЧАЙНА СИТУАЦІЯ, TELEGRAM-БОТ, WEB-ІНТЕРФЕЙС, ДИСПЕТЧЕРСЬКА ПАНЕЛЬ, PYTHON.

Об’єкт розробки – процес автоматизованого оперативного оповіщення населення при виникненні НС.

Предмет розробки – моделі, алгоритми та програмні засоби розроблення системи автоматизованої системи оповіщення.

Мета роботи – підвищення ефективності оперативного оповіщення населення при виникненні НС шляхом розробки та впровадження автоматизованої клієнт-серверної системи сповіщень.

У першому розділі було проведено аналіз сучасних автоматизованих систем оповіщення при надзвичайних ситуаціях.

У другому розділі було обрано та обґрунтовано стек програмних засобів для розробки. Спроектовано автоматизовану систему оповіщення, а також проведено математичні розрахунки.

У третьому розділі було розроблено та протестована автоматизована система оповіщення на базі фреймворку FastAPI яка включає інтеграцію Telegram бота та створено веб-панель диспетчера.

У результаті розроблена система може використовуватися як самостійне рішення для локальних територіальних громад або підприємств для забезпечення швидкого, цілеспрямованого інформування населення.

ABSTRACT

Explanatory note: 117 pages, 25 pictures, 2 appendices, 27 sources.

AUTOMATED NOTIFICATION SYSTEM, EMERGENCY SITUATION, TELEGRAM BOT, WEB INTERFACE, DISPATCHER PANEL, PYTHON.

Object of development – the process of automated operational notification of the population in the event of emergencies.

Subject of development – models, algorithms, and software tools for developing an automated notification system.

Purpose of the work – increasing the efficiency of operational population notification during emergencies through the development and implementation of an automated client-server notification system.

In the first chapter, an analysis of modern automated notification systems during emergency situations was conducted.

In the second chapter, the software stack for development was selected and justified. The automated notification system was designed, and mathematical calculations were performed.

In the third chapter, the automated notification system based on the FastAPI framework was developed and tested, which includes the integration of a Telegram bot and the creation of a dispatcher web panel.

As a result, the developed system can be used as a standalone solution for local territorial communities or enterprises to ensure rapid and targeted notification of the population.

ЗМІСТ

Перелік скорочень 9

Вступ 10

1 Аналіз сучасних систем оповіщення населення при надзвичайних ситуаціях 12

1.1. Загальні відомості про системи оповіщення 12

1.2. Аналіз технологій доставки повідомлень 14

1.3. Аналіз світових та вітчизняних систем оповіщення 15

1.3.1 ACO J-Alert 15

1.3.2. ShakeAlert 17

1.3.3. «Повітряна тривога» 18

1.3.4 Програмна платформа об’єктового оповіщення Ignition SCADA 20

2 Проектування архітектури автоматизованої системи оповіщення 22

2.1 Обґрунтування вибору стеку програмних технологій 22

2.1.1 Обґрунтування вибору мови програмування Python 22

2.1.2 Обґрунтування вибору фреймворку сервера FastAPI 23

2.1.3 Вибір СУБД SQLite 23

2.1.4 Вибір фреймворку для реалізації телеграм-боту 24

2.1.5 Технології для Front-end розробки 24

2.2 Проєктування архітектури серверної частини 24

2.3 Розрахунок максимального часу оповіщення 26

2.4 Побудова математичної моделі об'єкта керування 27

3 Розробка автоматизованої системи оповіщення 30

3.1 Розробка блок-схеми та алгоритму роботи серверної частини 30

3.2 Розробка системи автоматизованого оповіщення 33
3.3 Опис роботи серверної частини та інтерфейсу 34
3.3.1 Шар роутерів або Ендпоінти FastAPI 34
3.3.2 Сервісний шар або бізнес логіка 35
3.3.3 Шар репозиторій 36
3.3.4 Інтеграція Telegram-боту 37
3.3.5 Веб інтерфейс 40
3.3.6 Авторизація для доступу диспетчера 42
3.4 Охорона праці 46
Висновки 48
Перелік джерел посилання 50
ДОДАТОК А Код програми 54
ДОДАТОК Б Демонстраційний матеріал 116

ПЕРЕЛІК СКОРОЧЕНЬ

АЕС – атомна електростанція;

АСО – автоматизована система оповіщення;

БД – база даних;

ДБН – державні будівельні норми;

ДСН – державні санітарні норми;

ДСНС – Державна служба України з надзвичайних ситуацій;

НПАОП – нормативно-правовий акт з охорони праці;

НС – надзвичайна ситуація;

САПР – системи автоматизованого проектування;

СУБД – система управління базами даних;

API – Application Programming Interface (інтерфейс прикладного програмування);

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення);

CSS – Cascading Style Sheets (каскадні таблиці стилів);

EEW – Earthquake Early Warning (система раннього попередження про землетруси);

HTML – HyperText Markup Language (мова розмітки гіпертексту);

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту);

JS – JavaScript (мова програмування);

KISS – Keep It Simple, Stupid (принцип проєктування «роби простіше»);

ORM – Object-Relational Mapping (об'єктно-реляційне відображення);

RAM – Random Access Memory (оперативна пам'ять);

SCADA – Supervisory Control And Data Acquisition (система диспетчерського управління та збору даних);

SMS – Short Message Service (служба коротких повідомлень);

SQL – Structured Query Language (мова структурованих запитів).

ВСТУП

В епоху стрімкого розвитку інформаційних технології та зростання кількості виникнення надзвичайних ситуацій різного походження, від природних катаклізмів і техногенних аварій до загроз воєнного часу. Питання ефективного та швидкого інформування набуває критичного значення. Швидкість реакції на загрозу та своєчасність отримання інструкцій безпосередньо впливають на збереження людських життів та мінімізацію матеріальних збитків.

Традиційні системи оповіщення, такі як сирени чи радіотрансляції, хоч і досить ефективні для масового оповіщення мають суттєві обмеження через нездатність надавання деталізованих, персоналізованих інструкції. Вони повідомляють про сам факт НС, проте не здатні надавати оперативну інформацію про поточну інформацію загрози.

Водночас високий рівень розвитку мобільних технологій створює всі умови для впровадження сучасних програмних рішень у сфері цивільного захисту. Використання популярних месенджерів та спеціалізованих додатків відкриває нові можливості для автоматизації процесу розсилки. Розроблення автоматизованої системи оповіщення з використанням сучасних клієнт-серверних архітектур та інтеграцією сторонніх API дозволяє мінімізувати вплив людського фактору, забезпечити миттєву доставку повідомлень, структурувати інформацію за рівнем загрози та гарантувати громадянам доступ до актуальних даних у режимі реального часу.

Таким чином, мета роботи – підвищення ефективності оперативного оповіщення населення при виникненні НС шляхом розробки та впровадження автоматизованої клієнт-серверної системи сповіщень.

Об’єкт розробки – процес автоматизованого оперативного оповіщення населення при виникненні НС.

Предмет розробки – моделі, алгоритми та програмні засоби розроблення системи автоматизованої системи оповіщення.

Для досягнення поставленої мети необхідно вирішити такий комплекс завдань:

– аналіз існуючих систем автоматизованого оповіщення населення при НС;

– вибір засобів розробки програмного забезпечення для реалізації клієнтської та серверної частин системи;

– спроектувати архітектуру системи, серверну логіку;

– розробка алгоритму обробки подій та серверну логіку, реалізувати механізм створення, редагування та закриття тривог;

– створити користувацький інтерфейс для зручного та швидкого управління системою в умовах стресових навантажень;

– інтегрувати механізм гарантованої розсилки сповіщень через API зовнішнього месенджера.

Отримані результати можна віднести до Цілі сталого розвитку 11 «Сталі міста та громади» в аспекті підвищення безпеки, комплексної життєстійкості населених пунктів і захисту населення від наслідків надзвичайних ситуацій, а також Ціллю 9 «Індустріалізація, інновації та інфраструктура» шляхом впровадження новітніх інформаційних технологій та розбудови надійної цифрової інфраструктури у сфері цивільного захисту.

Пояснювальну записку кваліфікаційної роботи оформлено згідно з ДСТУ 3008:2015 [1] та з рекомендаціями з підготовки і оформлення кваліфікаційної роботи здобувачами першого (бакалаврського) рівня вищої освіти [2, 3].

1 АНАЛІЗ СУЧАСНИХ СИСТЕМ ОПОВІЩЕННЯ НАСЕЛЕННЯ ПРИ НАДЗВИЧАЙНИХ СИТУАЦІЯХ

1.1. Загальні відомості про системи оповіщення

Системи оповіщення населення про надзвичайні ситуації (НС) являє собою один з найважливіший елементів комплексу заходів цивільного захисту. Головною метою таких систем є максимально швидке та достовірне доведення сигналів тривоги до населення.

З технічної точки зору автоматизовані системи оповіщення (АСО) це програмно-технічний комплекс який поєднує апаратні засоби, програмне забезпечення, канали зв’язку та мережі передачі даних, метою якого становить автоматизація процесу збору, обробки та розсилки повідомлень про НС.

Відповідно до масштабу охоплення системи оповіщення класифікують за такими рівнями:

– загальнодержавний рівень: охоплення території всієї країни;

– територіальні рівень: охоплення в межах окремого великого регіону;

– місцеві рівень: охоплення в межах міста або района;

– локальний рівень: розміщується у районах потенційно небезпечних об’єктів (АЕС, хімічні заводи);

– об’єктовий рівень: оповіщення в межах підприємства або будівлі [4].

Сучасні АСО є складною багаторівневою структурою. Незалежно від масштабу охоплення, архітектура АСО концептуально складається з 4 підсистем. Підсистема моніторингу та збору даних відповідає за сбір даних з фізичних датчиків (сейсмометрів, газоаналізатори) або програмні засоби для введення інформації оператором. Зібрані дані переходять до підсистеми управління та обробки яка виступає логічним ядром системи. На цьому рівні здійснюється аналіз, верифікація даних, розрахунок зони ураження та формується відповідне повідомлення про загрозу, за наявності бази даних

також формується список користувачів яким буде відправлено повідомлення. Це повідомлення надходить до підсистеми зв’язку та маршрутизації яка відповідає за передачу інформаційних пакетів використовуючи для цього оптоволоконні лінії, супутникові канали, радіочастоти тощо. У кінці сигнал досягає підсистеми виконавчих пристроїв, що забезпечує безпосередньо передачу інформації населенню. До цієї підсистеми входять сирени, радіомовлення, системи перехоплення теле- і радіоефіру та смартфон кінцевого користувача (рис. 1.1).

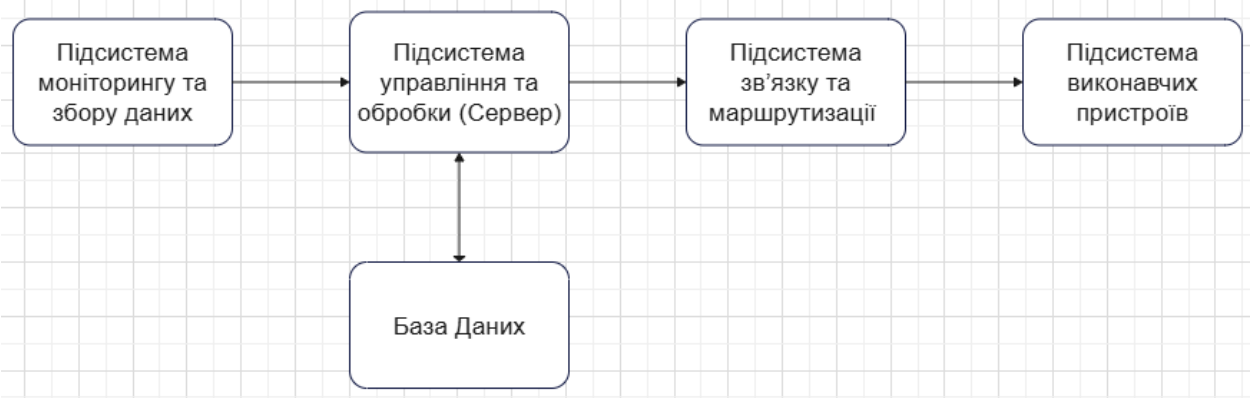


Рисунок 1.1 – Базова структура сучасних АСО

Історично базовим інструментом для оповіщення населення при НС став сигнал який транслюється через електромеханічні сирени та повідомляє про тривогу. Після цього сигналу слідує інформація яка надається через мережі радіомовлення, телебачення або радіостанції. Однак у такого підходу є суттєві недоліки:

– залежність від електропостачання. У разі пошкодження енергетичної інфраструктури класичні сирени та станції радіомовлення втрачають працездатність;

– низька інформативність. Сирени здатні привернути увагу населення але в більшості ситуацій не пояснюють суті загрози та не надають подальших інструкцій щодо дій;

– відсутність адресності. Традиційні засоби такі як сирену не дозволяють локалізувати зону сповіщення з високою точністю, що приводить до необґрунтованої паніки серед населення на безпечній території або навіть ігнорування тривоги у майбутньому.

