

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 проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МОНІТОРИНГУ
СТАНУ ЛЮДЕЙ ПОХИЛОГО ВІКУ
DEVELOPMENT OF SOFTWARE FOR MONITORING THE CONDITION OF
ELDERLY PEOPLE

Виконав: студент 4 курсу, групи КНТ-132

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

КУЩ Д.І.

(ПРІЗВИЩЕ та ініціали)

Керівник ПАРХОМЕНКО А.В.

(ПРІЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(ПРІЗВИЩЕ та ініціали)

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження.
2. Проєктування програмного забезпечення. 3. Розробка програмного забезпечення. 4. Тестування та експериментальне дослідження програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ПАРХОМЕНКО А.В., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання «20» квітня 2026 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.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури системи.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	ПЗ, Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент

Дмитро КУЩ
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

Анжеліка ПАРХОМЕНКО
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 133 с., 13 табл., 20 рис., 2 дод., 35 джерел.

СИСТЕМА МОНІТОРИНГУ БЕЗПЕКИ, ДЕТЕКТУВАННЯ ПАДІНЬ, ТЕЛЕГРАМ-БОТ, КОМП'ЮТЕРНИЙ ЗІР

Об'єкт дослідження – процеси програмної інженерії щодо створення програмних засобів для нагляду за людьми похилого віку.

Предмет дослідження – методи та програмні засоби детектування падінь.

Мета роботи – підвищення точності детектування падінь на основі створення програмного забезпечення для запобігання несподіваним ситуаціям, що пов'язані зі здоров'ям людей похилого віку.

Матеріали, методи та технічні засоби: OpenCV, YOLO, мова програмування Python, бібліотека pyTelegramBotAPI для реалізації Telegram-бота, реляційна база даних PostgreSQL, персональний комп'ютер з підключеною відеокамерою під управлінням Windows 11.

Результати. Створено програмну систему моніторингу безпеки, яка на основі відео потоку з камери із застосуванням OpenCV та YOLO в режимі реального часу виконує виявлення людини та аналіз її рухів для визначення можливого падіння, з подальшим формуванням тривожних повідомлень та їх передачею через телеграм-бот.

Висновки. Розроблено телеграм-бот для нагляду над людьми похилого віку, який допомагає уникнути серйозних наслідків та патових ситуацій.

Галузь використання – нагляд за людьми похилого віку в медичних закладах лікарями, співробітниками будинку для людей похилого віку або в домашніх умовах родичами.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 133 pages, 13 tables, 20 figures, 2 appendixes, 35 sources.

SAFETY MONITORING SYSTEM, FALL DETECTION, TELEGRAM BOT, COMPUTER VISION

Object of the study is software engineering processes for the creation of software tools for monitoring elderly people.

Subject of the study is methods and software tools for fall detection.

The purpose of the work is to improve the accuracy of fall detection by developing software to prevent unexpected situations related to the health of elderly people.

Materials, methods and technical tools: OpenCV, YOLO, the Python programming language, the pyTelegramBotAPI library for implementing a Telegram bot, the PostgreSQL relational database, a personal computer with a connected video camera running the Windows 11 operating system.

Results. A safety monitoring software system has been created which, based on a video stream from camera using OpenCV and YOLO in real time, detects a person and analyses their movements to identify a possible fall, followed by the generation of alarm messages and their transmission via a Telegram bot.

Conclusions. A Telegram bot has been developed for monitoring elderly people, which helps to avoid serious consequences and critical situations.

The field of application is monitoring of elderly people in healthcare facilities by doctors, staff at care homes, or at home relatives.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	7
Вступ.....	8
1 Аналіз проблеми та постановка завдань дослідження	9
1.1 Особливості забезпечення безпеки людей похилого віку	9
1.2 Огляд існуючих систем моніторингу та детектування падінь	9
1.3 Методи виявлення падінь.....	11
1.4 Постановка завдань дипломної роботи.....	16
2 Проєктування програмного забезпечення	18
2.1 Вимоги до системи моніторингу	18
2.2 Загальна архітектура системи	23
2.3 Проєктування структури даних та бази даних.....	27
2.4 Вибір технологій та інструментів розробки.....	31
2.5 Проєктування інтерфейсу користувача	39
2.6 Висновок до розділу 2	43
3 Розробка програмного забезпечення.....	44
3.1 Реалізація Telegram - бота для моніторингу безпеки	44
3.2 Реалізація механізму детектування падінь	48
3.3 Реалізація роботи з базою даних	51
3.4 Інтеграція компонентів системи.....	54
3.5 Висновок до розділу 3	56
4 Тестування та експериментальне дослідження програми	58
4.1 Тестування роботи користувача та опікуна	58
4.2 Експериментальне дослідження системи	64
4.1 Висновок до розділу 4	65
Висновки	67
Перелік джерел посилання	68
Додаток А Текст програми.....	71
Додаток В Слайди презентації.....	135

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

- API – Application Programming Interface;
CV – Computer Vision;
ID – Identifier;
ML – Machine Learning;
PIL – Python Imaging Library;
R-CNN – Region-based Convolutional Neural Network;
SSD – Single Shot MultiBox Detector;
YOLO – You Only Look Once;
ПЗ – програмне забезпечення.

ВСТУП

У сучасному суспільстві спостерігається тенденція до зростання частки людей похилого віку, що зумовлює підвищену увагу до питань їхньої безпеки та якості життя. Однією з найбільш серйозних загроз для цієї категорії населення є падіння, які можуть призводити до тяжких травм, втрати самостійності та навіть летальних наслідків. За даними міжнародних досліджень, падіння є однією з основних причин травматизму серед людей віком понад 60 років [1].

У зв'язку з цим особливої актуальності набуває розробка сучасних систем моніторингу, здатних автоматично виявляти небезпечні ситуації, зокрема падіння, та своєчасно повідомляти про них. Використання інформаційних технологій, сенсорних пристроїв та методів штучного інтелекту дозволяє значно підвищити ефективність таких систем.

Мета роботи – підвищення точності детектування падінь на основі створення програмного забезпечення для запобігання несподіваним ситуаціям, що пов'язані зі здоров'ям людей похилого віку.

Дипломна робота складається зі вступу, чотирьох розділів, висновків, списку джерел посилання та додатків. У першому розділі проведено аналіз проблеми та розглянуто існуючі рішення. У другому розділі описано проєктування системи. Третій розділ присвячено реалізації програмного забезпечення. У четвертому розділі наведено результати тестування та аналіз ефективності роботи системи.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Особливості забезпечення безпеки людей похилого віку

В нашому світі, падіння є 2 причиною смерті в результаті ненавмисних травм. За статистикою, понад 680 000 випадкових смертей щорічно трапляються в результаті падінь, в основному це люди похилого віку(старше 60 років), які страждають на аритмію, анемію, діабет, артрит, розлади щитовидної залози, втрата гостроти зору та слуху і ще багатьох інших хвороб. У таких людей може несподівано запаморочитись в голові, і також вони можуть втратити свідомість під час сидіння, стояння, або ходьби через накопичені хвороби свого віку та вживання деяких ліків, через що їм знадобиться невідкладна медична допомога. Частіше всього це тяжкі травми, як ушиби, травми голови, та переломи шийки стегна, все це пов'язано з фізичними, когнітивними та сенсорними змінами [1]-[2].

Так як життя навкруги швидко змінюється, а люди похилого віку не встигають пристосуватись до цих змін. Велика кількість літніх людей не повідомляють про падіння, так як вони вважають це процесом старіння, тому вони бояться про це повідомляти своїм рідним, а згодом бути обмеженими в своїх повсякденних справах, а також опинитися в медичному закладі. В результаті цього є ризик того, що падіння можуть повторитись, і призвести до більш тяжких наслідків, що також є причиною великого навантаження медичних закладів. І це навантаження збільшується завдяки зростанню старіючого населення [1]-[2].

1.2 Огляд існуючих систем моніторингу та детектування падінь

Старіння населення – це невід'ємна частина життя людства. За статистикою до 2050 року люди старше 60 років складатимуть 35% населення в світі. І сучасні технології такі як системи виявлення падінь та автоматичної активації тривоги покращують якість життя. За останнє

десятиліття було розроблено багато рішень по розробці дистанційного моніторингу та детектувань падінь. Багато людей працювали над цими системами. Сучасні технології такі як системи виявлення падінь та автоматичної активації тривоги покращують якість життя [3].

Всього існує 4 основних типів системи: носимі, візуальні, середовищні та гібридні, наведено на рисунку 1.1 [4].

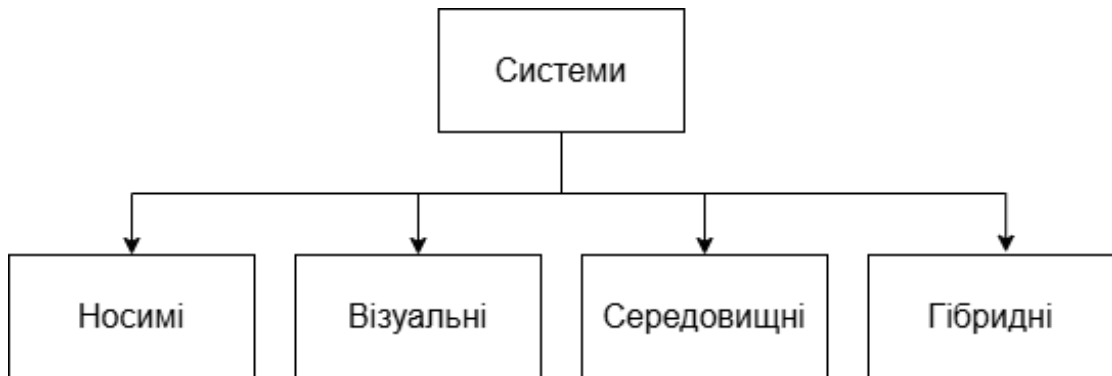


Рисунок 1.1 – Основні типи систем [4]

Носимі системи є одним з найбільш поширених і доступних рішень. Вони базуються на використанні датчиків, таких як акселерометри та гіроскопи, які вбудовані у смарт – годинники, браслети або мобільні пристрої. Ці датчики дозволять відстежувати рухи користувача та визначити різкі положення тіла, характерні для падіння. Основними перевагами носимих систем є відносно низька вартість, висока точність та мобільність. Водночас їх недоліком є необхідність постійного носіння пристрою [5].

Візуальні системи моніторингу базуються на використанні камер та алгоритмів комп’ютерного зору. У таких системах аналізується відео потоки для виявлення характерних поз або рухів, які відповідають падінню. Сучасні підходи часто використовують методи машинного навчання та глибинного навчання, що дозволяє значно підвищити точність розпізнавання подій. Однак основними недоліками є висока вартість реалізації та питання конфіденційності [4].

Середовищні системи передбачають використання різноманітних датчиків, розташованих у приміщенні, наприклад датчиків руху, тиску або інфрачервоних сенсорів. Такі системи дозволяють відстежувати активність людини без безпосереднього контакту з нею. Перевагою є зручність для користувача, проте недоліками є складність встановлення та обмеженість області покриття [6].

Останню категорію становлять гібридні системи, які поєднують кілька підходів одночасно. Вони забезпечують більш високу точність та надійність детектування падінь за рахунок комбінування різних джерел даних, але є складнішими в реалізації [4].

Аналіз показав, що ще жоден із підходів не є універсальним і кожен має свої переваги та обмеження. Таким чином, це свідчить про те, що системи детектування падінь мають активний розвиток та необхідність подальших досліджень [4].

1.3 Методи виявлення падінь

Ключовим елементом систем моніторингу безпеки людей похилого віку є методи виявлення падінь. Вони спрямовані на автоматичне розпізнавання ситуації, у яких людина зазнає падіння, з подальшим формуванням сигналу тривоги. Сучасні методи можна поділити на чотири основні групи: порогові, методи машинного навчання, методи комп'ютерного зору та гібридні методи [4].

1.3.1 Порогові методи

Порогові методи базуються на аналізі даних з інерційних сенсорів (акселерометрів і гіроскопів). Основна ідея полягає у визначенні падіння за різкою зміною прискорення тіла.

Ключовим показником є модуль вектора прискорення за формулою (1.1):

$$A = \sqrt{a_x^2 + a_y^2 + a_z^2}, \quad (1.1)$$

де a_x , a_y , a_z -компоненти прискорення вздовж відповідних осей.

Падіння фіксується, якщо виконується умова (1.2):

$$A > A_{threshold}, \quad (1.2)$$

де $A_{threshold}$ - порогове значення.

Переваги:

- простота реалізації;
- низькі обчислювальні витрати.

Недоліки:

- висока ймовірність хибних спрацювань;
- складність підбору універсального порогу [5].

1.3.2 Методи машинного навчання

Метод машинного навчання дозволяють значно підвищити точність виявлення падінь за рахунок аналізу складних патернів руху. Вони використовують навчальні вибірки, що містять приклади падінь і звичайної активності.

Класифікація виконується за допомогою функції за формулою (1.3):

$$y = f(x), \quad (1.3)$$

де x -вектор ознак (дані сенсорів);

у- клас (падіння або нормальний стан).

Для оцінки якості моделі використовують наступні метрики.

Точність розраховується за формулою (1.4):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} , \quad (1.4)$$

Чутливість визначається за формулою (1.5):

$$Sensitivity = \frac{TP}{TP + FN} , \quad (1.5)$$

де *TP*- правильно визначені падіння;

TN- правильно визначені нормальні стани;

FP- хибні спрацьовування;

FN- пропущені падіння.