Вищезазначені обмеження класичних систем оповіщення формують об’єктивну необхідність переходу на більш сучасні цифрові рішення. Використання мобільних додатків дозволить вирішити більшість недоліків класичних систем забезпечуючи персоналізоване, автономне та інформативне оповіщення громадян через їх смартфони [5].

1.2. Аналіз технологій доставки повідомлень

Ефективність будь-якої системи сповіщення буде напряму залежить від технології доставки повідомлення до кінцевого користувача. В умовах надзвичайної ситуації канал зв’язку повинен відповідати жорстким вимогам щодо затримки, відмовостійкості та здатності витримувати пікові навантаження без деградації якості. На сьогоднішній день виділяють чотири основні технології доставки масових повідомлень.

SMS (Short Message Service, “Служба коротких повідомлень”) історично перший і найпоширеніший метод оповіщення. Його головною перевагою є те що SMS підтримується всіма стандартами мобільного зв’язку і не вимагає наявності підключення до мережі інтернет. Проте у цієї технологій є великий недолік: архітектура SMS не призначена для одночасної масової розсилки повідомлень. Під час надзвичайних ситуацій стільникові мережі зазнають екстремальних перевантажень що приводить до критичних затримок [6].

Cell Broadcast (Стільникове мовлення) це стандарт обміну даними у стільникових мережах, спеціально розроблений для одночасної розсилки повідомлень багатьом абонентам у радіусі дії конкретних базових станцій. На відміну від SMS, Cell Broadcast не створює навантаження на мережу, оскільки повідомлення транслюється в односторонньому порядку. Незважаючи на

найвищу ефективність для загальнонаціонального оповіщення, ця технологія вимагає інтеграції безпосередньо в інфраструктуру національних операторів зв’язку. Її неможливо використати для локальних або об’єктових АСО [7].

Технологія Push-повідомлень сучасний метод доставки повідомлень через глобальну мережу інтернет із використанням хмарних брокерів повідомлень. Ця технологія здатна передавати мільйони повідомлень за мілісекунди та може передавати структуровані дані що містять координати, тип загрози та інструкцію. Підтримує пріоритетні канали, які здатні обходити беззвучний режим пристрою. Недоліком є повна залежність від інтернету та необхідність встановлення окремого додатка [8].

Боти популярних месенджерів (Telegram, Viber) як транспорт для сповіщення. Завдяки високому рівню проникнення у населенні, кросплатформеність, висока швидкість обробка асинхронних запитів та можливість реалізації складних сценаріїв зворотного зв’язку через інтерфейс чат-бота [9].

1.3. Аналіз світових та вітчизняних систем оповіщення

Для створення ефективної АСО критично важливим є вивчення вже існуючих та успішно впроваджених рішень як на міжнародному, так і на національному рівнях таких як американської ShakeAlert та японської J-Alert.

1.3.1 АСО J-Alert

J-Alert (повна назва Zenkoku Shunji Keihō Shisutemu, 'Національна система раннього оповіщення') це японська система раннього попередження що використовується у Японії. Запуск J-Alert відбувся у лютому 2007 року. Система призначена для швидкого інформування населення про загрози та надзвичайні ситуації, як землетруси, екстремальні погодні умови та інші небезпеки. Її було розроблено зі сподіванням, що завчасні попередження

пришвидшать евакуацію та допоможуть у координації дій реагування на надзвичайні ситуації.

J-Alert базується на супутникових системах, яка дозволяє владі швидко передавати попередження місцевим ЗМІ та безпосередньо громадянам через загальнонаціональну систему гучномовців, телебачення, радіо, електронну пошту та стільникове мовлення. За словами японських посадовців, інформування місцевої влади займає близько однієї секунди, а передача повідомлення громадянам — від чотирьох до двадцяти секунд. До кінця березня 2019 року було встановлено вдосконалену версію приймачів J-Alert. Нові моделі можуть автоматично обробляти інформацію протягом двох секунд, порівняно зі старішими моделями, яким на це може знадобитися до двадцяти секунд.

J-Alert здійснює мовлення як через наземні системи, так і через супутник зв'язку Superbird-B3 [10].

Діаграма роботи АСО J-Alert зображено на рисунку 1.2.

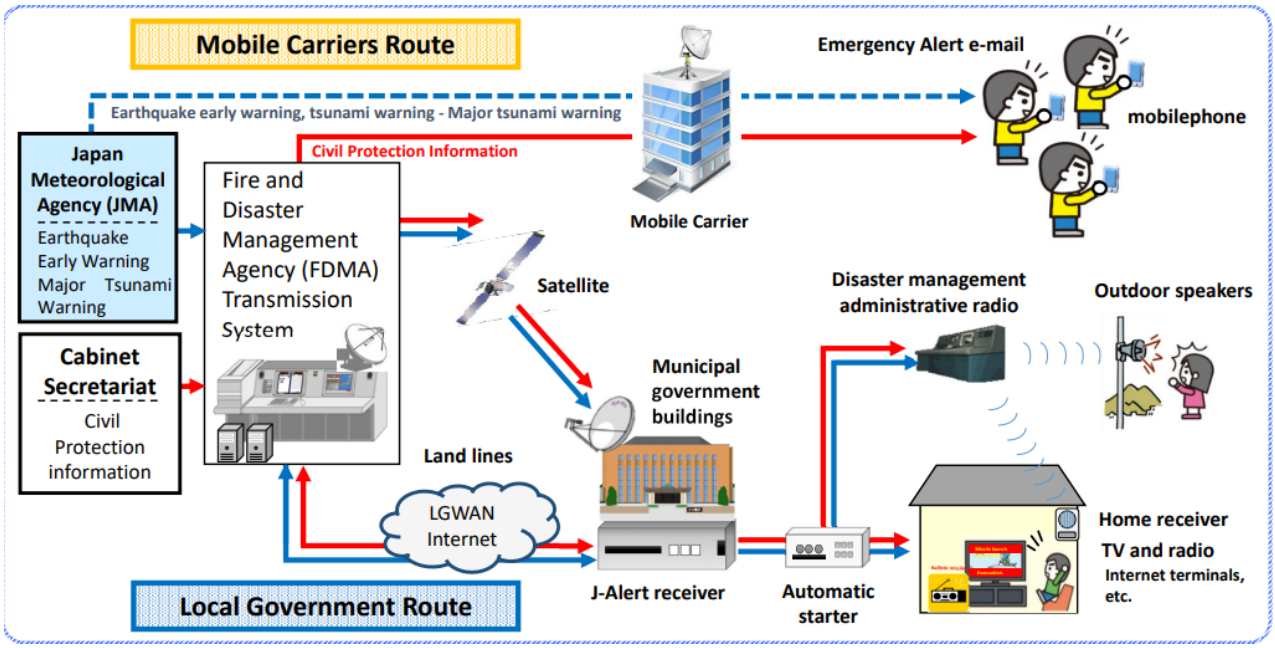


Рисунок 1.2 – Діаграма роботи J-Alert [11]

J-Alert автоматично передає інформацію про землетруси, цунамі, вулканічні виверження, екстремальні погодні умови, балістичні ракети,

повітряний напад, атаку партизанів або спецпідрозділів, терористичні атаки, тощо [12].

1.3.2. ShakeAlert

Американський ShakeAlert це система раннього попередження про землетруси (Earthquake Early Warning, EEW), розроблена геологічною службою США. На стан 2021 року система розвернута у регіоні західного узбережжя США (штати Каліфорнія, Орегон, Вашингтон) передбачається що система буде розширена на інші сейсмічно активні зони США у майбутньому.

Розробка системи почалася у 2006 році і у кінці 2018 року система була признана «достатньо функціональною та протестованою» для того щоб розпочати видавати повідомлення для штатів західного узбережжя. Вона була запущена в Каліфорнії 17 жовтня 2019 року, в Орегоні 11 березня 2021 року, а у Вашингтоні 4 травня 2021 року.

ShakeAlert не намагається передбачити землетруси замість цього вона як найшвидше знаходить сейсмічну активність та б’є тривогу поки ще не почали відчуватися сильні струси. Це є можливим завдяки безлічі високочутливих сейсмометрів, розміщених уздовж розломів, які уловлюють більш швидкі але слабші хвилі землетрусу, P-хвилі. Аналізуючи ці хвилі алгоритм системи за долі секунди визначає епіцентр та приблизну магнітуду землетрусу, якщо очікувана магнітуда перевищує поріг, система автоматично генерує сигнал тривоги до того як більш руйнівні S-хвилі дійдуть до населених пунктів (рис. 1.3) [13].

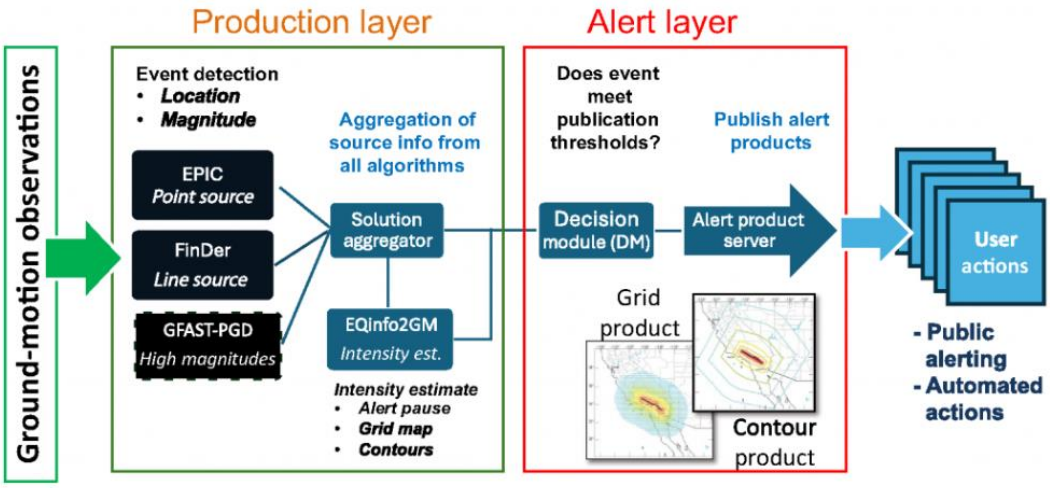


Рисунок 1.3 – Схема роботи системи ShakeAlert [13]

Система ShakeAlert сама по собі не є додатком, це бекенд-інфраструктура яка відповідає за серверну частину. Для доставки повідомлень вона використовує офіційних партнерів та їх додатки такі як MyShake від MyShake™ та Android Earthquake Alerts від Google [14].

1.3.3. «Повітряна тривога»

Застосунок «Повітряна тривога» це один з найкращих та успішних прикладів реалізації системи оповіщення населення при НС. Застосунок був розроблений у стислий час на початку повномасштабного вторгнення зусиллями компанії Ajax Systems та IT-компанії Stfalcon за підтримки Міністерства цифрової трансформації України. Головною причиною створення додатку стала неефективність та обмежене покриття традиційних систем оповіщення у вигляді електромеханічних сирен.

За принципом роботи застосунок, як зазначають розробники, не покладається на автоматизовані алгоритми, а безпосередньо інтегрований у державну систему цивільного захисту. Генерування тривоги ініціюється черговим центра управління, або ДСНС. Після натискання кнопки сигнал надходить на серверну частину, сервер обробляє запит та здійснює масову відправку push-сповіщень на пристрої користувачів відповідного регіону.

Завдяки механізму пріоритетного сповіщення додаток може ігнорувати беззвучний режим та режим «не турбувати» генеруючи сигнал максимальної гучності. Завдяки хмарним технологіям сервера здатні за лічені секунди розіслати мільйони повідомлень без затримок.

Інтерфейс додатку був розроблений з упором на мінімалізм та зручність використання під час НС, позбавлений декоративних функцій з максимальною ефективністю. На головній сторінці, на яку користувач потрапляє одразу після відкриття додатку, можна побачити статус кожного вибраного регіону та інструкцію щодо дій при цій загрозі (рис. 1.4).

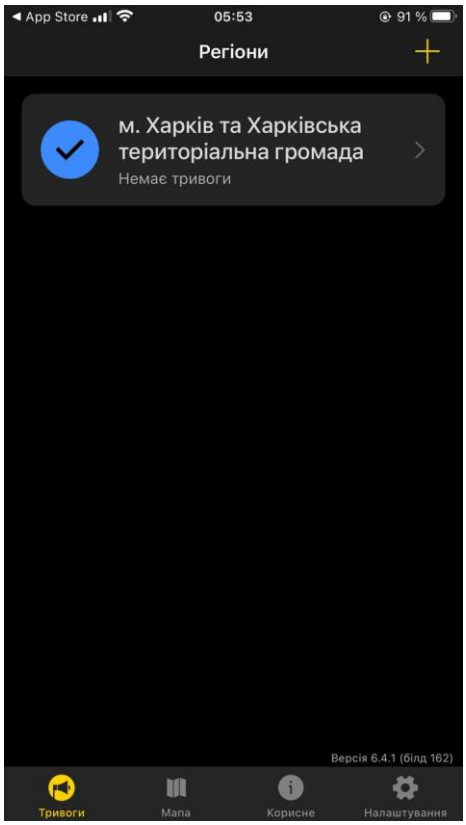


Рисунок 1.4 – Інтерфейс додатку «Повітряна тривога»

На самому початку додаток мав змогу передавати тільки сигнал про повітряну тривогу, але з його розвитком було впроваджено більше сигналів. Наразі система здатна оповіщати про: повітряну тривогу, артилерійський обстріл, вуличні бої, хімічну та радіаційну небезпеку, кожен з сигналів

супроводжується унікальним звуковим сигналом та текстової інструкції відповідно до загрози.

В останніх версіях системи було вдосконалено алгоритм обробки даних, впровадивши багаторівневу градацію небезпеки. Система здатна не просто сповістити про загальну небезпеку для області, але й генерувати повідомлення про «підвищену небезпеку» для конкретних територій чи міст куди безпосередньо рухається об’єкт загрози. Це стало можливим завдяки поглибленої інтеграції архітектури застосунку з даними моніторингу повітряного простору [15-16].

1.3.4 Програмна платформа об’єктового оповіщення Ignition SCADA

Провідною світовою платформою для створення локальних та об'єктових систем оповіщення на промислових об'єктах є Ignition SCADA від американської компанії Inductive Automation. Ключовою особливістю системи є глибока інтеграція інструментів інформування персоналу безпосередньо в середовище керування технологічними процесами. Це забезпечує миттєву реакцію як на зовнішні загрози, так і на внутрішні аварії — наприклад, витік небезпечних газів чи критичні стрибки тиску.

Важливою технологічною перевагою платформи є можливість візуального проєктування сценаріїв оповіщення. Завдяки зручним графічним інструментам диспетчери можуть налаштовувати складні логічні ланцюжки реагування без написання програмного коду (рис. 1.5).

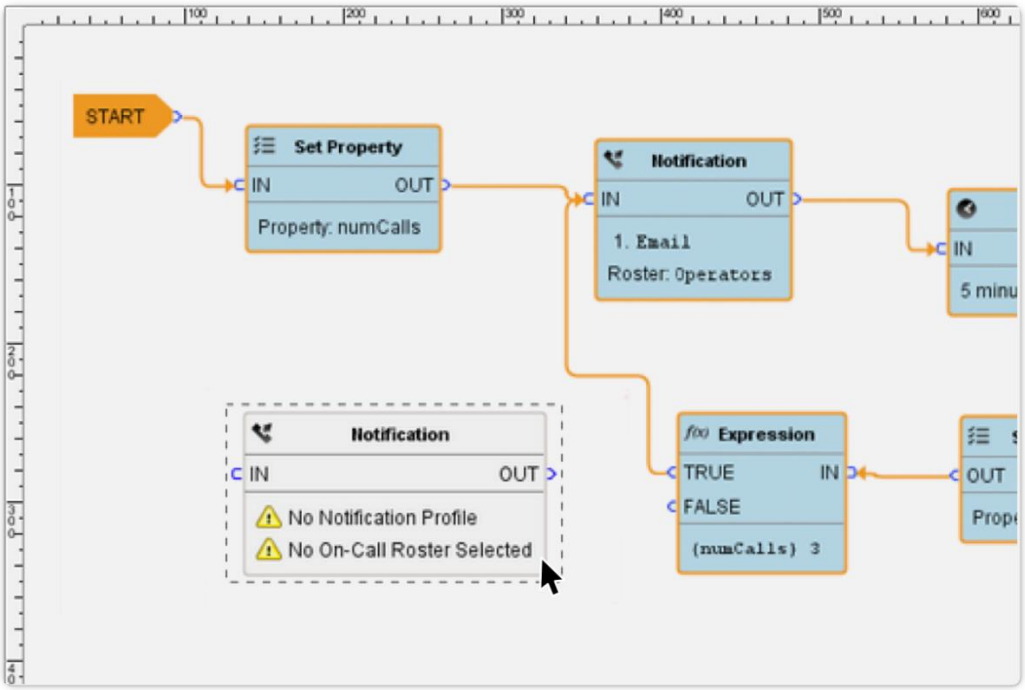


Рисунок 1.5 – Вигляд графічного інтерфейсу Ignition

Ці алгоритми дозволяють консолідувати масив дрібних тривог у єдине комплексне повідомлення для уникнення інформаційного перевантаження персоналу, а також налаштовувати тимчасові затримки, які дають операторам час на локалізацію проблеми без запуску масової розсилки. У випадку, якщо первинний адресат не відреагував на небезпеку протягом заданого часу, система ініціює процес ескалації, автоматично передаючи сигнал на вищий рівень керівництва.

Ключовим інструментом маршрутизації сигналів тривоги в платформі є спеціалізований модуль Alarm Notification Module. Його архітектура забезпечує гнучке налаштування логіки сповіщень відповідно до особливостей об'єкта. Система підтримує багатоканальну комунікацію, що дозволяє паралельно надсилати повідомлення через SMS, електронну пошту та автоматичні голосові виклики. Такий комплексний підхід гарантує успішну доставку критично важливої інформації персоналу навіть у виробничих зонах з обмеженим інтернет-покриттям [17].

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ АВТОМАТИЗОВАНОЇ СИСТЕМИ ОПОВІЩЕННЯ

2.1 Обґрунтування вибору стеку програмних технологій

Вибір стеку програмних технологій якими буде створена система є першим і найважливішим етапом проектування. У цьому пункті обґрунтовується вибір технологій для реалізації серверної частини, інтегрування телеграм-боту та візуалізації у вигляді веб сторінки.

Вибір стеку програмних технологій відбувався за принципом KISS (Keep it Simple, Stupid) що у перекладі «Роби простіше, дурню». Це принцип проектування що стверджує, що більша частина систем працюють найкраще якщо вони залишаються простими, а не ускладненими.

2.1.1 Обґрунтування вибору мови програмування Python

Як мова програмування для реалізації серверної частини було обрано Python, це обумовлюється універсальністю, високою швидкістю розробки та великою екосистемою стандартних і сторонніх бібліотек.

Синтаксис мови є лаконічним та зрозумілим, що дозволяє збільшити концентрацію на реалізації бізнес-логіки серверу та зробити код в цілому більш зрозумілим.

Сучасні стандарти Python широко підтримують асинхронність завдяки вбудованій бібліотеці asyncio, що є критичним для систем де сервер повинен підтримувати велику кількість з’єднань та виконувати операції вводу виводу без блокування основного потоку.

У мові Python наявна велика кількість бібліотек для роботи з базами даних, веб-протоколами та API месенджерів, це дозволяє швидко і без проблем інтегрувати компоненти системи в єдиний комплекс з мінімальними витратами ресурсів [18-20].

2.1.2 Обґрунтування вибору фреймворку сервера FastAPI

Для реалізації серверної частини веб панелі диспетчера було обрано сучасний та актуальний фреймворк FastAPI. На даний момент цей фреймворк вважається стандартом індустрії для подібних задач, він проектувався для асинхронних задач що ідеально підходить для системи моніторингу.

FastAPI збудовано поверх потужної бібліотеки валідації Pydantic, це дозволяє знизити об’єм коду не витрачаючи його на перевірки та валідації даних які надходять від фронтенду, що в свою чергу гарантує чистоту та безпеку бази даних.

Фреймворк має високу швидкодію завдяки мікрофреймворку Starlette. На даний момент FastAPI являє собою один з найшвидших веб-фреймворків на Python швидкість якого може зрівнятися з мовами програмування NodeJS та Go.

FastAPI автоматично генерує інтерактивну веб-сторінку з документацією за адресом «http://127.0.0.1:8000/docs» На цій сторінці відображаються усі ендпоінти які є у системі, також на цій сторінці їх можна протестувати та побачити відповідь від серверу [21-22].

2.1.3 Вибір СУБД SQLite

Для бази даних була обрана легка система управління базами даних SQLite. Більш складні та потужні СУБД як MySQL та PostgreSQL потребують складних налаштувань, відкриття портів, створювати окрему базу даних.

Для системи диспетчеризації не треба велика СУБД яка буде витримувати тисячі одночасних записів. Тривога буде додаватися не часто і тільки диспетчером що при роботі з великими СУБД приведе к тому що їх потужності просто будуть простоювати.

Великі СУБД тільки для роботи у фоновому режиму будуть «вижирати» велику кількість пам’яті RAM, коли SQLite поки до неї нема запитів взагалі не витрачає ресурси пам’яті.

Тому SQLite ідеальний вибір СУБД який гарно витримує одночасний запит на читання великої кількості запитів, а вибір MySQL або PostgreSQL буде надлишковою тратою ресурсів [23].

2.1.4 Вибір фреймворку для реалізації телеграм-боту

Для реалізації телеграм-боту було вибрано фреймворк aiogram, одним з найважливіших пунктів при виборі стала підтримка асинхронності для одночасної роботи з сервером на FastAPI.

За структурою та синтаксисом aiogram схожий на FastAPI що дозволяє легко розроблювати одночасно серверну частину і телеграм бот без навантаження та переходу від одного синтаксису до іншого [24].

2.1.5 Технології для Front-end розробки

Метою Front-End розробки було обрано веб інтерфейс диспетчерського меню, для його створення було обрано зв’язку Jinja2, HTML, Bootstrap5 та JS.

Jinja2 це шаблонізатор для Python який використовується для динамічного відображення контенту на веб-сторінках HTML. Він дозволяє відокремити данні від представлення, що дозволяє прямо у .html файлі підтримувати змінні, цикли та умови [25].

Bootstrap5 це фреймворк який надає інструменти для створення сайтів та веб додатків, з нього буде взято CSS шаблони для створення адаптивного сайту та пришвидшенню написання інтерфейсу через не потрібність налаштування власного CSS [26].

Завдяки JavaScript буде реалізовано дії диспетчера на сайті та взаємодія інтерфейсу з сервером FastAPI [27].

2.2 Проєктування архітектури серверної частини

Для реалізації серверної логіки було обрано використовувати багатошарову архітектуру, для розбиття коду на 3 базових шара:

– шар роутерів (API/Бот): Виступає кур’єром який тільки приймає JSON від браузера або повідомлення з телеграму та передає відповідь;

– сервісний шар (Бізнес-логіка): Цей шар виступає мозком системи у якому перевіряються умови, та виконуються перетворення;

– шар репозиторій (Робота з БД): Цей шар виконує запити та записи до бази даних.