Переваги:

- висока точність;
- адаптивність до різних умов.

Недоліки:

- потреба у великій кількості даних;
- більші обчислювальні витрати [4].

1.3.3 Методи комп'ютерного зору

Методи комп'ютерного зору базуються на аналізі відеоданих з камер. Вони використовують алгоритми розпізнавання поз і рухів людини.

Одним із підходів є аналіз співвідношення висоти та ширини силует людини за формулою (1.6):

$$R = \frac{H}{W}, \quad (1.6)$$

де H - висота силуету;

W - ширина.

Падіння може бути визначене при різкому зменшенні цього співвідношення.

Сучасні системи також використовують нейроні мережі, що дозволяє враховувати часову динаміку рухів.

Переваги:

- висока точність;
- відсутність необхідності носимих пристроїв.

Недоліки:

- залежність від умов освітлення;
- проблеми конфіденційності [7].

1.3.4 Гібридні методи

Гібридні методи поєднують кілька підходів, наприклад використання сенсорів разом із відеоаналізом. Це дозволяє підвищити надійність системи за рахунок комбінування різних джерел інформації.

Такі системи є найбільш перспективними, оскільки забезпечують баланс між точністю та зручністю використання [4].

Таким чином існує декілька підходів до виявлення падінь, кожен з яких має свої переваги та обмеження. Найпростішими є порогові методи. Однак вони поступаються за точністю методам машинного навчання та комп'ютерного зору. Найбільш ефективними вважаються гібридні системи, що поєднують різні технології.

1.3.5 Порівняльний аналіз методів виявлення падінь

Для обґрунтування вибору алгоритму детекції було проведено порівняльний аналіз методів виявлення падінь, який подано в таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз методів детектування падінь [4]-[7]

Метод	Принцип роботи	Переваги	Недоліки	Точність
Порогові методи	Аналіз різких змін прискорення, порівняння з порогом	Простота реалізації, низькі обчислювальні витрати	Висока кількість хибних спрацювань, складність вибору порогу	Низька / середня
Методи машинного навчання	Класифікація на основі ознак сенсорних даних	Висока точність, адаптивність до умов	Потреба у великих даних, висока обчислювальна складність	Висока
Комп'ютерний зір	Аналіз відео, поз та силуети людини	Висока точність, не потребує носимих пристроїв	Залежність від освітлення, проблеми конфіденційності	Висока
Гібридні методи	Комбінація сенсорів + відео/ML	Найвища надійність зменшення помилок	Складність реалізації, висока вартість	Дуже висока

1.3.6 Вибір методу для реалізації системи

У даній роботі для реалізації системи моніторингу безпеки обрано метод комп'ютерного зору на основі відео спостереження. Даний підхід дозволяє автоматично виявляти падіння без використання носимо пристроїв та забезпечує можливість оперативного надсилання сповіщень довіреним особам через Telegram-бот. Використання алгоритмів OpenCV та YOLO дозволяє реалізувати систему в режимі реального часу та досягти високої точності розпізнавання подій[8]-[9].

1.4 Постановка завдань дипломної роботи

В даному розділі було проведено аналіз проблеми забезпечення безпеки людей похилого віку та розглянуто основні причини виникнення падінь, які є однією з найпоширеніших причин травматизму та смертності серед осіб старшого віку. Встановлено, що своєчасне виявлення падіння та оперативне надання допомоги є важливими чинниками зниження ризику тяжких наслідків для здоров'я.

Було виконано огляд існуючих систем моніторингу та детектування падінь, зокрема носимих, візуальних, середовищних, та гібридних систем. Проведений аналіз показав, що кожен із підходів має власні переваги та недоліки, пов'язані з точністю роботи, зручністю використання, вартістю реалізації та рівнем надійності.

Також було розглянуто основні методи виявлення падінь: порогові методи, методи машинного навчання, методи комп'ютерного зору та гібридні методи. Виконано їх порівняльний аналіз, за результатами якого встановлено, що методи комп'ютерного зору забезпечують високу точність розпізнавання подій без необхідності використання додаткових носимих пристроїв.

На основі проведеного дослідження для реалізації системи моніторингу безпеки людей похилого віку було обрано метод комп'ютерного

зору з використанням OpenCV та YOLO. Обраний підхід дозволяє здійснювати автоматичне виявлення падінь у режимі реального часу та забезпечує можливість оперативного сповіщення довірених осіб про виникнення небезпечних ситуацій.

Мета роботи – підвищення точності детектування падінь на основі створення програмного забезпечення для запобігання несподіваним ситуаціям, що пов'язані зі здоров'ям людей похилого віку.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати проблему забезпечення безпеки людей похилого віку;
- дослідити існуючі системи моніторингу та методи детектування падінь;
- обґрунтувати вибір технології для реалізації системи;
- спроектувати архітектуру програмного забезпечення;
- реалізувати програмну систему детектування падінь;
- провести тестування та оцінку ефективності розробленого рішення.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Проєктування програмного забезпечення є важливим етапом розробки системи моніторингу безпеки людей похилого віку з функцією детектування падінь. Саме на цьому етапі визначаються основні вимоги до системи, її структура, логіка роботи та взаємодія між окремими компонентами.

Оскільки розроблюване рішення реалізується у вигляді Telegram-бота, особлива увага приділяється зручності користувача, швидкості реагування на небезпечні ситуації та надійності передачі повідомлень. Система повинна забезпечувати автоматичне виявлення падінь, оперативне сповіщення довірених осіб та можливість підтвердження безпечного стану користувача.

У даному розділі розглядаються функціональні та не функціональні вимоги до системи, загальна архітектура програмного забезпечення, структура бази даних, вибір технологій та інструментів розробки, а також особливості проєктування інтерфейсу користувача. Детальне опрацювання цих аспектів дозволяє створити ефективну, надійну та зручну систему моніторингу, що відповідає сучасним вимогам до програмних рішень у сфері HealthTech.

2.1 Вимоги до системи моніторингу

Проєктована система моніторингу безпеки людей похилого віку реалізується у вигляді Telegram-бота з функцією детектування падінь та автоматичного надсилання тривожних повідомлень. Основною метою системи є забезпечення оперативного реагування на небезпечні ситуації, зокрема падінь користувача а також підвищення рівня безпеки та самостійності людей похилого віку[1].

Для ефективного функціонування системи повинна відповідати функціональним і нефункціональним вимогам [4].

2.1.1 Функціональні вимоги

Функціональні вимоги визначають основні можливості системи та її поведінку під час взаємодії з користувачем. Система повинна забезпечувати реєстрацію користувача з можливістю введення основних персональних даних, таких як ім'я, пошта та контактні дані довіреної особи (родича, опікуна або медичного працівника). Це необхідно для персоналізації роботи системи та оперативної інформування відповідальних осіб у разі надзвичайної ситуації. Telegram-бот повинен отримувати дані з камер відео спостереження. У разі виявлення критичної ситуації система повинна автоматично формувати сигнал тривоги [10].

Однією з ключових функцій є надсилання повідомлення про можливе падіння через Telegram. Повідомлення повинно містити інформацію про час події та тип тривоги. Це дозволяє швидко оцінити ситуацію та надати необхідну допомогу [1].

Система також повинна надавати користувачу можливість підтвердити свій безпечний стан. Наприклад після отримання повідомлення про падіння користувач може натиснути кнопку “Я в безпеці”, що скасовує подальше надсилання екстрених сповіщень. Додатково система повинна зберігати історію тривожних повідомлень, що дозволяє переглядати попередні події, аналізувати частоту виникнення небезпечних ситуацій та покращувати якість моніторингу.

Основні функціональні можливості системи:

- реєстрація користувача;
- збереження контактних даних довірених осіб;
- отримання даних з камер відео спостереження;
- автоматичне визначення падіння;
- надсилання тривожних повідомлень;
- підтвердження безпечного стану;
- перегляд історії сповіщень.

2.1.2 Нефункціональні вимоги

Нефункціональні вимоги визначають характеристики якості програмного забезпечення, які забезпечують зручність, надійність та ефективність роботи системи.

Однією з найважливіших вимог є швидкодія системи. У разі виявлення падіння повідомлення повинно надсилатися максимально швидко, оскільки затримка може призвести до серйозних наслідків для здоров'я користувача [1].

Система повинна працювати стабільно та безперервно, забезпечуючи цілодобовий моніторинг стану користувача. Надійність роботи є критично важливою, оскільки помилка або збої можуть знизити ефективність реагування на небезпечні ситуації [4].

Інтерфейс Telegram-бота має бути простим, зрозумілим і доступним для людей похилого віку, які можуть мати обмежений досвід користування цифровими технологіями. Для цього необхідно використовувати мінімальну кількість команд, інтуїтивно зрозумілі кнопки та чіткі повідомлення [10].

Важливою вимогою є захист персональних даних користувачів. Контактна інформація та історія сповіщень повинні зберігатися безпечно та бути доступними лише авторизованим особам [11].

Система повинна мати можливість подальшого розширення функціоналу, наприклад інтеграції з медичними сервісами або системами екстреного виклику.

Основні нефункціональні вимоги:

- висока швидкість реагування;
- стабільність і надійність роботи;
- простота та зручність використання;
- захист персональних даних;
- цілодобова доступність;
- можливість модернізації.

Для більш наочного та зручного представлення функціональних та не функціональних вимог до системи моніторингу я вирішив сформувати таблицю 2.1, у якій наведено основні характеристики програмного забезпечення, їх опис та рівень пріоритетності для реалізації.

Таблиця 2.1 – Функціональні та не функціональні вимоги до системи моніторингу [4]-[11]

Категорія вимоги	Назва вимоги	Опис вимоги	Пріоритет
1	2	3	4
Функціональна	Реєстрація користувача	Система повинна забезпечувати введення персональних даних користувача та контактів довіреної особи	Високий
Функціональна	Детектування падіння	Автоматичне визначення падіння на основі даних з камери відео спостереження	Високий
Функціональна	Надсилання сповіщення	Надсилання тривожного повідомлення через Telegram у разі небезпечної ситуації	Високий
Функціональна	Історія повідомлень	Збереження попередніх тривожних сповіщень для подальшого перегляду	Середній
Нефункціональна	Швидкодія	Повідомлення повинно надсилатися без значної затримки	Високий

Кінець таблиці 2.1

1	2	3	4
Нефункціональна	Надійність	Система повинна працювати стабільно та безперервно	Високий
Нефункціональна	Простота прискорення	Інтерфейс Telegram – бота має бути зрозумілим для людей похилого віку	Високий
Нефункціональна	Захист даних	Персональні дані користувача повинні бути захищені	Високий
Нефункціональна	Модернізація	Можливість подальшого розширення функціоналу системи	Середній

Таким чином, сформульовані вимоги визначають основні характеристики системи моніторингу та створюють основу для подальшого проєктування архітектури програмного забезпечення. Реалізація зазначених вимог дозволить створити ефективну систему підтримки безпеки людей похилого віку з використанням Telegram-бота як зручного інструменту взаємодії.

2.2 Загальна архітектура системи

Загальна архітектура системи визначає структуру програмного забезпечення, основні компоненти та їх взаємодію. Системи реалізується у вигляді клієнт - серверного рішення з використанням Telegram-бота як інтерфейсу взаємодії з користувачем. Такий підхід дозволяє забезпечити оперативний обмін повідомленнями, високу доступність та зручність використання мобільних технологій у системах моніторингу [10].

Архітектура системи побудована за модульним принципом, що забезпечує гнучкість, масштабованість та можливість подальшого розширення функціоналу [12]. Основними компонентами системи є: відеокамера, модуль комп'ютерного зору, серверна частина, база даних, Telegram-бот та довірена особа.

2.2.1 Основні компоненти системи

2.2.1.1 Користувач (людина похилого віку)

Користувач є центральним елементом системи. Взаємодія із системою здійснюється через Telegram-бота, за допомогою якого користувача отримує повідомлення, може підтверджувати свій стан або самостійно викликати допомогу [10].

2.2.1.2 Модуль відеокамери

Відеокамера є джерелом даних для системи. Вона забезпечує безперервне відеоспостереження за користувачем та передає відеопотік до модуля комп'ютерного зору для подальшого аналізу.

Використання камери дозволяє здійснювати моніторинг без необхідності використання носимих пристроїв, що підвищує комфорт користувача.

2.2.1.3 Модуль комп'ютерного зору

Модуль комп'ютерного зору відповідає за аналіз відеопотоку, отриманого з камери спостереження. Для виявлення людини та визначення її положення використовуються алгоритми комп'ютерного зору та машинного навчання.

Основними функціями модуля є:

- отримання кадрів із відеокамери;
- виявлення людини в кадрі;
- аналіз положення тіла людини;
- визначення факту можливого падіння;
- передача інформації на сервер для подальшої обробки.

Для реалізації детектування об'єктів використовується алгоритм YOLO, який забезпечує швидке та точне виявлення людини в режимі реального часу [4]-[12].

2.2.1.4 Серверна частина

Серверна частина є основним обчислювальним модулем системи. Вона реалізується за допомогою Python та виконує наступні функції:

- прийом та обробка відеопотоку;
- аналіз події;
- формування тривожних повідомлень;
- взаємодія з Telegram Bot API;
- робота з базою даних.

Використання Python є доцільним завдяки великій кількості бібліотек для роботи з API, асинхронною обробкою подій та підтримкою роботи з базами даних [12].

2.2.1.5 База даних

Для збереження інформації використовується база даних, яка містить:

- персональні дані користувача;
- контакти довірених осіб;
- історія тривожних подій;
- ID камер;
- IDTelegram.

Для реалізації використовується PostgreSQL, що дозволяє забезпечити швидкий доступ до даних та їх надійне збереження [13].

2.2.1.6 Telegram-бот

Для обґрунтування засобів взаємодії користувача із системою моніторингу було проведено порівняльний аналіз засобів взаємодії, який подано в таблиці 2.2.

Таблиця 2.2 – Засоби взаємодії користувача[10]

Критерій	Telegram- бот	Веб - додаток	Мобільний додаток
Вартість розробки	Низька	Середня	Висока
Швидкість розробки	Висока	Середня	Низька
Миттєві сповіщення	Так	Обмежено	Так
Необхідність встановлення	Ні (лише Telegram)	Ні	Так
Перегляд відео з камер	Так	Так	Так
Зручність користування	Висока	Середня	Висока
Підтримка команд користувача	Так	Так	Так
Доступність на різних пристроях	Висока	Висока	Середня

Як видно з таблиці 2.2, Telegram – бот поєднує переваги мобільних та веб – додатків, забезпечуючи швидке отримання сповіщень, підтримку інтерактивних команд та можливість перегляду даних без необхідності

розробки окремого клієнтського програмного забезпечення. Крім того, більшість користувачів уже мають встановлений месенджер Telegram, що спрощує впровадження системи та зменшує витрати на її розробку. Тому для реалізації інтерфейсу взаємодії користувача із системою було обрано Telegram – бот. Робота бота здійснюється через TelegramBotAPI [10].

2.2.1.7 Довірена особа

Довірена особа (родич або опікун) отримує тривожні повідомлення у разі відсутності реакції користувача та може оперативно відреагувати на небезпечну ситуацію.

2.2.2 Принцип роботи системи

Робота системи відбувається за наступним алгоритмом.

Крок 1. Камера здійснює відео спостереження.

Крок 2. Модуль комп'ютерного зору аналізує кадри.

Крок 3. Алгоритм YOLO визначає людину та її положення.

Крок 4. У разі виявлення падіння система формує сигнал тривоги.

Крок 5. Telegram-бот надсилає повідомлення довірений особі.

Крок 6. Уся інформація зберігається в базі даних [8]-[13].

2.2.3 Переваги обраної архітектури

Запропонована архітектура має такі переваги:

- модульність системи;
- висока швидкість обробки повідомлень;
- простота впровадження;
- відсутність необхідності створення окремого мобільного застосунку [10].

Використання Telegram як клієнтського інтерфейсу дозволяє значно зменшити витрати на розробку та забезпечити швидкий доступ користувачів до функціоналу системи.

2.3 Проєктування структури даних та бази даних

Одним із ключових етапів розробки є проєктування структури даних та бази даних, База даних використовується для збереження інформації про користувачів, контактні дані довірених осіб, історію тривожних подій та системні повідомлення. Правильно спроектована структура даних забезпечує швидкий доступ до інформації та надійність збереження системи [12].

Для реалізації Telegram-бота обрано реляційну модель бази даних, оскільки вона забезпечує структуроване зберігання інформації, підтримку зв'язків між таблицями та високу надійність обробки даних [13]. Як систему керування базами даних доцільно використовувати PostgreSQL, оскільки вона забезпечує високу продуктивність, масштабованість та підтримку складних запитів.

2.3.1 Основні сутності бази даних

У процесі проєктування було визначено три основні сутності системи: таблиця Users, таблиця Guardians та таблиця Alerts. Дана структура забезпечує збереження інформації про користувачів, опікунів та зафіксовані тривожні події.

Таблиця 2.3 призначена для збереження основної інформації про користувача системи. Ця таблиця дозволяє ідентифікувати користувача та забезпечувати персоналізовану взаємодію через Telegram [13].

Таблиця 2.4 призначена для збереження основної інформації про опікуна системи.

Таблиця 2.3 – Users

Поле	Тип даних	Опис
id	SERIAL	Унікальний ідентифікатор користувача
login	VARCHAR(100)	Логін користувача
password	VARCHAR(255)	Пароль користувача
name	VARCHAR(150)	Ім'я користувача
birth_date	DATE	Дата народження
email	VARCHAR(150)	Адреса електронної пошти
guardian_phone	VARCHAR(20)	Номер телефону опікуна
camera_type	VARCHAR(20)	Тип камери для моніторингу
camera_source	VARCHAR(255)	Джерело відео потоку або камери
tg_id	BIGINT	Ідентифікатор користувача в Telegram

Таблиця 2.4 – Guardians

Поле	Тип даних	Опис
id	SERIAL	Унікальний ідентифікатор опікуна
login	VARCHAR(100)	Логін опікуна
password	VARCHAR(255)	Пароль опікуна
name	VARCHAR(150)	Ім'я опікуна
email	VARCHAR(150)	Адреса електронної пошти
phone	VARCHAR(20)	Номер телефону
tg_id	BIGINT	Ідентифікатор користувача в Telegram

Ця таблиця дозволяє ідентифікувати опікуна та забезпечувати персоналізовану взаємодію через Telegram [13].

Таблиця 2.5 використовується для збереження інформації про тривожні події. Кожне повідомлення про падіння або іншу небезпечну ситуацію реєструється у цій таблиці для подальшого аналізу [10].

Таблиця 2.5 – Alerts

Поле	Тип даних	Опис
id	SERIAL	Унікальний ідентифікатор тривоги
user_id	INTEGER	Ідентифікатор користувача
guardian_id	INTEGER	Ідентифікатор опікуна
detected_at	TIMESTAMP	Дата та час виявлення падіння
responded	BOOLEAN	Ознака отримання відповіді на сповіщення
response	VARCHAR(10)	Відповідь на сповіщення

2.3.2 Зв'язки між таблицями

Між таблицями встановлені наступні зв'язки:

- один користувач може бути пов'язаний з одним або декількома записами про тривожні події (Users → Alerts);
- один опікун може отримувати сповіщення про одну або декілька тривожних подій (Guardians → Alerts);
- кожна тривожна подія пов'язана з одним користувачем та одним опікуном (Alerts → Users, Alerts → Guardians).

Ці зв'язки реалізуються за допомогою зовнішніх ключів, що забезпечує цілісність даних та коректну взаємодію між сутностями системи [13].

2.3.3 Логічна модель бази даних

Структуру взаємодії таблиць наведено на рис. 2.1. Запропонована логічна модель бази даних дозволяє зберігати інформацію про користувачів, опікунів та тривожні події.

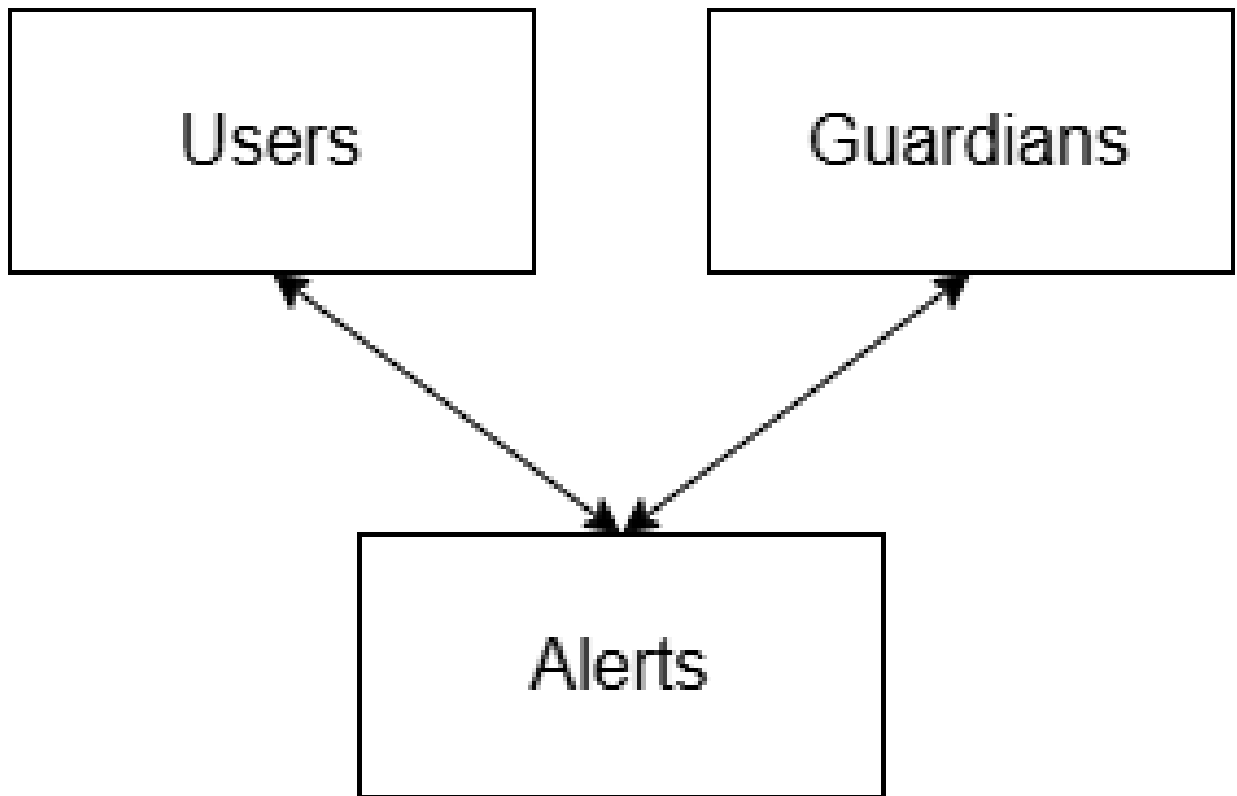


Рисунок 2.1 – Структура взаємодії таблиць [13]

Така структура забезпечує можливість отримання історії детекцій для кожного користувача, аналізу зафіксованих падінь, а також формування звітів про роботу системи моніторингу.

2.3.4 Переваги обраної структури

Розроблена структура бази даних має наступні переваги:

- простота реалізації;
- висока швидкість пошуку інформації;
- підтримки історії подій;
- зручність інтеграції з Telegram-ботом.

Таким чином спроектована база даних забезпечує ефективне збереження та обробку інформації в системі моніторингу, що є необхідною умовою для стабільності роботи програмного забезпечення [13].

2.4 Вибір технологій та інструментів розробки

Одним із важливих етапів проєктування програмного забезпечення є вибір технологій та інструментів розробки. Від обраних програмних засобів залежать продуктивність системи, точність виявлення подій та зручність подальшого супроводу програмного продукту. Для реалізації системи моніторингу було обрано сучасний набір технологій, який забезпечує обробку відеопотоку в режимі реального часу, взаємодію з користувачами через Telegram та надійне збереження інформації [8]-[14].

2.4.1 Мова програмування

У таблиці 2.6 наведено порівняння мов програмування.

Таблиця 2.6 – Порівняння мов програмування [14]-[18]

Критерій	Python	C++	Java	C#
1	2	3	4	5
Простота синтаксису	Висока	Низька	Середня	Середня
Швидкість розробки	Висока	Низька	Середня	Середня
Продуктивність	Середня	Висока	Висока	Висока
Засоби комп'ютерного зору	Дуже багато	Багато	Достатньо	Достатньо
Робота з Telegram API	Дуже проста	Складна	Середня	Середня
Засоби комп'ютерного зору	Дуже багато	Багато	Достатньо	Достатньо

Кінець таблиці 2.6

1	2	3	4	5
Робота з Telegram API	Дуже проста	Складна	Середня	Середня
Асинхронне програмування	Підтримується	Підтримується	Підтримується	Підтримується
Робота з базами даних	Зручна	Складніша	Зручна	Зручна
Швидкість виконання	Середня	Дуже висока	Висока	Висока
Придатність для прототипування	Дуже висока	Низька	Середня	Середня

Основною мовою програмування для реалізації системи було обрано Python. Дана мова широко використовується у сфері веброзробки, комп'ютерного зору, машинного навчання та автоматизації.

Завдяки цим перевагам Python добре підходить для створення систем реального часу, до яких належить система моніторингу [14].

2.4.2 Бібліотека для обробки зображень та відео

В таблиці 2.7 наведено порівняння бібліотек для обробки зображень та відео.

Для роботи з відеопотоком було вирішено використовувати бібліотеку OpenCV. Вона надає широкий набір інструментів для обробки зображень та відео.

Використання OpenCV дозволяє забезпечити швидку обробку відеоданих та інтеграцію з алгоритмами машинного навчання [8].

Таблиця 2.7 – Порівняння бібліотек[8]-[21]

Критерії	OpenCV	Pillow(PIL)	Scikit-image	ImageIO
Робота з відео в реальному часі	Так	Ні	Обмежено	Обмежено
Захоплення відео з камери	Так	Ні	Ні	Так
Обробка зображень	Так	Так	Так	Частково
Комп'ютерний зір	Так	Ні	Частково	Ні
Інтеграція з моделями ШІ	Висока	Низька	Середня	Низька
Продуктивність	Висока	Середня	Середня	Середня
Підтримка відеопотоків	Так	Ні	Ні	Частково
Зручність для системи моніторингу	Висока	Низька	Середня	Низька

2.4.3 Порівняння алгоритмів детекції об'єктів

У таблиці 2.8 наведено порівняння алгоритмів детекції об'єктів.