Шар роутерів ділиться на дві частини, а саме: ендпоінти FastAPI та хендлери повідомлень телеграм-боту.

Також існує окрема частина яку не можна записати не в один з цих шарів а саме файл sender.py який виконує роль відправника повідомлень до телеграм боту щоб уникнути циклічного імпортування.

Багатошаровість забезпечує незалежність бізнес-логіки від інтерфейсу або бази даних та легкість тестування.

Слід зазначити існування відокремлених допоміжних файлів:

– database: Ініціалізація ядра SQLAlchemy та налаштування підключення до бази даних SQLite;

– dependency: Забезпечує безпечне відкриття та закриття сесії бази даних для кожного HTTP-запиту за допомогою механізмів впровадження залежності;

– models: Описує структуру таблиць бази даних у вигляді Python-класів;

– schemas: Відповідає за строгу валідацію вхідних даних та вихідних відповідей API за допомогою Pydantic-моделей;

– exceptions: Зберігає шаблони виключень.

На рисунку 2.1 зображено функціональну схему серверної логіки серверу.

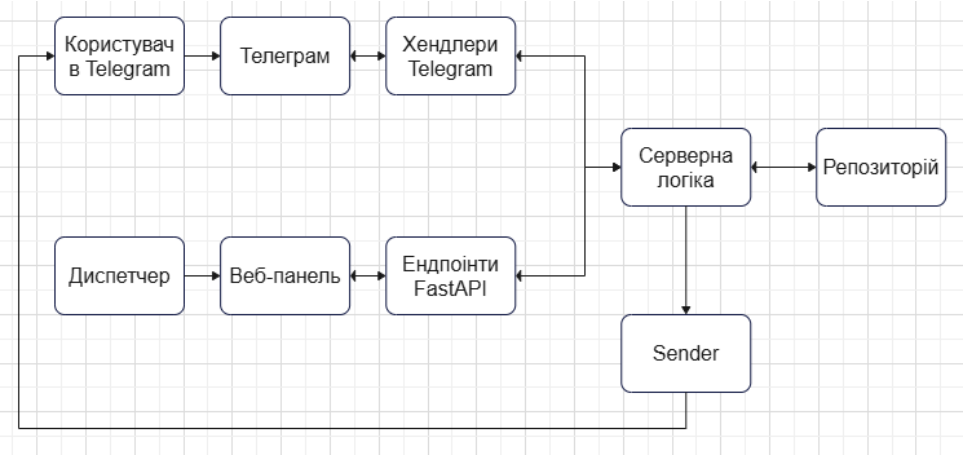


Рисунок 2.1 – Функціональна схема серверу

2.3 Розрахунок максимального часу оповіщення

При НС час є критичним. Telegram API має жорсткі обмеження на розсилку – 30 повідомлень на секунду. Необхідно розрахувати теоретичний час повного циклу оповіщення.

Загальний час повного циклу оповіщення розраховується по формулі:

$$T_{total} = t_{pack} + t_{db} + t_{send} + t_{net} \quad (2.1)$$

де t_{pack} – час формування пакету сервером (≈0.05 секунд);

t_{db} – час отримання chat_id з бази даних;

t_{send} – час відправки повідомлення у Telegram API;

t_{net} – затримка мережі (≈0.07*n секунд).

t_{db} можна розрахувати за формулою:

$$t_{db} = t_{conn} + (n \cdot t_{row}) \quad (2.2)$$

де t_{conn} – час підключення до БД (≈0.02);

n – кількість користувачів;

t_{row} – час читання одного запису у БД(≈0.001).

t_{send} можна розрахувати за формулою:

$$t_{send} = \frac{n}{V_{api}} \quad (2.3)$$

де n – кількість отримувачів;

V_{api} – обмеження Telegram API (30 на секунду).

Припустимо що на виробництві працює 100 працівників, тоді:

$$t_{db} = 0.02 + (100 \cdot 0.001) = 0.12 \text{ сек.}$$

$$t_{send} = \frac{100}{30} \approx 3.33 \text{ сек.}$$

$$T_{total} = 0.05 + 0.12 + 3.33 + 7 \approx 10.45 \text{ сек.}$$

Теоретичний розрахунок показав, що масова розсилка для 1000 користувачів займе приблизно мінімум 35 секунд. Це доказує необхідність використання асинхронних систем щоб система не «зависала» на великий час при відправці повідомлень.

2.4 Побудова математичної моделі об'єкта керування

Об'єктом керування (ОК) є асинхронний стек черги повідомлень вебсервера FastAPI, який взаємодіє із віддаленим сервером Telegram API через мережу Internet. Вхідним впливом $u(t)$ є інтенсивність генерації HTTP-запитів сервером (повідомлень/с), а вихідною величиною $y(t)$ – фактична

інтенсивність успішно оброблених та доставлених повідомлень Telegram API (повідомлень/с).

Через час на підготовку мережевого пакета до відправлення, затримки передачі даних каналами зв’язку (Ping) та часу внутрішньої черги обробки на серверах Telegram, об'єкт керування має інерційні властивості та чисте транспортне запізнення. На основі серії експериментальних вимірювань часу відгуку було отримано передавальну функцію ОК:

$$W_o(s) = \frac{K_o}{T_o s+1} \cdot e^{-\tau s} \qquad (2.4)$$

де $K_o = 1.0$ – коефіцієнт передачі об'єкта у сталому режимі (кількість надісланих запитів дорівнює кількості прийнятих за відсутності втрат пакетів);

$T_o = 0.10$ с – стала часу, що характеризує внутрішню динаміку асинхронного буфера сервера FastAPI;

$\tau = 0.15$ с – час чистого транспортного запізнення мережевого каналу до серверів Telegram.

Тоді, математична модель об'єкта керування має вигляд:

$$W_o(s) = \frac{1.0}{0.10s+1} \cdot e^{-\times 0.15s} \qquad (2.5)$$

Для усунення похибки та забезпечення високої швидкодії оберемо пропорційно-інтегральний (ПІ) закон регулювання:

$$W_p(s) = K_p + \frac{K_i}{s} = K_p\left(1 + \frac{1}{T_i s}\right) \qquad (2.6)$$

Проведемо розрахунок коефіцієнтів регулятора, де бажана стала часу замкненої системи є $\tau_c = \tau = 0.15$ с.

Розрахуємо коефіцієнт пропорційної частини K_p:

$$K_p = \frac{T_o}{K_o \cdot (\tau_c + \tau)} = \frac{0.10}{1.0 \cdot (0.15 + 0.15)} = \frac{0.10}{0.30} \approx 0.333$$

Стала часу інтегрування T_i визначається за умовою:

$$T_i = \min(T_o, 4(\tau_c + \tau)) = T_o = 0.10 \text{ с})$$

Тоді коефіцієнт інтегральної частини регулятора K_i становить:

$$K_i = \frac{K_p}{T_i} = \frac{0.333}{0.10} = 3.33 \text{ с}^{-1}$$

Таким образом, передавальна функція синтезованого регулятора:

$$W_p(s) = 0.333 + \frac{3.33}{s} \qquad (2.7)$$

Таким чином, у даному підрозділі було сформовано математичну модель асинхронної черги повідомлень як об'єкта керування з транспортним запізненням та синтезовано оптимальний пропорційно-інтегральний регулятор. Перехід від безперервної передавальної функції до дискретного різницевого рівняння дозволив адаптувати теоретичні засади автоматичного керування для безпосередньої програмної реалізації в архітектурі вебсервера FastAPI. Впровадження розробленого алгоритму дозволяє динамічно оптимізувати інтенсивність відправки запитів, забезпечуючи максимальну швидкодію системи оповіщення в умовах жорстких лімітів зовнішнього Telegram API.

3 РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ ОПОВІЩЕННЯ

3.1 Розробка блок-схеми та алгоритму роботи серверної частини

При безпосередньо розробці системи оповіщення першим кроком є розроблення блок-схеми алгоритму роботи серверу. Для кращого розуміння блок-схему було розбито на 3 частини а саме: авторизація, головний цикл та цикл Telegram API.

На рисунку 3.1 зображена блок-схема циклу авторизації та виходу з аккаунту диспетчера.



Рисунок 3.1 – Блок-схема авторизації диспетчера

При авторизації першим кроком є введення даних для входу в аккаунт, якщо дані правильні то сторінка міняється і користувача буде перенесено до окремої сторінки до якої є доступ тільки у диспетчерів. Після успішної авторизації система очікує подальші дії від диспетчера. Блок-схема для дій диспетчера зображено на рисунку 3.2.

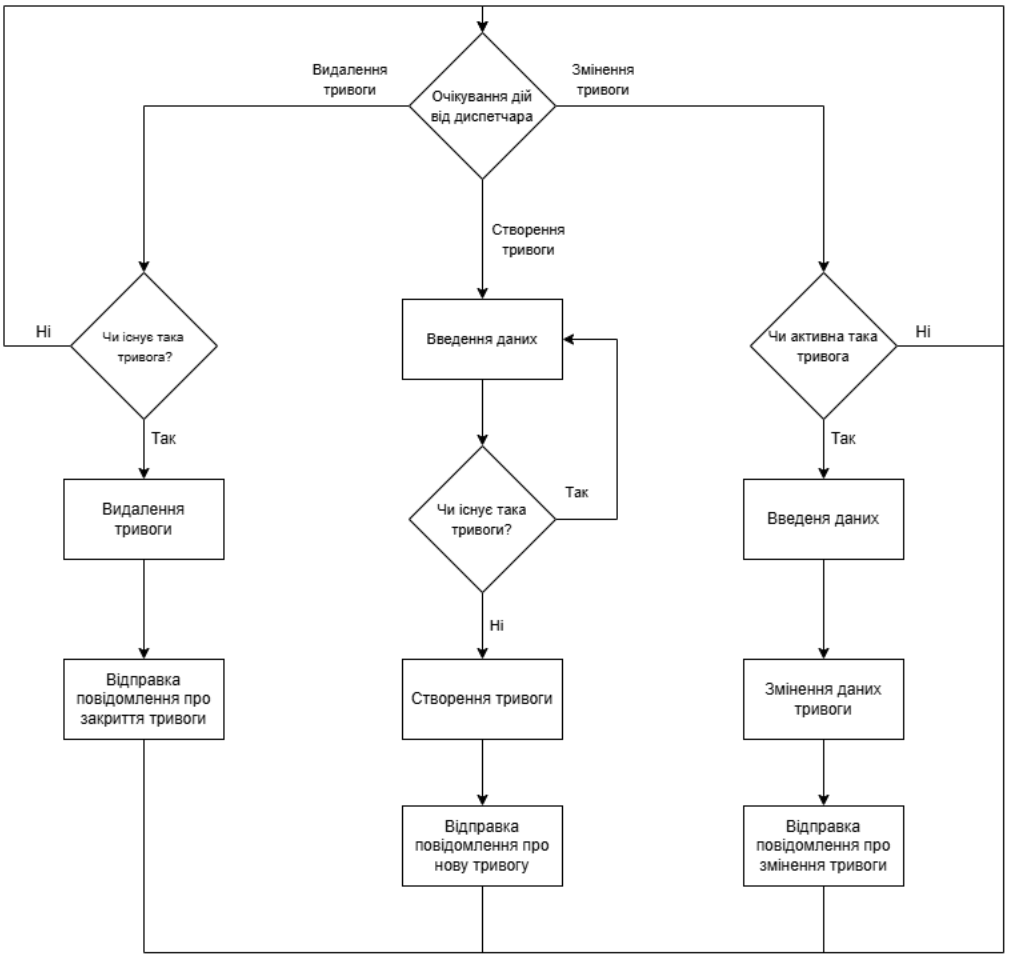


Рисунок 3.2 – Блок-схема при діях диспетчера

У диспетчера загалом є 3 (якщо вилучити вихід з аккаунту) дії а саме: створення, змінення та закриття тривоги.

При створенні тривоги диспетчер вводить дані щодо тривоги (назва, рівень, опис), система перевіряє чи є існуюча тривога з такою назвою яку надав диспетчер, якщо тривоги існує сервер повертає помилку про вже існуючу помилку, якщо-ж не існує то створює таку тривогу, додає її до бази даних та відправляє повідомлення у Telegram усім користувачам.