Як видно з таблиці 2.8, алгоритм YOLO забезпечує найкраще співвідношення між швидкістю роботи та точністю детекції об'єктів. На відміну від FasterR-CNN, який характеризується високою точністю, але значними обчислювальними витратами, YOLO здатний виконувати обробку відеопотоку в режимі реального часу.

Таблиця 2.8 - Порівняння алгоритмів детекції об'єктів [9]-[24].

Критерій	YOLO	Faster R-CNN	SSD	Haar Cascde
Швидкість роботи	Дуже висока	Низька	Висока	Дуже висока
Робота в реальному часі	Так	Ні	Так	Так
Точність детекції	Висока	Дуже висока	Висока	Низька
Виявлення кількох об'єктів	Так	Так	Так	Обмежено
Стійкість до змін освітлення	Висока	Висока	Середня	Низька
Використання нейронних мереж	Так	Так	Так	Ні
Обчислювальні вимоги	Середня	Висока	Середня	Низька
Придатність для систем моніторингу в реальному часі	Дуже висока	Середня	Висока	Низька

У порівнянні з SSD алгоритм YOLO демонструє вищу точність виявлення об'єктів, а порівняно з HaarCascade використовує сучасні нейронні мережі та є більш стійким до змін умов зйомки. Тому для реалізації системи виявлення падінь людини було обрано алгоритм YOLO [9]-[24].

2.4.4 Фреймворк для Telegram - бота

У таблиці 2.9 наведено порівняння бібліотек для створення Telegram - бота.

Таблиця 2.9 - Порівняння бібліотек для створення Telegram– бота [12]-[27]

Критерій	pyTelegram BotAPI	python-telegram-bot	aiogram	Node.js (Telegraf)
Простота використання	Висока	Середня	Середня	Середня
Асинхронна підтримка	Обмежена	Частково	Повна	Повна
Швидкість розробки	Висока	Середня	Висока	Висока
Робота з командами та повідомленнями	Зручна	Зручна	Дуже зручна	Зручна
Підтримка inline кнопок та меню	Так	Так	Так	Так
Продуктивність	Середня	Середня	Висока	Висока
Складність проєкту	Для простих і середніх ботів	Для середніх проєктів	Для складних і високонавантажених ботів	Для веб - екосистеми
Підтримка спільнотою	Висока	Висока	Висока	Висока

Для реалізації Telegram-бота вирішено використовувати бібліотеку `TelegramBotAPI`, яка забезпечує зручну та легку взаємодію з `TelegramBotAPI` та дозволяє швидко створювати ботів різної складності.

Використання `pyTelegramBotAPI` дозволяє реалізувати надійний механізм сповіщення довірених осіб про виявлені небезпечні ситуації, забезпечуючи оперативну передачу повідомлень через Telegram [12]-[27].

2.4.5 Система керування базами даних

У таблиці 2.10 наведено порівняння різних СУБД.

Таблиця 2.10 - Порівняння різних СУБД [13]-[31]

Критерій	PostgreSQL	MySQL	SQLite	MongoDB
Тип СУБД	Реляційна (клієнт – серверна)	Реляційна	Вбудована	NoSQL (документна)
Продуктивність	Висока	Висока	Висока (для малих даних)	Висока
Масштабованість	Дуже висока	Висока	Низька	Дуже висока
Підтримка складних запитів	Дуже розвинена	Добра	Обмежена	Обмежена (інший підхід)
Транзакція (ACID)	Повна підтримка	Повна підтримка	Повна підтримка	Часткова
Робота з великими даними	Дуже ефективна	Ефективна	Обмежена	Дуже ефективна
Гнучкість структури даних	Середня	Середня	Низька	Дуже висока

Кінець таблиці 2.10

1	2	3	4	5
Складність налаштування	Середня	Середня	Дуже низька	Середня
Підтримка багато користувачів	Повна	Повна	Обмежена	Повна
Використання у системах моніторингу	Дуже підходить	Підходить	Для простих систем	Підходить для аналітики

Як видно з таблиці 2.10, PostgreSQL є однією з найбільш збалансованих та потужних реляційних СУБД, що забезпечує високу продуктивність, повну підтримку транзакцій та розширені можливості роботи з даними. У порівнянні з MySQL вона має більш розвинені функції для складних запитів, а у порівнянні з SQLite — значно вищу масштабованість та підтримку багатокористувацького режиму. MongoDB, у свою чергу, забезпечує високу гнучкість, але має інший (NoSQL) підхід до зберігання даних. Тому PostgreSQL є оптимальним вибором для серверної частини систем моніторингу[13]-[31].

2.4.6 Середовище розробки

На таблиці 2.11 ми можемо побачити порівняння різних середовищ розробки програмного забезпечення.

Для створення системи моніторингу було вирішено використовувати VisualStudioCode. VisualStudioCode є легким та універсальним середовищем розробки, яке забезпечує достатній рівень функціональності для створення системи моніторингу. У порівнянні з PyCharm, воно має менше вбудованих інструментів для Python, однак є значно легшим та швидшим.

Таблиця 2.11 - Порівняння різних середовищ розробки [15]-[34]

Критерій	Visual Studio Code	PyCharm	Visual Studio	Jupyter Notebook
Підтримка Python	Висока (через розширення)	Дуже висока	Середня	Висока
Швидкість запуску	Висока	Середня	Середня	Висока
Використання ресурсів	Низьке	Висока	Висока	Середня
Інтелектуальна підказка коду	Висока	Дуже висока	Висока	Середня
Налагодження (debugging)	Добре	Дуже добре	Дуже добре	Обмежено
Підтримка розширень	Дуже велика	Середня	Середня	Низьке
Зручність для невеликих проєктів	Дуже висока	Висока	Середня	Висока
Зручність для ML/CV задач	Висока	Дуже висока	Середня	Дуже висока

VisualStudio орієнтована переважно на розробку під .NET, а JupyterNotebook більше підходить для досліджень та експериментів. Таким чином, VisualStudioCode є оптимальним вибором для реалізації програмної частини системи[15]-[34].

2.4.7 Обґрунтування вибору технологій

Обраний набір технологій забезпечує:

- обробку відео потоку в режимі реального часу;

- автоматичне виявлення падінь;
- швидке надсилання сповіщень;
- надійне збереження інформації.

Таким чином, використання Python, OpenCV, YOLO, pyTelegramBotAPI та SQLite є оптимальним рішенням для реалізації системи моніторингу безпеки людей похилого віку з функцією автоматичного виявлення падінь та надсилання повідомлень через Telegram-бот [8]-[13].

2.5 Проєктування інтерфейсу користувача

Інтерфейс користувача є важливою складовою системи моніторингу, оскільки забезпечує взаємодію між користувачем, довіреною особою та програмним забезпеченням. Під час проєктування особлива увага приділялася простоті використання, доступності функцій та оперативному отриманню повідомлень про небезпечні ситуації.

Для реалізації користувацького інтерфейсу було обрано Telegram-бот, який дозволяє здійснювати взаємодію із системою без необхідності встановлення окремого мобільного застосунку. Використання Telegram забезпечує швидку доставку повідомлень, підтримку мобільних пристроїв та зручність використання [10].

2.5.1 Вимоги до інтерфейсу

Під час проєктування інтерфейсу було визначено наступні вимоги:

- простота використання;
- зрозумілий інтерфейс;
- швидкий доступ до основних функцій;
- оперативне отримання сповіщень;
- підтримка мобільних пристроїв;
- мінімальна кількість дій для виконання основних операцій.

Інтерфейс повинен бути зрозумілим як для людей похилого віку, так і для їхніх родичів або опікунів.

2.5.2 Структура інтерфейсу

Основним елементом взаємодії є Telegram-бот. Через нього користувач та довірена особа можуть отримувати повідомлення про стан системи та можливі небезпечні події [10].

Структура інтерфейсу включає:

- головне меню;
- інформаційні повідомлення;
- тривожні повідомлення;
- команди користувача;
- команди опікуна.

2.5.3 Проєкт головного меню

Головне меню Telegram-бота повинно забезпечувати швидкий доступ до основних функцій системи.

Приклад структури меню: Почати роботу; Історія подій; Видалити акаунт; Змінити акаунт.

За допомогою меню користувач або опікун можуть керувати своїми акаунтами, отримувати інформацію про роботу системи та переглядати зафіксовані події [10].

2.5.4 Проєктування повідомлень

Одним із головних завдань інтерфейсу є інформування користувача та довіреної особи про виявлені падіння.

Приклад повідомлення про небезпечну ситуацію: "Увага! Система зафіксувала можливе падіння людини. Перевірте її стан."

Приклад інформаційного повідомлення: "Система моніторингу працює у штатному режимі."

Повідомлення повинні бути короткими, зрозумілими та містити лише необхідну інформацію.

2.5.5 Сценарій взаємодії користувача із системою

Робота користувача із системою передбачає наступний сценарій.

Крок 1. Камера здійснює відео спостереження.

Крок 2. Відеопотік передається до модуля комп'ютерного зору.

Крок 3. Алгоритм аналізує положення людини.

Крок 4. У разі виявлення падіння формується тривожна подія.

Крок 5. Telegram – бот надсилає повідомлення користувачу.

Крок 6. Якщо користувач не реагує - бот надсилає повідомлення довіреній особі.

Крок 7. Інформація про подію записується до бази даних.

Сценарій взаємодії компонентів інтерфейсу зображено на рис.2.2 [8]-[13].

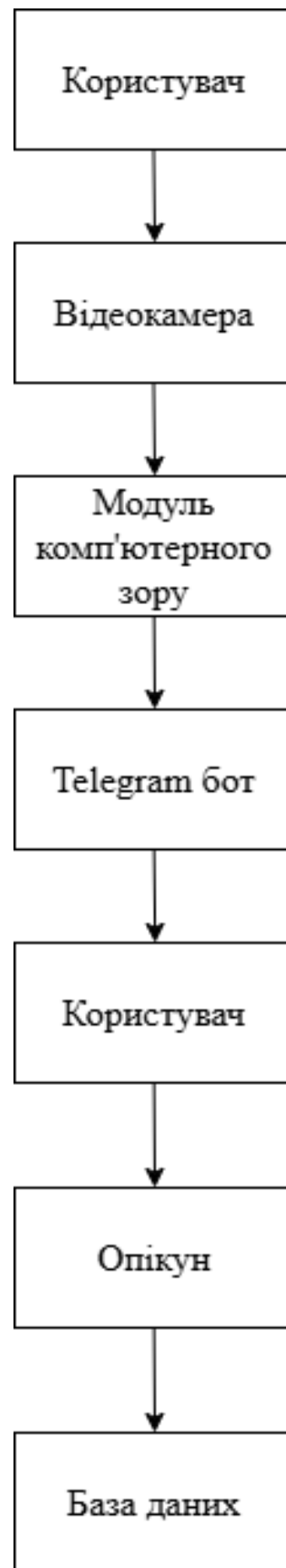


Рисунок 2.2 – Сценарій взаємодії [8]-[13]

2.6 Висновок до розділу 2

У даному розділі було виконано проєктування програмного забезпечення системи моніторингу. Розглянуто основні функціональні та не функціональні вимоги до системи, визначено ключові характеристики її роботи, а також сформовано вимоги до продуктивності, надійності та зручності використання.

Було розроблено загальну архітектуру системи, що базується на модульному підході та включає відеокамеру, модуль комп'ютерного зору, серверну частину, базу даних, Telegram-бота та довірених осіб. Така структура забезпечує гнучкість, масштабованість та можливість подальшого розширення функціоналу системи.

Окрему увагу приділено проєктуванню структури бази даних, яка реалізована на основі реляційної моделі та забезпечує збереження інформації про користувачів, опікунів і тривожні події. Використання PostgreSQL дозволяє ефективно працювати з даними та підтримує багатокористувацький режим.

Також обґрунтовано вибір технологій та інструментів розробки. Показано доцільність використання Python як основної мови програмування, бібліотеки OpenCV для обробки відеопотоку, алгоритму YOLO для детекції об'єктів, а також TelegramBotAPI для організації взаємодії з користувачем. Проведено порівняльний аналіз альтернативних рішень, що підтвердило обґрунтованість вибраних технологій.

Описано інтерфейс користувача на основі Telegram-бота, який забезпечує просту та швидку взаємодію з системою, а також оперативне отримання повідомлень про можливі небезпечні ситуації.

Таким чином, у результаті виконаного проєктування створено повну концептуальну модель програмного забезпечення системи моніторингу, яка відповідає сучасним вимогам до систем безпеки, забезпечує ефективне виявлення падінь та оперативне інформування довірених осіб.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

На даному етапі здійснюється практична реалізація архітектури системи, програмних модулів та механізмів взаємодії між користувачем, модулем аналізу відеопотоку та Telegram-ботом.

Основною метою розробки є створення надійної та зручної системи, здатної автоматично визначати небезпечні ситуації, оперативно надсилати тривожні повідомлення та забезпечувати взаємодію з довіреними особами.

Для реалізації програмного забезпечення використовується мова програмування Python. Обробка відеопотоку та виявлення об'єктів здійснюється із застосуванням бібліотек OpenCV та моделі YOLO. Для реалізації Telegram-бота використовується бібліотека pyTelegramBotAPI. Збереження даних здійснюється у реляційній базі даних PostgreSQL, яка містить інформацію про користувачів, результати відеоаналізу та зафіксовані тривожні події [8]-[13].

У цьому розділі розглядаються особливості реалізації Telegram-бота, механізм детектування падінь, організація роботи з базою даних, а також реалізація основних функцій системи моніторингу. Окрему увагу приділено алгоритмам обробки відеопотоку, формуванню сповіщень та забезпеченню стабільної роботи програмного забезпечення в режимі реального часу.

3.1 Реалізація Telegram – бота для моніторингу безпеки

Для забезпечення взаємодії між користувачами, опікуном та системою моніторингу було реалізовано Telegram-бота. Його основним призначенням є надсилання повідомлень про виявлені небезпечні ситуації та забезпечення взаємодії між користувачем і довіреною особою.

Розробка Telegram-бота здійснювалася мовою програмування Python із використанням бібліотеки pyTelegramBotAPI. Дана бібліотека має можливість інтеграції із зовнішніми сервісами та базами даних, забезпечує

зручну роботу з командами та повідомленнями. Вона дозволяє забезпечити надійну передачу повідомлень про небезпечні ситуації до опікунів[10]-[12].

3.1.1 Структура Telegram – бота

Програмна структура бота побудована за модульним принципом, що забезпечує зручність підтримки та можливість подальшого розширення функціоналу системи.

Основними модулями системи є:

- модуль запуску бота;
- модуль запису відео та інтеграції з YOLO;
- модуль взаємодії з базою даних;
- модуль реєстрації акаунту;
- модуль отримання повідомлень тривоги.

Такий підхід дозволяє розділити функціональність системи на окремі компоненти та спростити процес супроводу програмного забезпечення.

3.1.2 Робота Telegram – бота

Для взаємодії людини із системою реалізовано набір основних команд Telegram – бота [10].

Команда `/start` використовується для запуску бота та реєстрації користувача або опікуна у системі. Після виконання команди з'являється вибір.

На рис. 3.1 представлено приклад реалізації команди `/start`.

Після виконання команди у людини є вибір, за кого вона хоче зайти.

На рис. 3.2 представлено схему роботи бота за користувача.



Рисунок 3.1 – Команда /start [10]

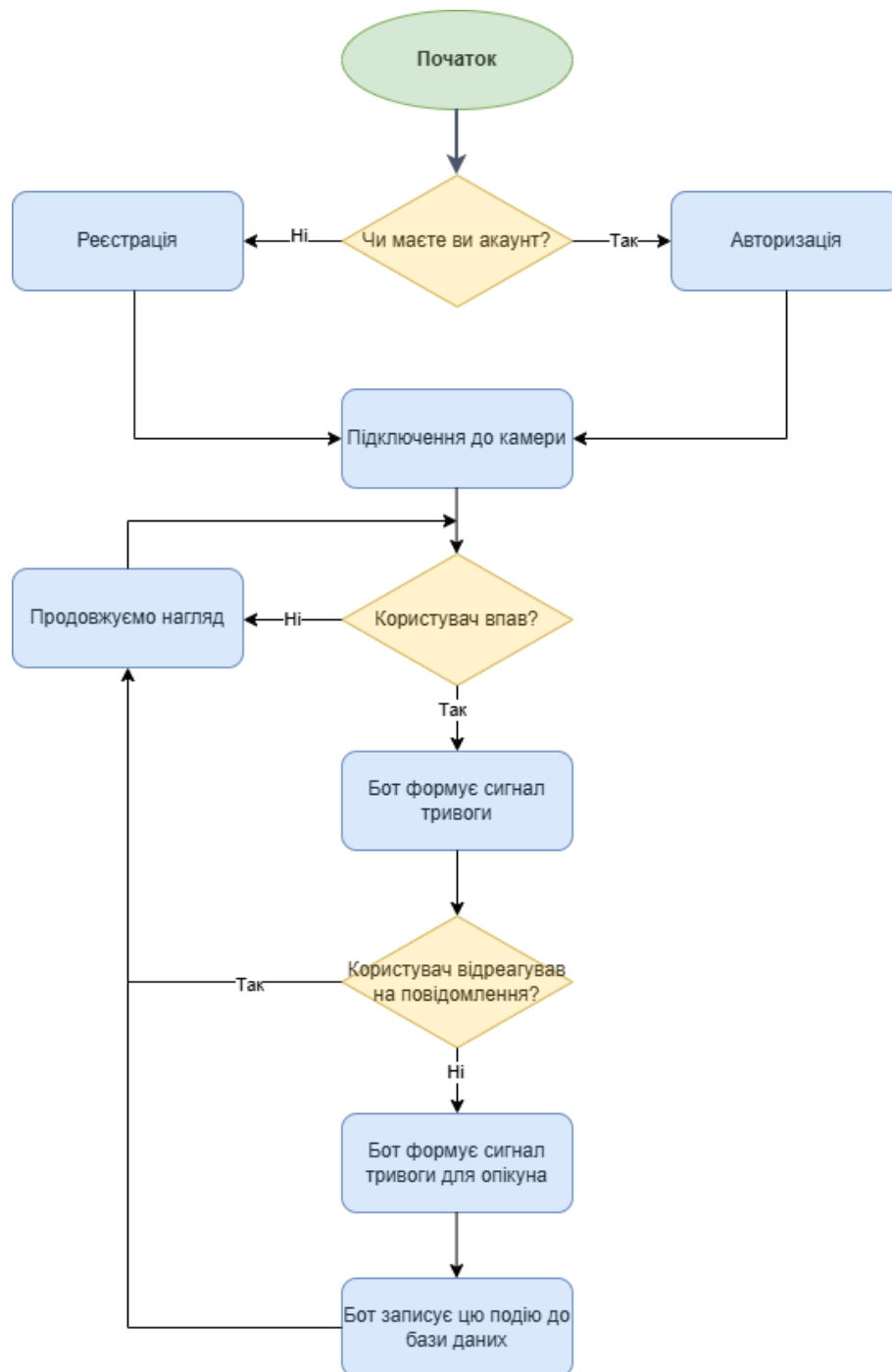


Рисунок 3.2 – Загальна схема роботи бота за користувача [8]-[14]

На рис. 3.3 представлено схему роботи бота за опікуна.

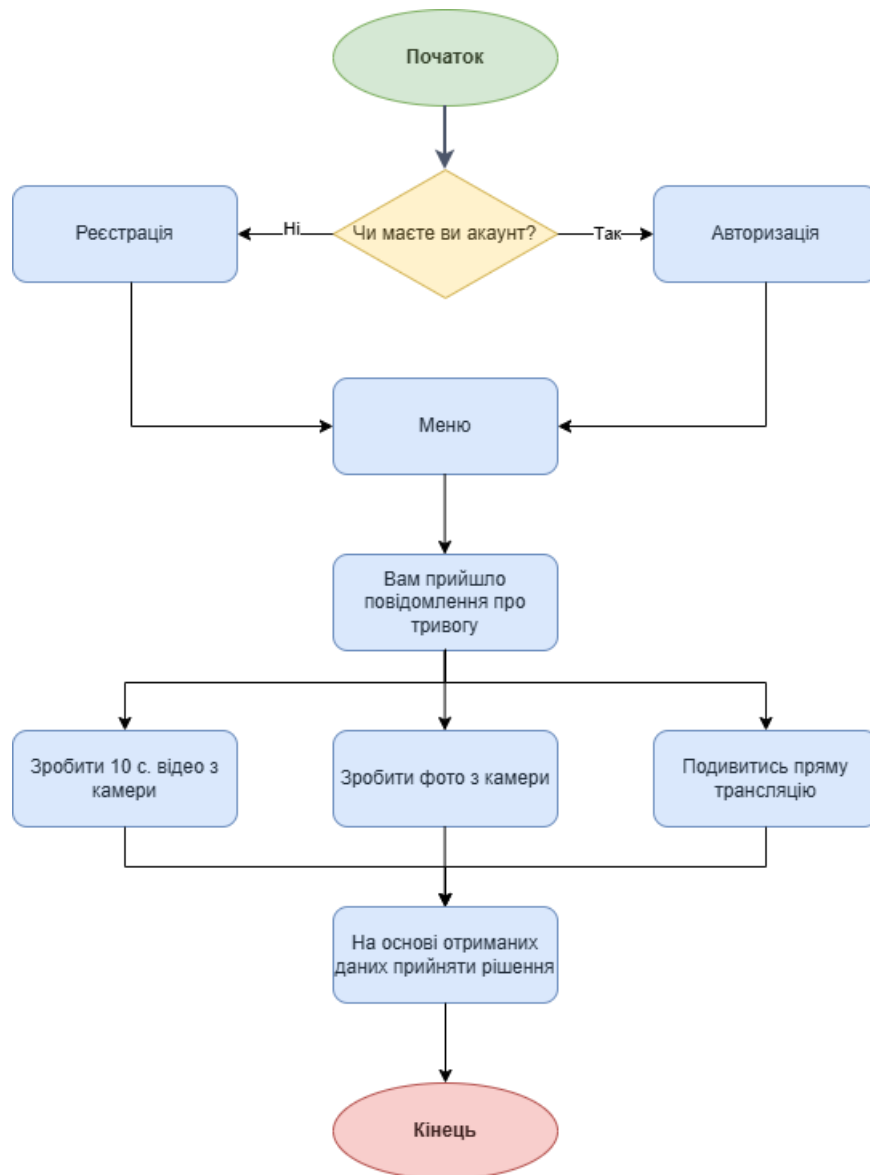


Рисунок 3.3 – Загальна схема роботи бота за опікуна [8]-[14]

На цих схемах показано основні функції та можливі варіанти роботи даного Telegram – бота.

3.1.3 Надсилання тривожного повідомлення

Однією з ключових функцій Telegram-бота є надсилання тривожних повідомлень у разі виявлення падіння користувача. Після отримання сигналу

від модуля комп'ютерного зору система формує повідомлення та надсилає його користувачу з можливістю підтвердження власного стану.

Для цього реалізовано механізм інтерактивного підтвердження, який дозволяє користувачу повідомити, що загрози немає, за допомогою кнопки «Я в порядку». Це забезпечує первинну перевірку достовірності інциденту та зменшує кількість хибних спрацювань.

Після надсилання повідомлення користувачу запускається таймер очікування відповіді. Якщо протягом заданого інтервалу часу (1 хвилина) користувач не підтверджує свій стан, система автоматично ініціює ескалацію тривоги та надсилає повідомлення довіреній особі (опікуну).

У разі підтвердження користувачем свого стану, тригер ескалації скасовується, а інцидент позначається як такий, що не потребує подальшого реагування.

3.2 Реалізація механізму детектування падінь

Механізм детектування падінь є ключовою алгоритмічною складовою системи моніторингу безпеки, що відповідає за аналіз відеопотоку з камери та виявлення аномальних рухів людини. На відміну від сенсорних методів, дана реалізація базується на технологіях комп'ютерного зору, що дозволяє здійснювати аналіз поведінки користувача без використання носимих пристроїв.

Основою алгоритму є обробка послідовності відеокадрів у режимі реального часу, що дозволяє визначати зміни положення тіла людини, характерні для падіння.

3.2.1 Обробка відео потоку

Для захоплення та попередньої обробки відеопотоку використовується бібліотека OpenCV, яка забезпечує отримання кадрів з

камери, їх декодування та підготовку до подальшого аналізу. OpenCV дозволяє виконувати масштабування зображення, фільтрацію шумів та нормалізацію вхідних даних, що підвищує стабільність роботи алгоритму [8].

Для виявлення людини у відеопотоці використовується модель глибокого навчання YOLO, яка належить до класу одноетапних детекторів об'єктів. YOLO забезпечує високу швидкість та точність розпізнавання об'єктів у реальному часі, що є критично важливим для задач системи моніторингу безпеки [9].

Таким чином, OpenCV відповідає за роботу з відеопотоком, тоді як YOLO виконує детекцію людини в кожному кадрі.

3.2.2 Визначення ключових точок тіла

Після виявлення людини виконується аналіз її положення у кадрі. Для цього використовуються методи оцінки пози, які дозволяють визначити координати ключових точок тіла, таких як голова, плечі, таз та кінцівки.

Отримані координати використовуються для подальшого аналізу просторової орієнтації тіла та його зміщення у часі [9].

3.2.3 Алгоритм детектування падіння

На рисунку 3.4 наведено схему алгоритму детектування падіння на основі відеоаналізу.

Алгоритм починається з отримання відеокадру через OpenCV, після чого за допомогою YOLO виконується виявлення людини у кадрі. Далі визначаються ключові точки тіла та обчислюється кут нахилу корпусу [8]-[9].

У разі різкого переходу з вертикального положення у горизонтальне та відсутності подальшого відновлення пози протягом певного часу система формує сигнал про можливе падіння. Після цього ініціюється процедура підтвердження стану користувача через Telegram-бота [10].