При зміненні тривоги першим кроком перевіряється чи активна тривога яку намагається змінити диспетчер, зроблено це для того щоб архівні дані з попередніми тривоги не можна було змінити, якщо тривога активна диспетчер оновлює дані для тривоги, сервер змінює дані у базі даних після чого відправляє повідомлення користувачам про змінну тривоги та що саме було змінено.

При закритті тривоги перевіряється чи існує і чи активна ця тривога, при існуванні активної тривоги її статус змінюється з «активна» на «закрита» після чого також відправляється повідомлення користувачам про закриття тривоги.

Для обробки повідомлень від користувачів у Telegram було розроблено окрему блок-схему (рис. 3.3).

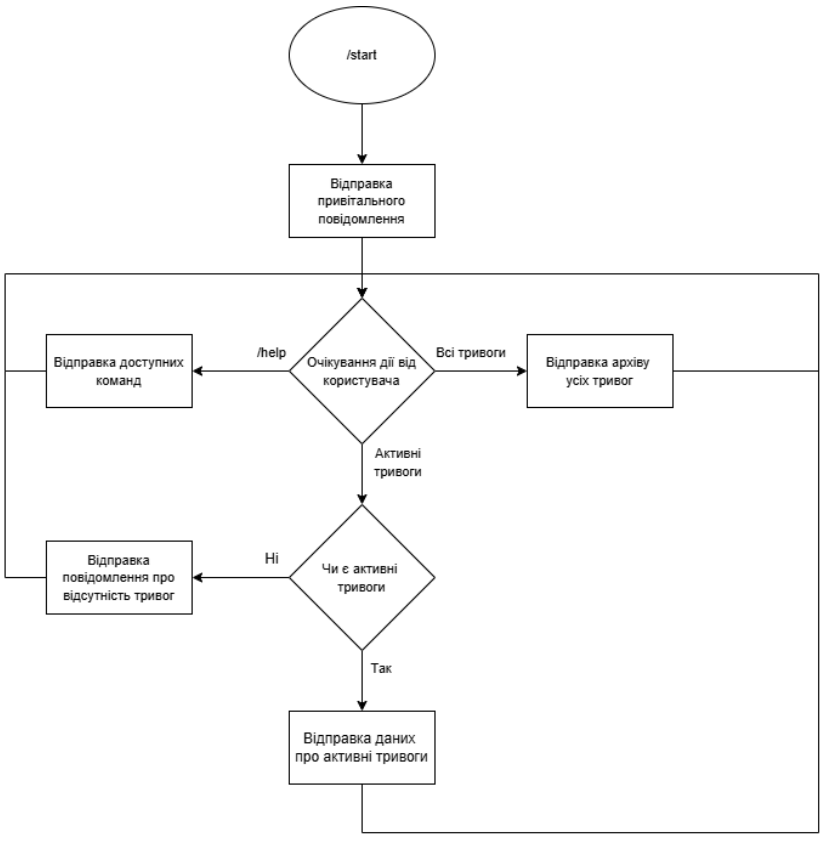


Рисунок 3.3 – Блок-схема алгоритму Telegram API

Користувач також загалом має 3 опції а саме: /help, активні та всі тривоги. Команда /help виведе доступні команди, а дії активні тривоги та всі тривоги говорять самі за себе виводять активні та всі тривоги.

3.2 Розробка системи автоматизованого оповіщення

На початковому етапі було розроблено ядро серверної частини на базі вебфреймворку FastAPI та налаштовано взаємодію з базою даних SQLite через ORM SQLAlchemy. Було спроєктовано базові моделі даних та реалізовано CRUD-операції для створення та закриття тривог. Також на цьому етапі було створено функціонал для отримання списку активних та архівних тривог.

Наступним кроком стало створення телеграм-бота з використанням асинхронної бібліотеки aiogram. Основним завданням бота стало проактивне сповіщення про створення та закриття тривог. Логіку взаємодії було побудовано за асиметричним принципом: бот працює в режимі «тільки для читання» і приймає команди лише для перегляду активних та архівних тривог, не маючи прав на їх зміну чи видалення.

Після стабілізації серверної частини розпочалася розробка фронтенду диспетчерської панелі. Для генерації динамічних сторінок на стороні сервера використано шаблонізатор Jinja2, що дозволило ефективно інтегрувати дані з бази в HTML-структуру. Візуальне оформлення реалізовано на базі CSS-фреймворку Bootstrap 5, що значно прискорило процес створення інтерфейсу. Клієнтську логіку, валідацію форм та інтерактивні елементи забезпечено за допомогою JavaScript, а взаємодію між клієнтом і сервером налаштовано через асинхронні HTTP-запити.

Наступний етап роботи зосередився на масштабуванні та модернізації серверної логіки. До опису тривог було додано параметр рівня загрози, а також впроваджено функціонал редагування існуючих записів. Для оптимізації роботи диспетчерів реалізовано можливість масового закриття всіх активних тривог одночасно.

На завершальному етапі розробки було реалізовано механізм авторизації та розмежування прав користувачів. Графічний інтерфейс було розділено на два середовища: загальну інформаційну панель для моніторингу поточної

ситуації без прав редагування та захищену панель диспетчера. Доступ до останньої надається тільки після валідації облікових даних у базі даних.

3.3 Опис роботи серверної частини та інтерфейсу

Як зазначено у пункті 2.2 при розробці серверної частини було обрано використовувати багатошарову архітектуру, що дозволяє розподілити функціонал файлів за шаром до якого він належить.

3.3.1 Шар роутерів або Ендпоінти FastAPI

Цей шар виконує функціонал «вхідної двері» від зовнішніх запитів до системи. Цей шар реалізовано завдяки асинхронному фреймворку FastAPI та його головною задачею виступає прийняття HTTP-запитів від панелі диспетчера, перевірити правильність введених даних та передати ці данні на обробітку до сервісної частини після чого сформувати кінцеву HTTP-відповідь.

Для запобігання попадання некоректних даних у ядро системи використовується бібліотека Pydantic. Типізація параметрів запиту дозволяють фреймворку автоматично валідувати дані, у випадку невірного формату запиту шар роутерів перериває обробку і повертає помилку 422 Unprocessable Entity (Необроблювана сутність).

Для роботи з базою даних на рівні ендпоінтів використовується механізм залежності Depends. Це дозволяє для кожного окремого HTTP-запиту створювати ізольовану сесію роботи с БД, яка автоматично закривається після завершення обробки запиту.

Код функції get_db з файлу dependency.py:

```
def get_db() -> Generator[Session, None, None]:
    db = SessionLocal()
    try:
        yield db
```

```
    finally:
        db.close()
```

Для подальшого впровадження Telegram треба щоб функції у яких буде реалізовано відправку повідомлення до месенджеру працювали у асинхронному режимі, це дозволить не переривати основний потік функції на відправку повідомленнь до усіх користувачів. Це досягається за допомогою вбудованій бібліотеці asyncio та асинхронним функціям.

Код функції create_alarm:

```
@router.post("/alarm")
async def create_alarm(alarm: CreateAlarmRequest, db: Session = Depends(get_db)):
    try:
        return await alarm_service.create_alarm(db, alarm)
    except AlarmAlreadyExistsError as e:
        raise HTTPException(status_code=400, detail=str(e))
```

3.3.2 Сервісний шар або бізнес логіка

У сервісному шарі системи оброблюються дані запиту які потрапляють сюди з роутерного шару. Його головною задачею становить прийняття рішень на основі перевірки умов та даних які були введені.

Якщо шар роутерів перевіряв формат вхідних даних, то шар сервісу перевіряє логіку процесів. Наприклад, перед закриттям окремої тривоги сервіс звертається до шару репозиторії для перевірки її існування та поточного статусу, якщо тривога вже закрита або навіть не існує у базі даних то сервіс перериває процес повертаючи виключення.

Сервісний шар повністю абстрагований від інтерфейсу вводу виводу. Завдяки цьому методи сервісу універсальні і можуть бути викликані як із FastAPI ендпоінтів так і із хендлерів Telegram-боту, що в свою чергу працює за принципом DRY (Don`t repeat yourself, Не повторюй себе).

Код функції close_alarm:

```
async def close_alarm(db: Session, alarm_id: int):
    alarm = alarm_repository.get_alarm_by_id(db, alarm_id)
    if not alarm or alarm.date_finished is not None:
        raise AlarmNotFoundError("This alarm not exists or already closed.")
    alarm.date_finished = datetime.datetime.now()
    db.commit()
    users = user_repository.get_all_users(db)
    await bot_sender.on_close_alarm(alarm = alarm, users = users)
    return AlarmResponse.model_validate(alarm)
```

3.3.3 Шар репозиторій

Шар репозиторій відповідає за спілкуванням з базою даних, а саме за додавання даних чи читання з БД. У цьому шарі нема ніяких перевірок та іншої логіки він існує виключно для того щоб відокремити систему від бази даних.

Для вилучення даних з бази даних об тривогах використовується бібліотека SQLAlchemy. Репозиторій виконує трансляцію викликів методів Python до оптимізованих SQL-запитів. Механізми ORM (Object-Relational Mapping, Об’єктно-реляційне відображення) автоматично перетворюють отримані строки таблиць в екземпляри Python-класів, з якими в свою чергу працює сервісний шар.

Архітектурною особливістю цього шару є делегування управління транзакціями до сервісного шару. Завдяки методу Identity Map, вбудованому в SQLAlchemy, сесія бази даних автоматично відстежує змінення атрибутів об’єктів вилучених репозиторієм. Це гарантує атомарність транзакції і перешкоджає частинному запису даних при виникненні помилок

Код функції create_alarm():

```
def create_alarm(db:Session, alarm_name:str, alarm_description) -> Alarm:
    alarm = Alarm(name = alarm_name, description= alarm_description, date_started = datetime.datetime.now())
```

```
    db.add(alarm)
    db.flush()
    return alarm
```

3.3.4 Інтеграція Telegram-боту

Для роботи з ботом використовувалася бібліотека aiogram що забезпечує асинхронну роботу боту одночасно з FastAPI. Це дозволяє серверу API та Telegram-боту функціонувати у одному циклі подій, не блокуючи один одного.

Хендлери бота спроектовані по аналогії з ендпоінтами FastAPI, вони не включають в себе бізнес логіки або роботи з базами даних. При отриманні повідомлення хендлери делегують виконання сервісному шару перевикористовуючи логіку.

Окрім реакційних моделей, бот також виконує функціонал push-оповіщень. Для цього реалізовано окремий модуль sender виконуючій роль відправника повідомлення. Наприклад, коли у системі створюється нова тривога, сервісний шар визиває методи sender при цьому запускаючи їх у асинхронному режимі для запобігання припиненню основного циклу.

Код функції on_create_alarm:

```
async def on_create_alarm(alarm: Alarm, users: list[UserTelegram]):
    text = (f"Нова загроза - {alarm.name}"
            f"\nРівень загрози: {alarm.level}")
    if alarm.date_finished is None:
        text += f"\nБудь ласка дійте за інструкцією нижче:\n{alarm.description}"
    for user in users:
        await bot.send_message(chat_id = user.id, text= text)
        await asyncio.sleep(0.04)
```

На рисунках 3.4-3.10 зображено приклад роботи Telegram-боту.

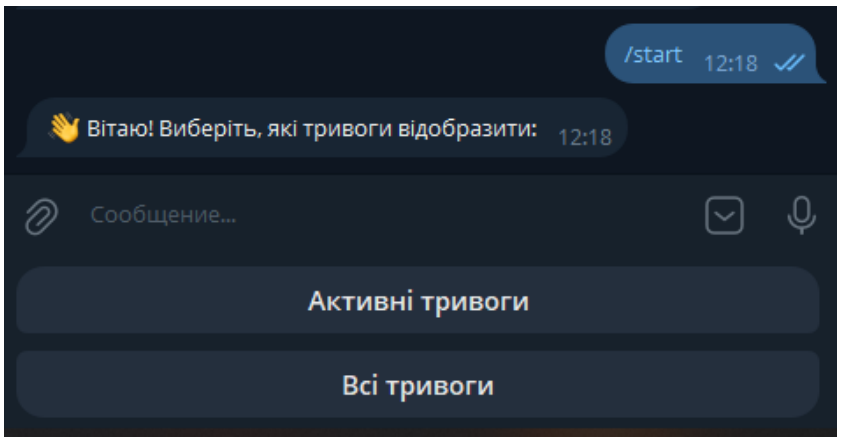


Рисунок 3.4 – Старт роботи телеграм-боту

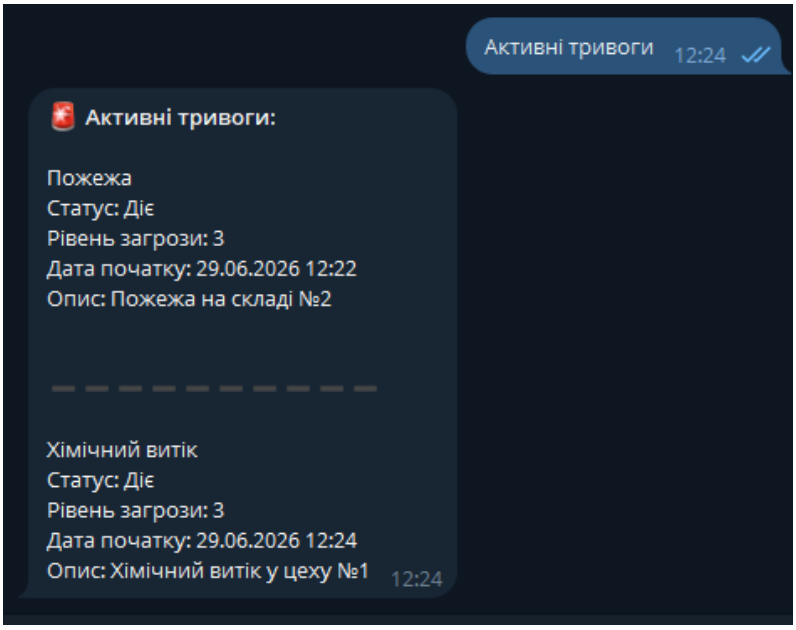


Рисунок 3.5 – Відповідь на запит «Активні тривоги»

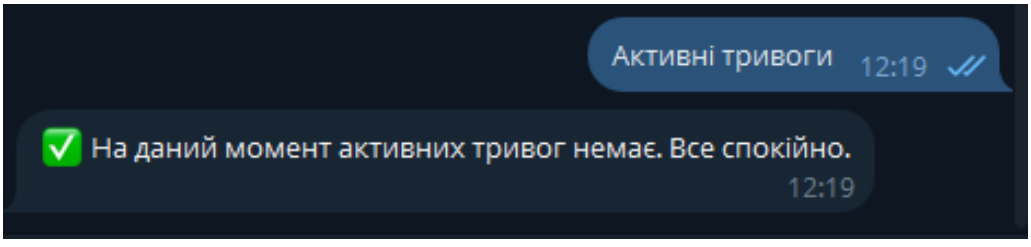


Рисунок 3.6 – Відповідь на запит «Активні тривоги» при відсутності тривог

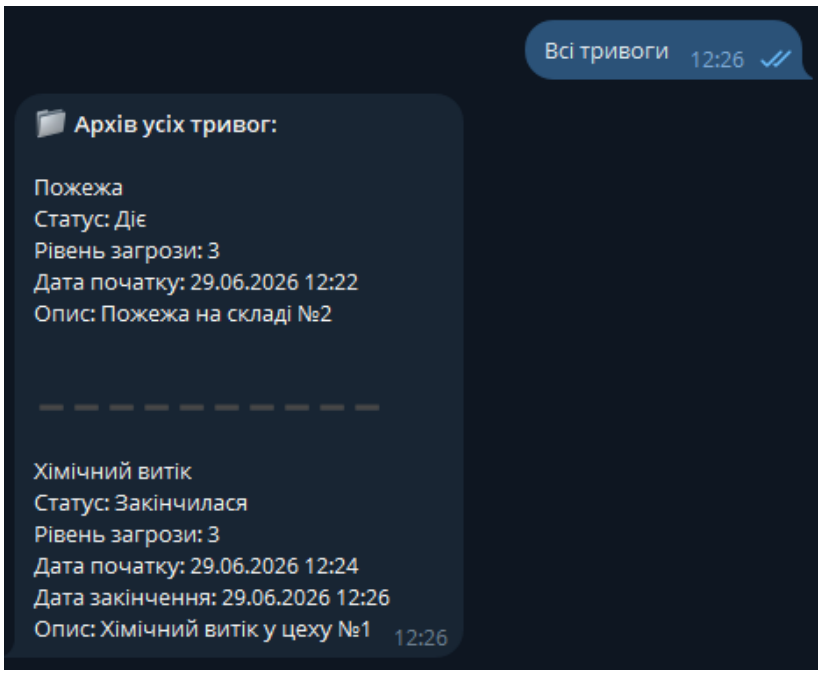


Рисунок 3.7 – Відповідь на запит «Всі тривоги»

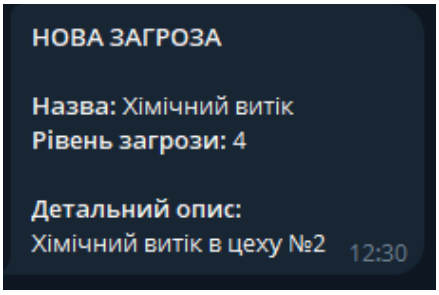


Рисунок 3.8 – Повідомлення про нову загрозу

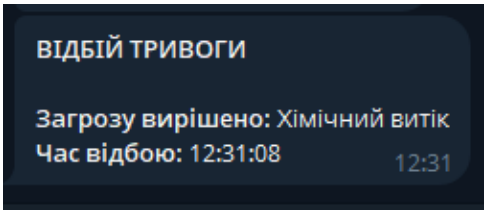


Рисунок 3.9 – Повідомлення вирішення загрози

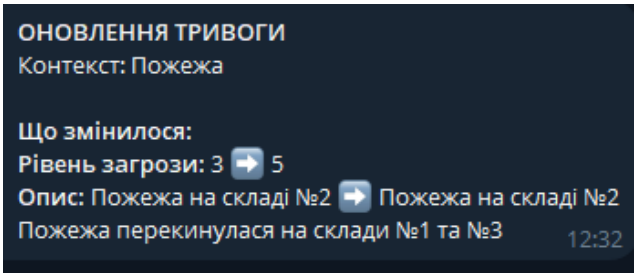


Рисунок 3.10 – Повідомлення про оновлення даних тривоги

3.3.5 Веб інтерфейс

Головною задачею клієнтської частини являється забезпечення надійної, швидкої та зручної взаємодії диспетчера з системою управління тривогами. З метою підвищення безпеки та логічного розмежування функцій, веб-інтерфейс було розділено на дві окремі сторінки: загальну інформаційну панель для візуального моніторингу ситуації та захищену панель диспетчера. Доступ до диспетчерської панелі надається виключно після проходження авторизації. Саме ця панель виступає у ролі головного пульту управління, дозволяючи диспетчеру відстежувати активні тривоги, швидко додавати нові, редагувати їх параметри або закривати існуючі.

Відображення інформації про тривоги відбувається завдяки шаблонізатору Jinja2. При запиті диспетчера до веб-панелі, сервер отримує актуальні данні щодо тривог і вставляє їх у HTML-каркас завдяки вбудованим циклам та умовним операторам шаблонізатора, після чого відправляє браузеру вже повністю готову сторінку.

Для візуального оформлення був використан CSS-фреймворк Bootstrap 5. Використання цього фреймворку приводить до адаптивності інтерфейсу, який буде корректно відображатися на широкоформатних моніторах так і на планшетах чи телефонах.

Взаємодія між серверною та клієнтською веб частиною була здійснена завдяки чистому JavaScript. Завдяки ньому перехоплюється автоматичне перезавантаження сторінки і просто оновлюються дані при створені чи змінені тривоги. Також саме завдяки ньому при створенні нової тривоги до сервера відправляється відповідний HTTP-запит

Інтерфейс виконано в мінімалістичному стилі з упором на швидке зчитування інформації. Зверху справа дві кнопки одна для додавання нових тривог, інша для одночасного закриття усіх поточних тривог. По центру розташовано таблицю з актуальними чи усіма тривогами на вибір, до кожної тривоги також додано її назву, рівень загрози, дату виникнення та її ID. Справа від кожної тривоги також можна закрити тривогу чи відкрити розширену

інформацію у окремому спливаючому віконці у якому також буде опис тривоги, у цьому-ж місці можна редагувати тривогу(рис. 3.12-3.15).