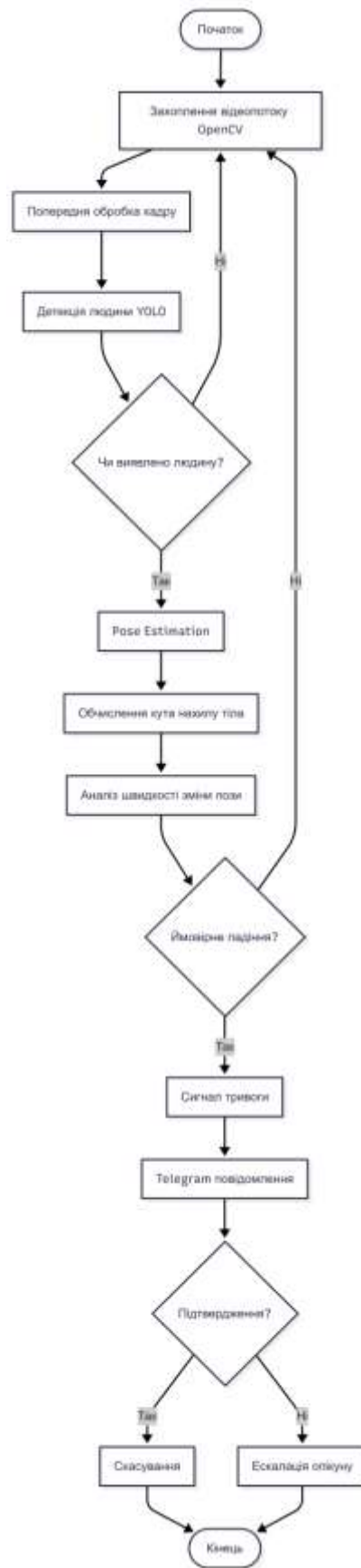


Рисунок 3.4 – Схема алгоритму [8]-[9]

3.3 Реалізація роботи з базою даних

Одним із ключових компонентів системи моніторингу безпеки є база даних, яка забезпечує централізоване збереження інформації про користувачів, опікунів та тривожні події. Реалізація роботи з базою даних дозволяє організувати структуроване зберігання даних та підтримувати історію подій.

У межах розробленого програмного забезпечення база даних використовується для:

- реєстрації користувачів;
- збереження Telegram ID;
- збереження ID камери;
- прив'язки опікуна до користувача;
- зберігання історії падінь;
- забезпечення взаємодії між модулями системи.

Для реалізації бази даних було використано PostgreSQL, оскільки дана система керування базами даних є легкою у використанні, не потребує окремого серверного налаштування та добре інтегрується з мовою програмування Python [13].

3.3.1 Створення структури бази даних

Для реалізації системи було створено декілька основних таблиць:

- Users;
- Guardians;
- Alerts.

Таблиця Users містить інформацію про користувачів. Приклад таблиці на рис. 3.5.

public
users
id serial
login character varying(100)
password character varying(255)
name character varying(150)
birth_date date
email character varying(150)
guardian_phone character varying(20)
camera_type character varying(20)
camera_source character varying(255)
tg_id bigint

Рисунок 3.5 –таблиці Users [13]

Таблиця Guardians містить інформацію про опікунів. Приклад таблиці на рис. 3.6.

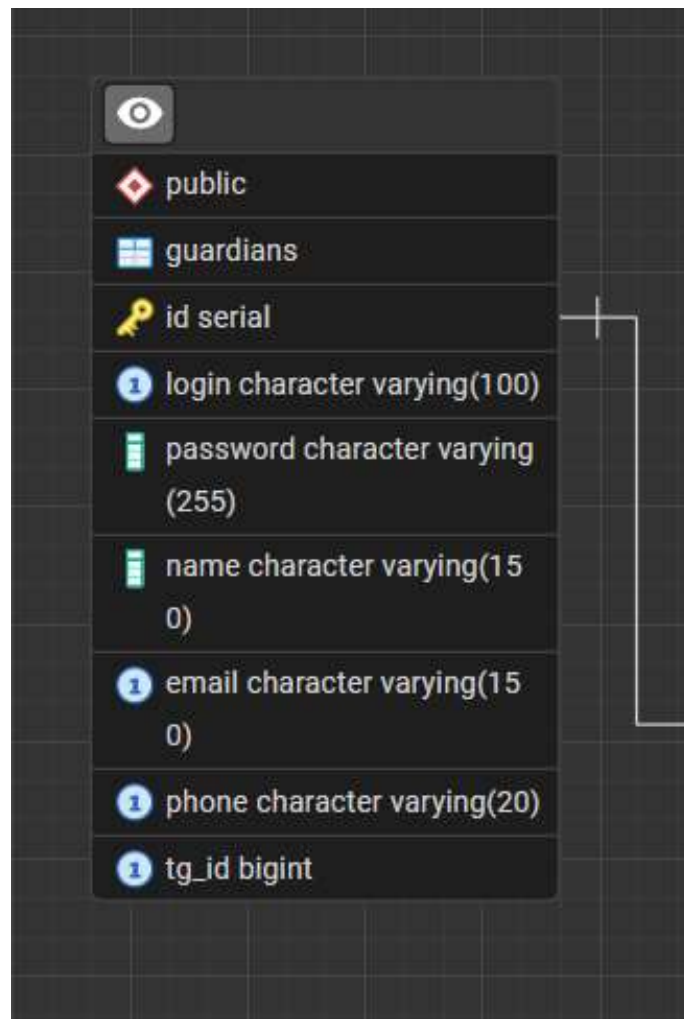


Рисунок 3.6 – Приклад таблиці Guardians [13]

Таблиця Alerts використовується для фіксації тривожних подій, що формуються на основі аналізу відеопотоку.

Приклад таблиці на рис. 3.7.

3.3.2 Підключення до бази даних

Для взаємодії з базою даних у Python використовується бібліотека `psycopg2`, яка забезпечує виконання SQL-запитів та роботу з локальною базою даних без необхідності серверної інфраструктури.

Після встановлення з'єднання система отримує можливість виконувати операції запису та отримання даних, пов'язаних із користувачем, опікуном та тривожними подіями [35].

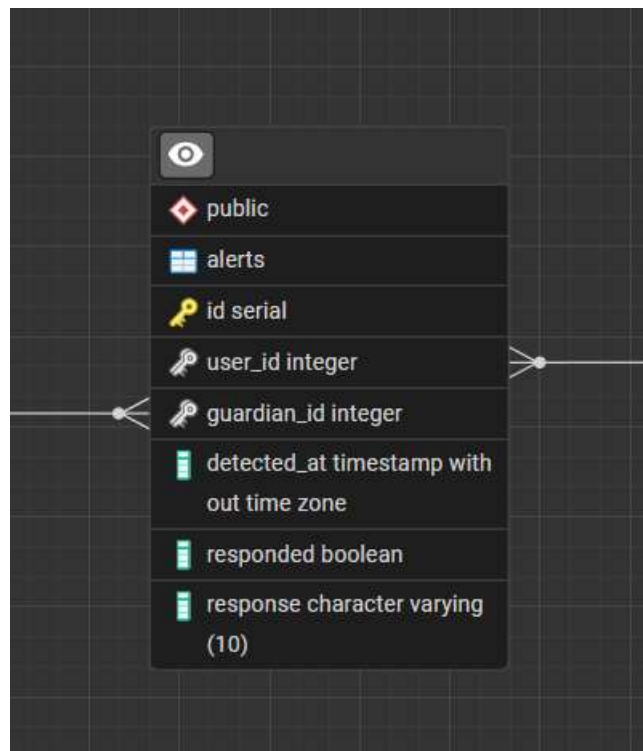


Рисунок 3.7 – Приклад таблиці Alerts[13]

3.4 Інтеграція компонентів системи

Інтеграція компонентів є завершальним етапом розробки системи моніторингу безпеки, який забезпечує взаємодію між усіма програмними модулями: модулем відеоаналізу, базою даних та Telegram-ботом. Основною метою інтеграції є забезпечення безперервного потоку даних від камери до користувача з можливістю фіксації та обробки інцидентів у режимі реального часу.

Розроблена система має модульну архітектуру, що дозволяє незалежно реалізовувати та тестувати кожен компонент, а саме:

- модуль захоплення та обробки відео потоку;
- модуль детекції об'єктів;
- модуль аналізу падінь;
- модуль роботи з базою даних;
- модуль взаємодії з користувачем.

3.4.1 Загальна схема взаємодії компонентів

Робота системи будується за принципом послідовної обробки даних.

Крок 1. Камера передає відеопотік у систему.

Крок 2. OpenCV виконує захоплення та попередню обробку кадрів.

Крок 3. YOLO виконує детекцію людини у кожному кадрі.

Крок 4. Алгоритм аналізує зміну положення тіла та визначає можливе падіння.

Крок 5. У разі виявлення інциденту Telegram-бот надсилає повідомлення користувачу.

Крок 6. У разі відсутності підтвердження виконується ескалація до довіреної особи.

Крок 7. Результат зберігаються у базі даних PostgreSQL [8]-[13].

3.4.2 Взаємодія з базою даних

База даних виконує роль центрального сховища інформації та забезпечує зв'язок між усіма компонентами системи [13].

У процесі роботи:

- події падіння фіксуються у таблиці Alerts;
- інформація про користувачів зберігається у таблиці Users;
- інформація про опікунів зберігається у таблиці Guardians.

Це дозволяє відновлювати повну історію подій та виконувати подальший аналіз ефективності системи[13].

3.4.3 Інтеграція з модулем комп'ютерного зору

Модуль комп'ютерного зору на основі OpenCV та YOLO є джерелом основних даних для системи. OpenCV забезпечує обробку відеопотоку в режимі реального часу, тоді як YOLO виконує детекцію людини у кадрі.

Отримані результати передаються до алгоритму аналізу руху, який визначає зміну положення тіла та формує ознаки можливого падіння[8]-[9].

3.4.4 Інтеграція з Telegram – ботом

Telegram-бот є інтерфейсом взаємодії між системою та користувачем.

У разі виявлення падіння система автоматично:

- надсилає повідомлення користувачу;
- очікує підтвердження стану;
- у разі відсутності відповіді виконує ескалацію до довіреної особи.
- формує запис у базі даних.

Таким чином забезпечується багаторівнева система реагування на інциденти [10].

3.5 Висновок до розділу 3

У даному розділі було виконано практичну реалізацію системи моніторингу безпеки людей похилого віку з функцією автоматичного виявлення падінь. Розроблено програмну архітектуру та реалізовано основні модулі системи, включаючи Telegram-бота, модуль комп'ютерного зору, механізм детектування падінь та роботу з базою даних.

Реалізація Telegram-бота забезпечує взаємодію користувача та довіреної особи із системою, а також надсилання та підтвердження тривожних повідомлень. Модуль відеоаналізу на основі OpenCV та YOLO дозволяє виконувати детекцію людини та аналіз її положення в режимі реального часу.

Також було реалізовано алгоритм визначення падіння на основі зміни положення тіла та часової затримки, що дозволяє зменшити кількість хибних спрацьовувань. База даних PostgreSQL використовується для збереження

інформації про користувачів, опікунів та тривожні події, забезпечуючи цілісність і структурованість даних.

У результаті виконано інтеграцію всіх компонентів системи в єдине програмне рішення, що забезпечує автоматичне виявлення падінь, оперативне сповіщення та подальшу ескалацію інцидентів.

4 ТЕСТУВАННЯ ТА ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

У даному розділі наведена робота за користувача та опікуна для Telegram – бота з системою моніторингу. Проведено тестування головного інтерфейсу, реєстрації користувача та опікуна, точності роботи детекції падіння та надійності доставки сигналів тривоги. Окрім того, виконано експериментальне дослідження по ефективності програми.

4.1 Тестування роботи користувача та опікуна

Даний Telegram – бот призначений для моніторингу безпеки похилих людей з детектуванням падінь. Для коректної роботи бот має мати доступ к камері та мати стабільне підключення до локальної мережі.

Для початку роботи з ботом нам потрібно:

- знайти даного бота в додатку Telegram;
- знайти головний файл `main.py` та запустити його через VisualStudioCode;
- після запуску, відкриваємо новий термінал, і прописуємо `pip install -r requirements.txt` для того, щоб встановити потрібні для функціонування боту бібліотеки;
- як тільки все буде встановлено, включаємо бота прописуючи в терміналі `python main.py`.

Про готовність бота до роботи буде свідчити повідомлення в терміналі.

Після того як бот запустився, з'являються дві кнопки, які відповідають за кого людина хоче зайти. Перша кнопка відповідає за вхід за користувача(людину похилого віку), а друга кнопка відповідає за вхід опікуна(довірена особа).

Перед тим як бот почне виконувати свої функції, користувач і опікун повинні зареєструватися. На рис. 4.1 показано реєстрацію за користувача, а на рис. 4.2 показано реєстрацію за опікуна.

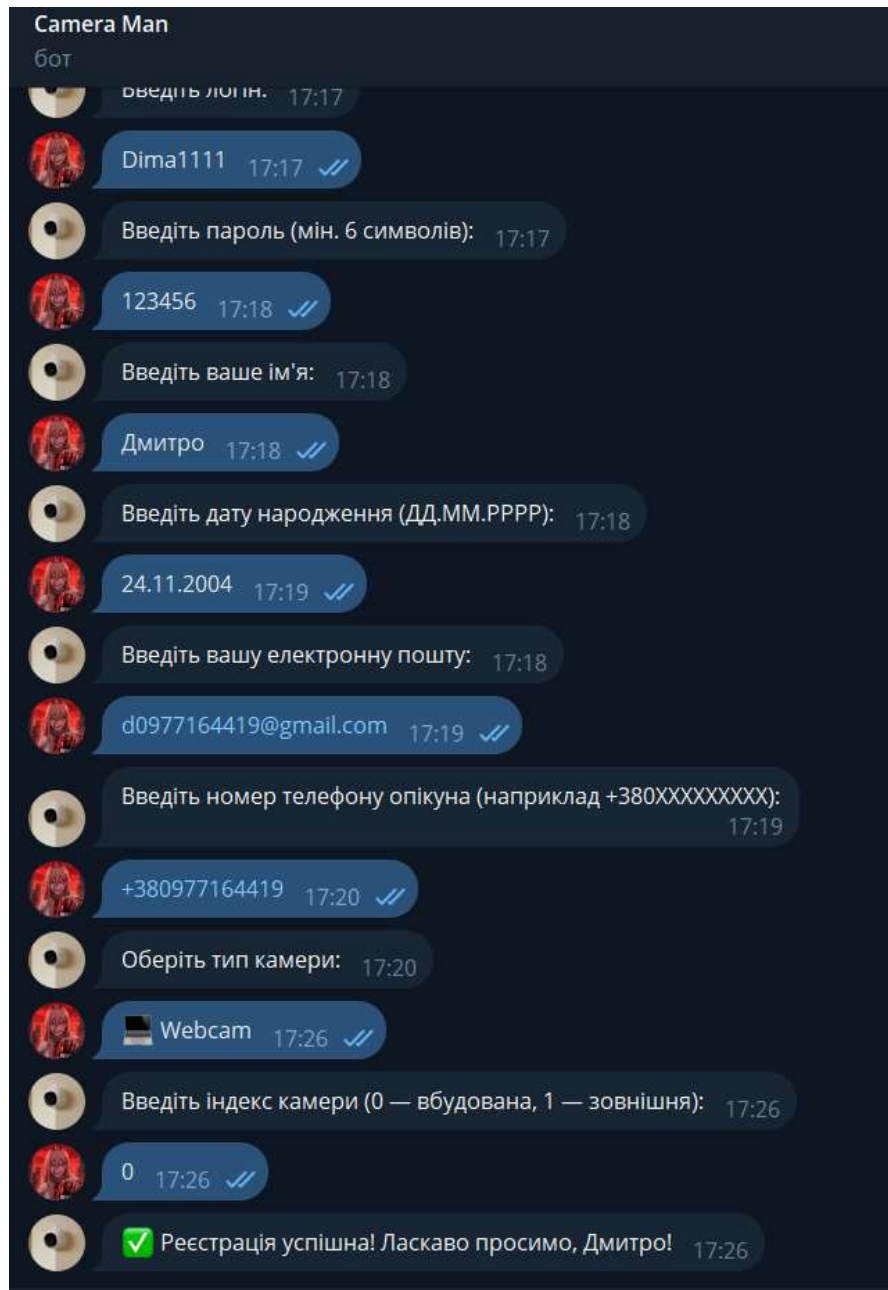


Рисунок 4.1 – Реєстрація за користувача

Окрім реєстрації, є можливість просто зайти в акаунт, якщо він вже є. Після реєстрації, з'являється меню з вибором.



Рисунок 4.2 – Реєстрація за опікуна

На рис. 4.3 показано меню користувача, а на рис. 4.4 показано меню опікуна.

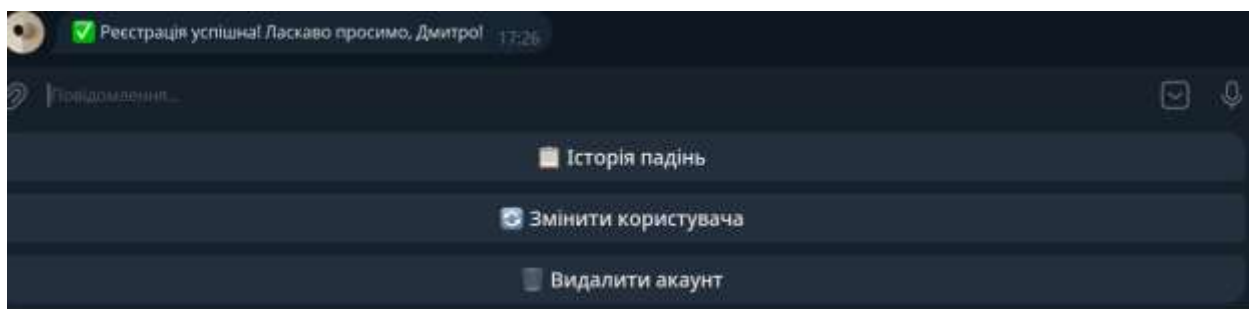


Рисунок 4.3 – Меню користувача

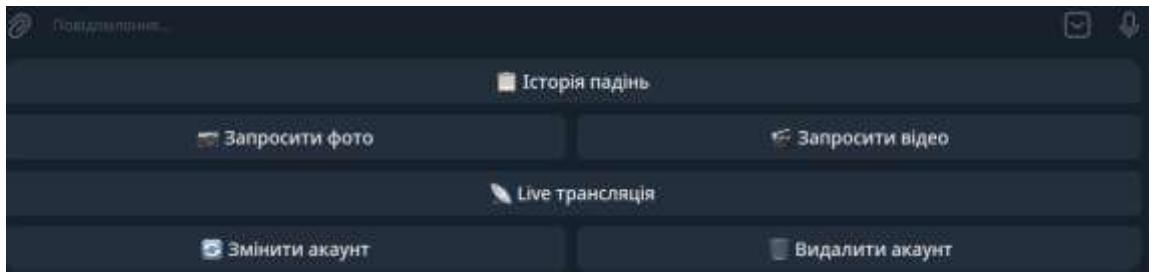


Рисунок 4.4 – Меню опікуна

У меню користувача можна видалити акаунт, змінити користувача, а також подивитись історію падінь, на той випадок, якщо опікун з користувачем захочуть подивитись коли, і скільки раз користувач впав.

У меню опікуна можна видалити акаунт, змінити акаунт, а також запросити відео, фото, або подивитись прямий ефір з камери, на той випадок, якщо опікун захоче перевірити стан користувача. Також у опікуна як і у користувача, є кнопка історія падінь.

Якщо опікун хоче перевірити стан користувача, він може запросити фото, якщо опікун хоче ще більше переконатись в безпеці користувача, він може запросити відео, або увімкнути лайв трансляцію.

На рис. 4.5 показано запит фото с камери, на рис 4.6 показано запит відео.

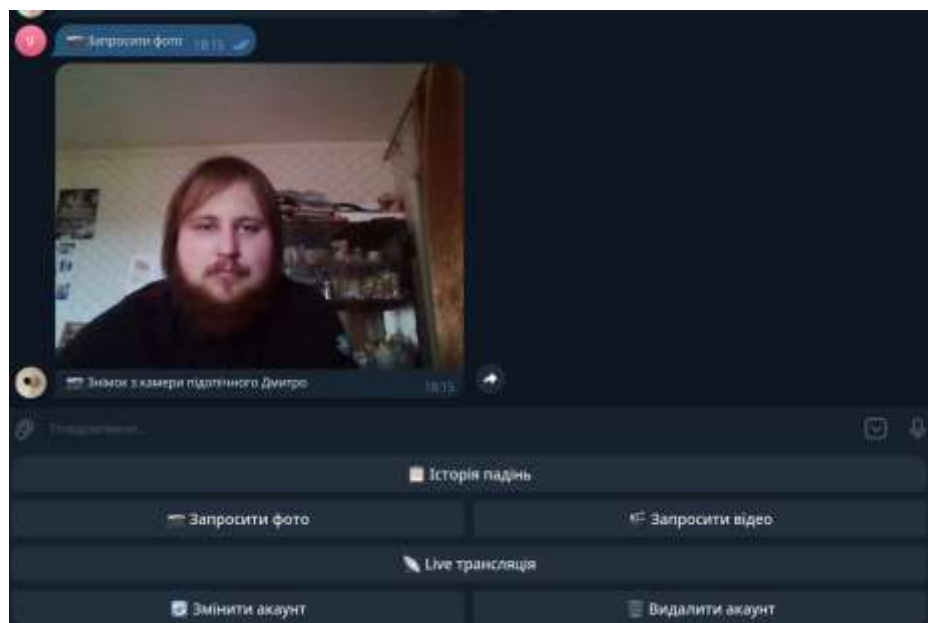


Рисунок 4.5 – Фото з камери

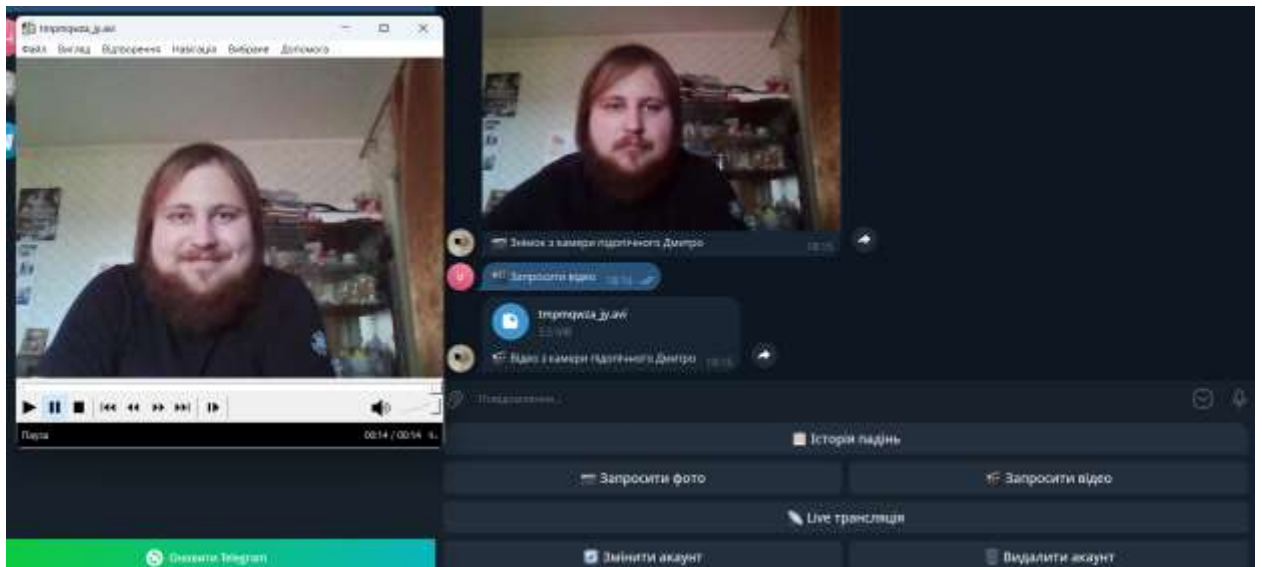


Рисунок 4.6 – Відео з камери

На рис. 4.7 показано реалізацію прямої трансляції.



Рисунок 4.7 – Пряма трансляція

Пряма трансляція реалізована таким чином, що кожну секунду опікуну приходить фото с камери користувача, також опікун може зупинити пряму трансляцію, натиснувши на кнопку "Зупинити трансляцію".

Якщо користувач впав і це зафіксувала камера, на рис. 4.8 показано, що користувачу приходить повідомлення з питанням “З вами все гаразд”, якщо користувач натиснув “Все добре”, повідомлення опікуну не відправляється, якщо ж користувач натиснув Ні, або не відповів на повідомлення опікуну відправляється тривожний сигнал.

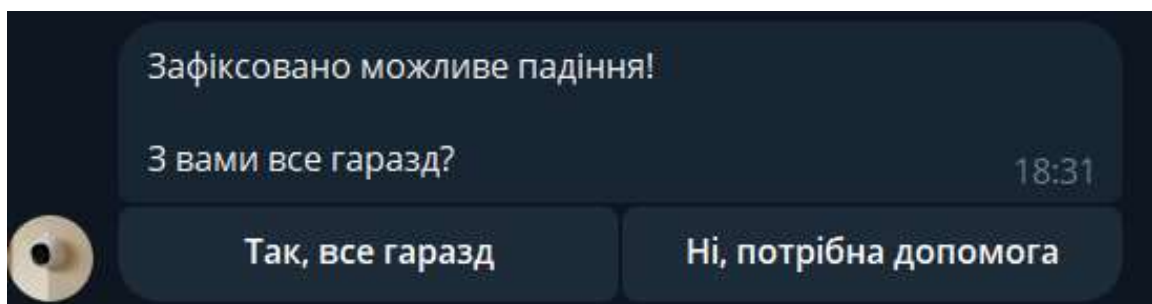


Рисунок 4.8 – Повідомлення про падіння користувачу

На рис. 4.9 показано повідомлення опікуну, якщо користувач не відповів або відповів негативно на повідомлення про падіння.