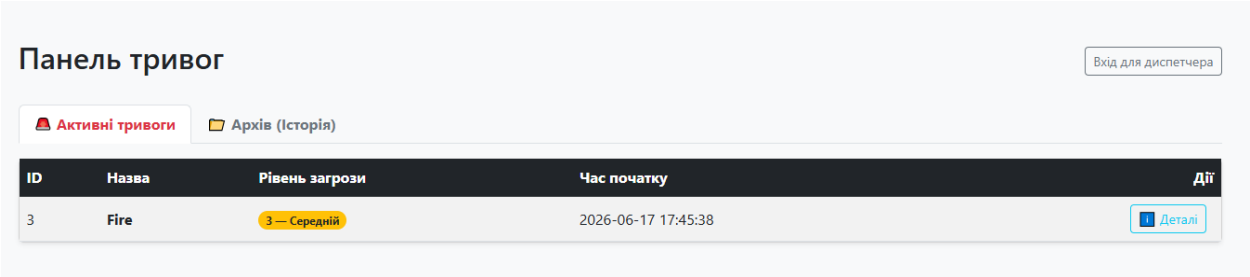


Рисунок 3.11 – Інтерфейс веб сторінки загальної інформаційної панелі

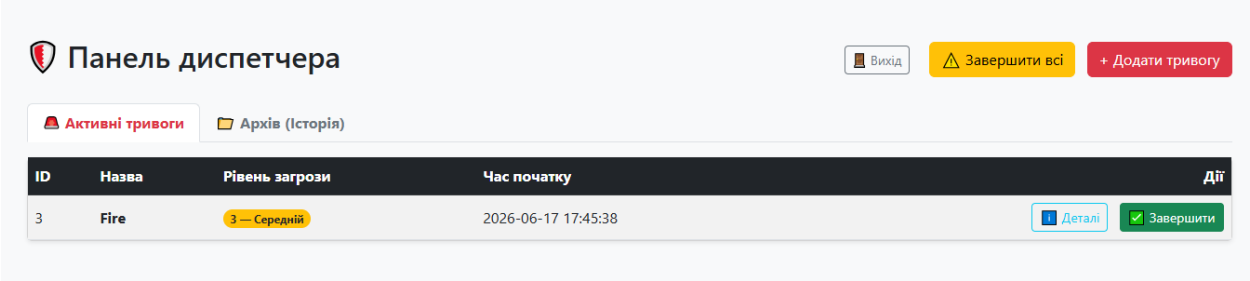


Рисунок 3.12 – Інтерфейс веб сторінки панелі диспетчера

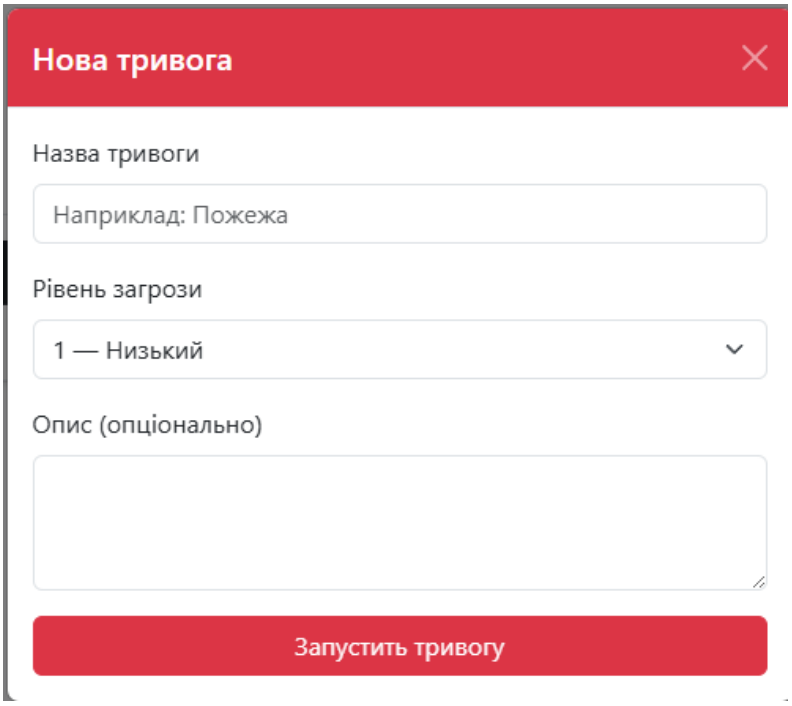


Рисунок 3.13 – Спливаюче вікно додавання тривоги

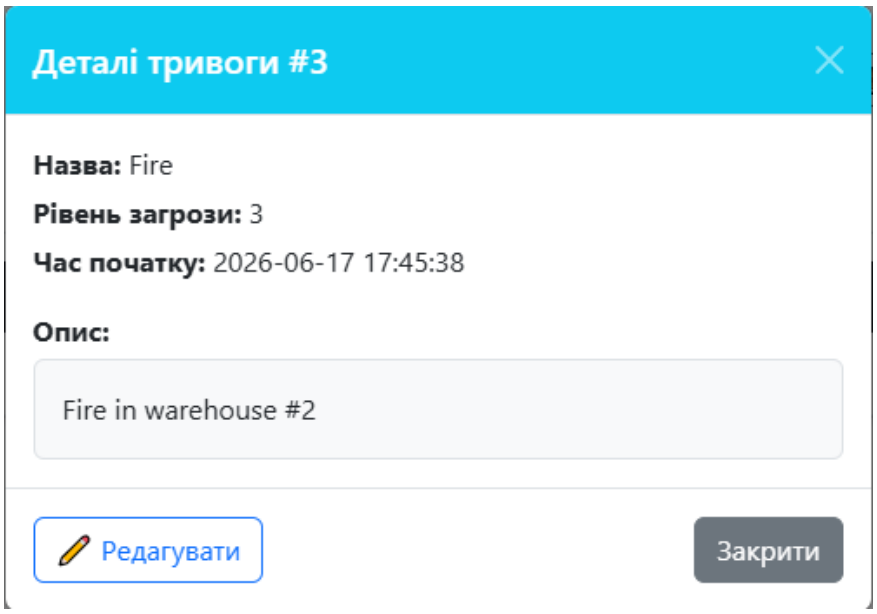


Рисунок 3.14 – Спливаюче вікно детальної інформації

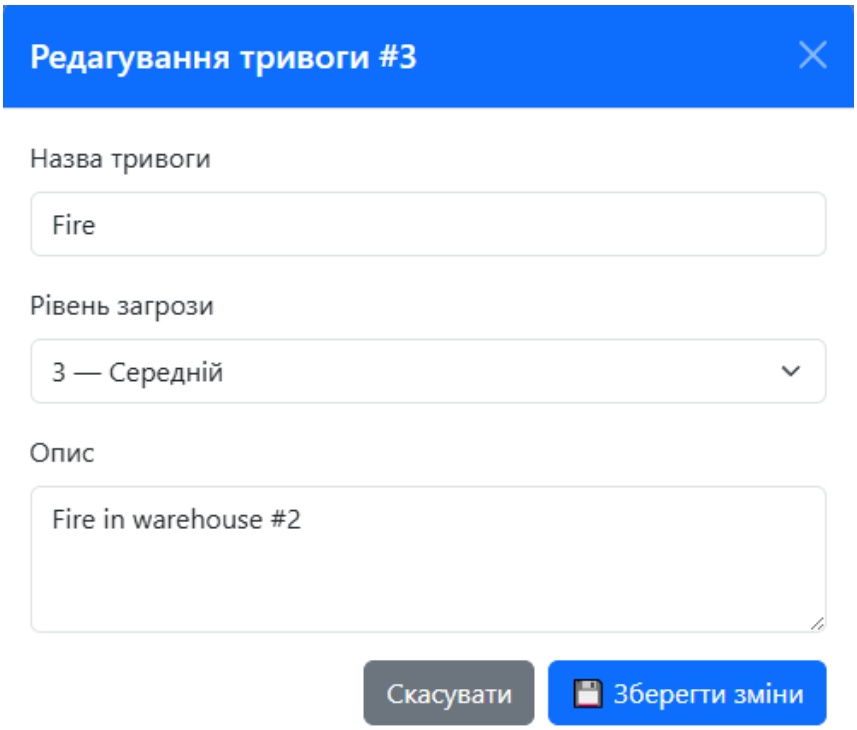


Рисунок 3.15 – Спливаюче вікно редагування інформації

3.3.6 Авторизація для доступу диспетчера

Для забезпечення санкціонного доступу до диспетчерської панелі було реалізовано сесійну автентифікацію на основі веб-кукі. При успішному введенні облікових даних диспетчера сервер перевіряє їх та звіряє з базою даних після чого генерує ідентифікатор сесії, який фіксується на стороні

бекенду, а клієнту повертається у HTTP-заголовку. Браузер автоматично зберігає цей ідентифікатор у локальному сховищі кукі та додає його до кожного наступного HTTP-запиту до сервера, що дозволяє системі розпізнавати поточного користувача та перевіряти його права без необхідності повторного надсилання логіна та пароля.

На рисунках 3.16-3.17 зображено сторінку для введення даних користувача.

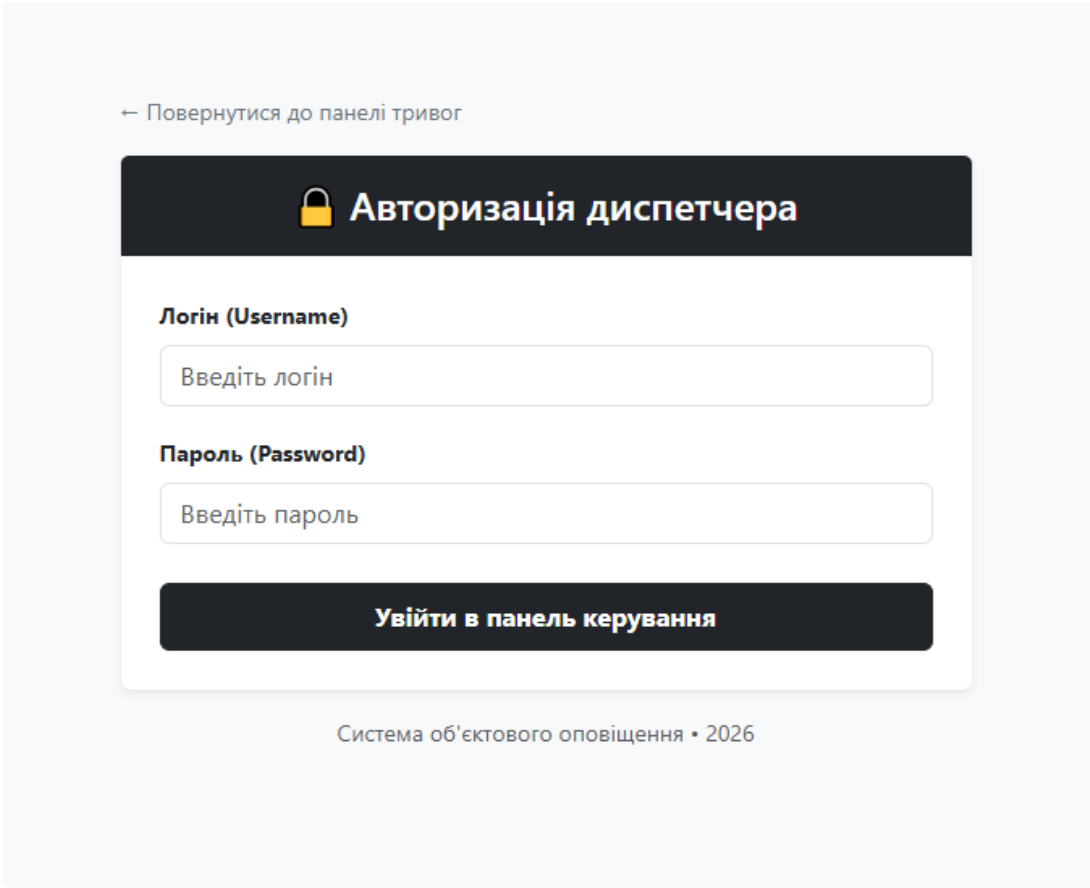


Рисунок 3.16 – Інтерфейс веб-сторінки авторизації

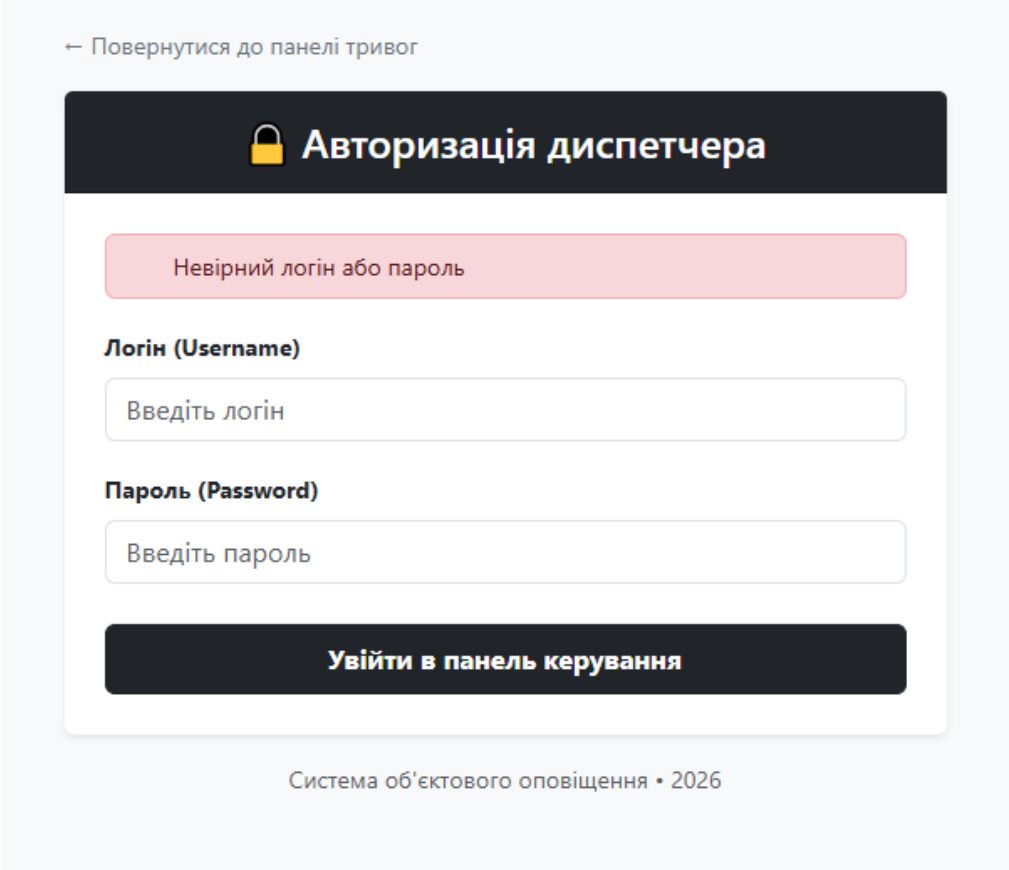


Рисунок 3.17 – Приклад помилки при авторизації

3.4 Тестування та аналіз швидкодії системи

Оскільки розроблена система призначена для екстреного оповіщення під час НС, критичним показником її ефективності є швидкодія, час необхідний для доставки повідомлення від моменту ініціації тривоги диспетчером до її фактичної обробки серверами месенджера.

Для отримання об’єктивних метрик було вирішено вимірювати чотири контрольні точки:

– t1(Ініціація на клієнті): Фіксується у браузері диспетчера у момент натискання кнопки відправки;

– t2 (Прийом запиту сервером): Фіксується маршрутизатором серверної частини одразу після отримання пакету. Різниця t2-t1 дозволяє вирахувати чисту мережеву затримку;

– t3 (Початок відправки повідомлень): Фіксується безпосередньо перед викликом методу відправки повідомлень у Telegram. Різниця t3-t2 відображає час внутрішньої обробки даних сервером;

– t4 (Успішна відправка повідомлень): Фіксується після успішного прийняття повідомлення серверами Telegram. Різниця t4-t3 є чистою затримкою зовнішнього API.

Після проведених тестових запусків були отримані середні показники для кожного з показників. Мережева затримка варіюється між 6-13 мс що дає середні 10 мс. Час внутрішньої обробки повідомлень сервером варіюється між 6-10 мс що дає середні 8 мс. Середня затримка зовнішнього API становить приблизно 75 мс. Загальний час розсилки для 51 користувача займає 6-6.5 секунд що дає приблизні 83 мс на відправку повідомлення для одного користувача (рис. 3.18).

```
2026-07-02 18:23:33,274 - INFO - Мережева затримка (Network Latency): 13 мс
2026-07-02 18:23:33,284 - INFO - Серверна затримка (Server Latency): 9 мс
2026-07-02 18:23:39,359 - INFO -
========================================
ЗВІТ ПРО МАСОВЕ ОПОВІЩЕННЯ (ID: 26)
----------------------------------------
Користувачів у базі: 51
Успішно доставлено: 51
Середня затримка Telegram API (t4 - t3): 75 мс
Загальний час розсилки (з урахуванням sleep): 6.075 сек
========================================
```

Рисунок 3.18 – Приклад логів відправки повідомлення

Ефективність та оперативність розробленої системи підтверджена результатами практичних замірів наскрізного часу. Тестування довело, що чистий час обробки та відправки повідомлення одному адресату через Telegram API становить близько 80-117 мс. Завдяки цьому гарантується доставка критичної інформації цільовій аудиторії об'єктового рівня всього за 6 секунд, що значно перевищує швидкість традиційних ручних методів оповіщення.

3.5 Охорона праці

У цьому розділі розглянуто умови праці розробника та диспетчера системи оповіщення, які зазнають значного нервово-емоційного і зорового навантаження при роботі за комп'ютером. Відповідно до НПАОП 0.00-7.15-18, основними небезпечними факторами є фізичні (електромагнітне випромінювання, поганий мікроклімат, ризик ураження електричним струмом) та психофізіологічні (перевтома очей і м’язів). Щоб зменшити їхній вплив, потрібно правильно організувати робоче місце.

Робоче місце має відповідати ергономічним вимогам: площа мінімум 6 м², об’єм не менше 20 м³. Стіл повинен мати регулювання висоти (680-800 мм), а крісло – механізм підйому та повороту, що підтримує правильну поставу і спину. Монітор слід розташувати на відстані 600-700 мм від очей, щоб уникнути напруги шиї. Згідно з ДСН 3.3.6.042-99, для легких фізичних робіт (категорія Іа) вентиляція і кондиціонування мають підтримувати оптимальний мікроклімат: температуру 22-25 °С, вологість 40–60% і швидкість повітря не більше 0,1 м/с.

Щоб зменшити зорове навантаження, відповідно до ДБН В.2.5-28:2018, організовується комбіноване освітлення: природне світло без прямого попадання сонячних променів на екран і загальне штучне LED-освітлення без мерехтіння. Рівень освітленості на робочій поверхні має бути 300-500 люкс, а поблизу монітора не перевищувати 300 люкс. Електробезпека забезпечується трипроводною мережею з заземленням і пристроями захисного відключення на 30 мА. Оскільки приміщення відноситься до категорії «В» за пожежною небезпекою, воно оснащене автоматичною пожежною сигналізацією з датчиками диму та вуглекислотними вогнегасниками, які підходять для гасіння електрообладнання під напругою.

Режим роботи встановлений так, щоб знизити зорове та емоційне навантаження. За НПАОП 0.00-7.15-18 після кожної години роботи за комп’ютером передбачені перерви на 10-15 хвилин. У цей час рекомендується

виконувати вправи для розслаблення очних м’язів і м’язів спини та плечового пояса.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було досліджено, розроблено та реалізовано автоматизовану клієнт-серверну систему оповіщення населення при надзвичайних ситуаціях. Робота охоплює повний цикл розробки програмного забезпечення від аналізу сучасних систем до програмної реалізації.

Аналіз сучасних систем оповіщення показав суттєві недоліки традиційних засобів оповіщення, таких як радіостанції та звукові сирени, які показали недостатню гнучкість в умовах сучасних кризових ситуацій. Це дозволило обґрунтувати доцільність інтеграції веб-застосунків та месенджерів для організації швидкого каналу зв’язку, а також сформувати базові архітектурні вимоги до системи.

Було успішно здійснено проектування загальної архітектури системи та проведено вибір технічних засобів і стеку технології розробки. Для реалізації серверної логіки було використано мову програмування Python та веб-фреймворк FastAPI. Для зберігання даних було використано SQLite з використанням ORM SQLAlchemy для надійності та структурованості даних. Клієнтську фронтенд частину було розроблено на основі HTML, CSS Bootstrap 5, JavaScript та Jinja2. Крім того, було побудовано функціональну схему взаємодії компонентів.

Практична частина включала реалізацію таких ключових етапів:

– розроблено блок-схеми роботи серверної логіки;

– створено інтерактивну диспетчерську панель з мінімалістичним графічним інтерфейсом;

– інтегровано Telegram для відправки повідомлень користувачам

– розглянуто питання охорони праці.

Таким чином, в ході виконання кваліфікаційної роботи було досягнуто мети виконання роботи. Отримані результати мають високу практичну

цінність. Створений програмний продукт може бути адаптована для виконання більш складних запитів та автоматизації створення тривог на основі датчиків.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008-15. Документація. Звіти у сфері науки та техніки. Структура та правила оформлення. Введ. 2015-06-22. К. Держстандарт України, 2017. - 29 с.

2. Навчальний посібник з підготовки кваліфікаційної роботи бакалавра для здобувачів вищої освіти денної і заочної форм навчання спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» освітньої програми «Автоматизація та комп'ютерно-інтегровані технології» : Навчальний посібник / І. Ш. Невлюдов, В. А. Андрусевич, О. В. Токарєва, С. П. Новоселов, О. В. Сичова. – Харків : Видавництво Іванченка І. С., 2022. 151 с.

3. Методичні вказівки з підготовки кваліфікаційної роботи для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології освітньої програми «Системна інженерія» / Упоряд.: І.Ш. Невлюдов, О. М. Цимбал, О. В. Токарєва, А. І. Бронніков. Харків: ХНУРЕ, 2023. 65 с.

4. Постанова Кабінету Міністрів України від 27.09.2017 № 733 «Про затвердження Положення про організацію оповіщення про загрозу виникнення або виникнення надзвичайних ситуацій та зв'язку у сфері цивільного захисту». – URL: https://ips.ligazakon.net/document/kp170733?an=1&ed=0000_00_00 (дата звернення: 24.04.2026).

5. Шведун В. О., Ігнатьєв О. М., Поступна О. В. Проблеми функціонування механізмів державного управління щодо оперативного моніторингу стану потенційно небезпечних об'єктів. Актуальні проблеми державного управління. 2022. № 2 (61). С. 57–70. DOI: https://doi. org/10.26565/1684-8489-2022-2-04.

6. Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); LTE; 5G; Technical realization of

the Short Message Service (SMS) (3GPP TS 23.040 version 16.0.0 Release 16). – URL: https://www.etsi.org/deliver/etsi_ts/123000_123099/123040/16.00.00_60/ts_123040v160000p.pdf (дата звернення: 26.04.2026).

7. Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); Technical realization of Cell Broadcast Service (CBS) (3GPP TS 23.041 version 15.2.0 Release 15). – URL: https://www.etsi.org/deliver/etsi_ts/123000_123099/123041/15.02.00_60/ts_123041v150200p.pdf (дата звернення: 26.04.2026).

8. FCM Architectural Overview // Firebase. – URL: https://firebase.google.com/docs/cloud-messaging/fcm-architecture (дата звернення: 26.05.2026).

9. Bots FAQ. // Telegram. – URL: https://core.telegram.org/bots/faq (дата звернення: 26.04.2026).

10. J-Alert disaster warning technology in Japan. // Center for public Impact. – URL: https://centreforpublicimpact.org/public-impact-fundamentals/j-alert-disaster-warning-technology-in-japan/ (дата звернення: 15.04.2026).

11. Early warning system. Nishimura Yosuke Director FDMA. – URL: https://www.itu.int/en/ITU-D/Regional-Presence/AsiaPacific/Documents/Events/2025/Sendai%202025/PPTs/Session%202_EarlyWarningSystem_NISHIMURO.pdf (дата звернення: 15.04.2026).

12. J-arāto no gaiyō [Огляд системи J-Alert] – URL: https://web.archive.org/web/20170918061721/http://www.fdma.go.jp/html/intro/form/pdf/kokuminhogo_unyou/kokuminhogo_unyou_main/J-ALERT_gaiyou_h28.pdf (дата звернення: 15.04.2026).

13. ShakeAlert // Wikipedia. – URL: https://en.wikipedia.org/wiki/ShakeAlert (дата звернення: 16.04.2026).

14. License to Operate (LtO) Technical Partners // ShakeAlert. – URL: https://www.shakealert.org/system-information/license-to-operate-lto-technical-partners/ (дата звернення: 16.04.2026).

15. Повітряна тривога (застосунок). // Wikipedia. – URL: https://uk.wikipedia.org/wiki/Повітряна_тривога_(застосунок) (дата звернення: 17.04.2026).

16. «Повітряна тривога» та бункери із запуском сирени. Як створювали один з найбільш затребуваних застосунків в Україні // Dou. – URL: https://dou.ua/lenta/interviews/air-alarm-application/ (дата звернення: 17.04.2026).

17. Build Advanced Alarming Systems with Drag-and-Drop Ease. // Inductive Automation. – URL: https://inductiveautomation.com/solutions/alarm-notification# (дата звернення: 27.06.2026).

18. Welcome to Python. // Python. – URL: https://www.python.org (дата звернення: 02.05.2026).

19. About Python. // Python. – URL: https://www.python.org/about/ (дата звернення: 02.05.2026).

20. Applications for Python. // Python. – URL: https://www.python.org/about/apps/ (дата звернення: 02.05.2026.)

21. FastAPI. – URL: https://fastapi.tiangolo.com/uk/ (дата звернення: 02.05.2026).

22. Функціональні можливості. // FastAPI. – URL: https://fastapi.tiangolo.com/uk/features/ (дата звернення: 02.05.2026)

23. About SQLite. // SQLite. – URL: https://sqlite.org/about.html (дата звернення: 02.05.2026)

24. aiogram. – URL: https://aiogram.dev (дата звернення: 02.05.2026)

25. Jinja2 Documentation. – URL: https://jinja.palletsprojects.com/en/stable/ (дата звернення 02.05.2026).

26. Introduction. // Bootstrap 5. – URL: https://getbootstrap.com/docs/5.0/getting-started/introduction/ (дата звернення: 02.05.2026).

27. JavaScript. // Wikipedia. – URL: https://uk.wikipedia.org/wiki/JavaScript (дата звернення: 02.05.2026).

28. НПАОП 0.00-7.15-18. Вимоги безпеки та захисту здоров'я під час використання виробничого обладнання працівниками : затверджено Наказом Міністерства соціальної політики України від 28.12.2017 № 2072. URL: https://zakon.rada.gov.ua/laws/show/z0097-18 (дата звернення: 15.05.2026).

29. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень : затверджено Постановою Головного державного санітарного лікаря України від 01.12.1999 № 42. URL: https://zakon.rada.gov.ua/laws/show/va042282-99 (дата звернення: 15.05.2026).

30. ДБН В.2.5-28:2018. Природне і штучне освітлення. Київ : Мінрегіонбуд України, 2018. 133 с.