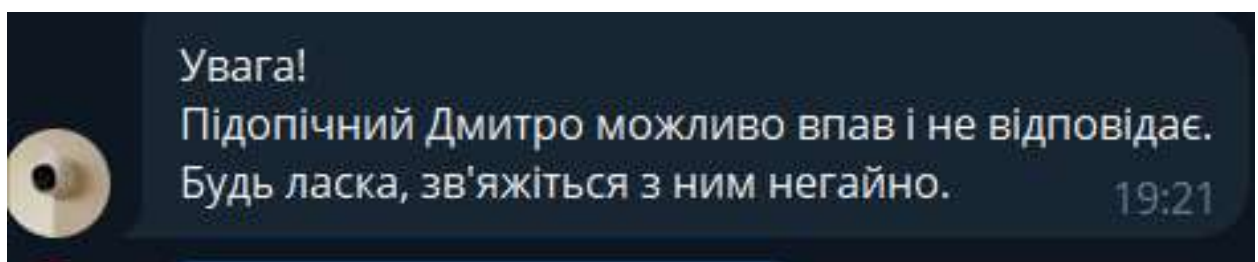


Рисунок 4.9 – Повідомлення опікуну про падіння

Якщо користувач, або опікун, хочуть дізнатися скільки раз, і коли впав користувач, вони можуть натиснути на кнопку “Історія падінь”, яка видасть їм історію тривожних подій.

На рис 4.10 показано повідомлення, якщо натиснути на кнопку “Історія падінь”.

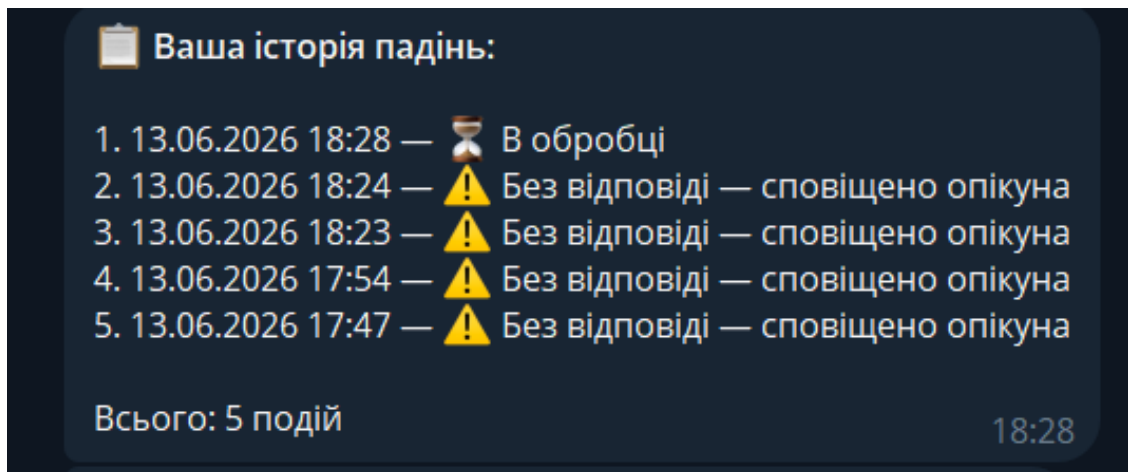


Рисунок 4.10 –Історією падінь

4.2 Експериментальне дослідження системи

Головним недоліком системи моніторингу безпеки людей похилого віку є можливість виникнення хибних спрацювань або пропуску реальних подій. Це пов'язано із складністю точного розпізнавання різних типів рухової активності людини, впливом умов навколишнього середовища, особливостями поведінки користувача. У результаті система може надсилати помилкові тривожні повідомлення або, навпаки, не зафіксувати небезпечну ситуацію.

Для доведення ефективності розробленого Telegram-бота було проведено експериментальне дослідження. Метою експерименту було порівняння хибної роботи системи моніторингу при використанні порогових методів та розробленої відео аналітичної підсистеми в однакових побутових умовах [5]-[9].

Завдання експерименту – о 13:40 користувач впав, обидві системи зафіксували падіння, і згенерували сигнал, потім він піднявся і ліг на ліжку, Протягом наступних 20 хвилин користувач перебував у стані спокою. О 14:05 користувач піднявся з ліжка, і знову впав, але вже більше не міг встати.

Результати виявлення падінь під час експерименту наведено в табл. 4.1.

Таблиця 4.1 – Порівняння систем зафіксування падінь[5]-[9]

Час	Дія користувача	Робота акселерометра та гіроскопів	Стан розробленої підсистеми (YOLO)
13:40	Падіння	Сигналу нема	Зафіксували сигнал
13:41	Користувач піднявся	Сигналу нема	Сигналу нема
13:45	Користувач ліг у ліжко	Зафіксували сигнал	Сигналу нема
14:05	Користувач піднявся	Сигналу нема	Сигналу нема
14:06	Користувач знову впав	Зафіксували сигнал	Зафіксували сигнал

Як видно з таблиці 4.1 акселерометри та гіроскопи продемонстрували нестабільну роботу під час першого падіння користувача. О 13:40 вони не зафіксували сигнал про падіння, що могло призвести до критичної ситуації. Більш того, після того як користувач ліг у ліжко о 13:45, вони зафіксували сигнал падіння, хоча користувач просто ліг в ліжко. Це могло призвести до не дуже гарної ситуації між користувачем та опікуном.

Отже експеримент показав, що точність детекції розробленої підсистеми становить 100%, тоді як акселерометри та гіроскопи демонструють 50% точності у цьому сценарії.

4.3 Висновок до розділу 4

У даному розділі успішно проведено комплексне тестування та експериментальне дослідження розробленого Telegram- бота з датчиками падіння.

Успішно протестовано всі ключові моменти бота: реєстрацію користувача і опікуна для прив'язки їх один до одного, їх функціонал, а також нейромережу YOLO для точної детекції падіння людини.

Виконано порівняльне експериментальне дослідження точності розробленої підсистеми у порівнянні з датчиками руху. Доведено, що завдяки використанню комп'ютерного зору, розроблена підсистема здатна безпомилково фіксувати падіння користувача. Це дозволило скоротити шанс непередбачених ситуацій, що дозволить врятувати на багато більше життів.

ВИСНОВКИ

У дипломній кваліфікаційній роботі вирішено актуальне питання практичне завдання, мета якого полягає у підвищенні безпеки людей на основі розроблення Telegram - бота з функцією детектування падіння.

У ході виконання роботи отримано такі результати:

- проаналізовано існуючі методи виявлення падінь та виявлено їхні недоліки;
- сформульовано вимоги до ПЗ підсистеми та обґрунтовано вибір інструментарію розробки (мова Python, платформа VisualStudioCode, бібліотека OpenCVpython, бібліотека psycorg2, неймережа YOLO);
- спроектовано Telegram – бот з інтерфейсом користувача для нагляду, та інтерфейс опікуна, який наглядає за користувачем;
- розроблено алгоритм фільтрації хибних спрацювань на основі ігнорування хибних падінь для уникнення складних ситуацій, а також реалізовано незалежні мікросервіси для перевірки опікуном стану користувача;
- виконано комплексне тестування розробленого Telegram – бота та доведено, що підсистема на базі YOLO стабільно фіксує стан людини та ігнорує планове або статичне лежання користувача.

Експериментальне дослідження підтвердило, що розроблена підсистема дозволяє врятувати більше життів людей похилого віку з більшою точністю до детектування падінь з 50%(акселерометри та гіроскопи) до 100%. Отримані результати підтверджують придатність програмної підсистеми датчиків для успішного моніторингу безпеки.

Можливими шляхами подальшого розвитку проєкту є розширення його функціоналу, більш розширену безпеку акаунтів, додавання більш надійнішої перевірки здоров'я користувачів та використання в інших сферах діяльності (наприклад радіо – няня).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. World Health Organization. Falls. URL: <https://www.who.int/news-room/fact-sheets/detail/falls> (date of access: 15.06.2026).
2. Falls in Older Adults. MSD Manuals. URL: <https://www.msdmanuals.com/uk/professional/geriatrics/falls-in-older-adults/falls-in-older-adults> (date of access: 30.05.2026).
3. Fall Detection Systems Review. Sensors. URL: <https://www.mdpi.com/1424-8220/25/24/7423> (date of access: 30.05.2026).
4. The Methods of Fall Detection: A Literature Review. Sensors. URL: <https://www.mdpi.com/1424-8220/23/11/5212> (date of access: 30.05.2026).
5. Shany T., Redmond S. J., Narayanan M. R., Lovell N. H. Sensors-based wearable systems for monitoring of human movement and falls. IEEE Sensors Journal. 2012. Vol. 12, No. 3. P. 658–670.
6. Igual R., Medrano C., Plaza I. Challenges, issues and trends in fall detection systems. BioMedical Engineering Online. 2013. Vol. 12. URL: <https://link.springer.com/article/10.1186/1475-925X-12-66> (date of access: 30.05.2026).
7. Vision-based Fall Detection with Deep Learning. URL: <https://arxiv.org/abs/2207.10952> (date of access: 30.05.2026).
8. OpenCV Documentation. URL: <https://opencv.org/> (дата звернення: 30.05.2026).
9. UltralyticsYOLODocumentation. URL: <https://docs.ultralytics.com/> (date of access: 30.05.2026).
10. Telegram. Bot API Documentation. URL: <https://core.telegram.org/bots/api> (date of access: 30.05.2026).
11. OWASP Foundation. Application Security Verification Standard. URL: <https://owasp.org/www-project-application-security-verification-standard/> (date of access: 30.05.2026).
12. pyTelegramBotAPI Documentation.

URL:<https://core.telegram.org/bots/api> (date of access: 30.05.2026).

13. PostgreSQL Documentation. URL: <https://www.postgresql.org/docs/> (date of access: 30.05.2026).

14. Python Documentation. URL: <https://docs.python.org/3/> (date of access: 30.05.2026).

15. Visual Studio Code Documentation. URL: <https://code.visualstudio.com/docs> (date of access: 03.06.2026).

16. Java Documentation. URL: <https://docs.oracle.com/en/java/javase/> (date of access: 03.06.2026).

17. C++ Documentation. URL: <https://en.cppreference.com/> (date of access: 30.05.2026).

18. C# Documentation. URL: <https://learn.microsoft.com/en-us/dotnet/csharp/> (date of access: 03.06.2026).

19. Pillow Documentation. URL: <https://pillow.readthedocs.io/en/stable/> (date of access: 03.06.2026).

20. Skikit – image Documentation. URL: <https://scikit-image.org/docs/stable/> (date of access: 03.06.2026).

21. ImageIO Documentation. URL: <https://imageio.readthedocs.io/en/stable/> (date of access: 03.06.2026).

22. Faster R-CNN Documentation. URL: <https://arxiv.org/abs/1506.01497> (date of access: 03.06.2026).

23. SSD Documentation. URL: <https://arxiv.org/abs/1512.02325> (date of access: 03.06.2026).

24. Haar Cascade Documentation. URL: https://docs.opencv.org/4.13.0/dc/d88/tutorial_traincascade.html (date of access: 03.06.2026).

25. Python – telegram- bot Documentation. URL: <https://docs.python-telegram-bot.org/en/stable/> (date of access: 03.06.2026).

26. Aiogram Documentation. URL: <https://docs.aiogram.dev/en/v3.28.2/> (date of access: 03.06.2026).

27. Node.jsDocumentation. URL: <https://telegraf.js.org/> (date of access: 03.06.2026).
28. MySQLDocumentation. URL: <https://dev.mysql.com/doc/> (date of access: 03.06.2026).
29. MySQLDocumentation. URL: <https://dev.mysql.com/doc/> (date of access: 03.06.2026).
30. SQLite Documentation. URL: <https://www.sqlite.org/docs.html> (date of access: 03.06.2026).
31. MongoDB Documentation. URL: <https://www.mongodb.com/docs/> (date of access: 03.06.2026).
32. PyCharmDocumentation. URL: <https://www.jetbrains.com/help/pycharm/getting-started.html> (date of access: 03.06.2026).
33. Visual Studio Documentation. URL: <https://learn.microsoft.com/ru-ru/visualstudio/?view=visualstudio> (date of access: 03.06.2026).
34. Jupiter Notebook Documentation. URL: <https://jupyter-notebook.readthedocs.io/en/stable/> (date of access: 03.06.2026).
35. Psycopg 2.9.12 documentationDocumentation. URL: <https://www.psycopg.org/docs/> (date of access: 03.06.2026).

ДОДАТОК А
Текст програми

A.1 Текстфайлу main.py

```

import os
import telebot
from telebot.types import ReplyKeyboardMarkup, KeyboardButton
from dotenv import load_dotenv

load_dotenv()

import database as db
import notifications
import handlers_user as hu
import handlers_guardian as hg

TOKEN = os.getenv("BOT_TOKEN")

bot = telebot.TeleBot(TOKEN, parse_mode=None)

@bot.message_handler(commands=["start"])
def cmd_start(message):
    kb = ReplyKeyboardMarkup(resize_keyboard=True)
    kb.add(KeyboardButton("👤 Користувач"), KeyboardButton("👤👤 Опікун"))
    bot.send_message(
        message.chat.id,
        "👤 Вітаємо у системі моніторингу безпеки!\n\n"
        "Оберіть вашу роль:",
        reply_markup=kb
    )

@bot.message_handler(func=lambda m: m.text == "👤 Користувач")
def role_user(message):
    hu.start_user_flow(bot, message)

@bot.message_handler(func=lambda m: m.text == "👤👤 Опікун")
def role_guardian(message):

```

```

hg.start_guardian_flow(bot, message)

@bot.callback_query_handler(func=lambda c: c.data.startswith("fall_ok:") or
c.data.startswith("fall_no:"))
def fall_response(call):
    parts = call.data.split(":")
    action = parts[0]
    alert_id = int(parts[1])
    response = "ok" if action == "fall_ok" else "no"

    notifications.handle_fall_response(bot, call.from_user.id, alert_id, response)
    bot.answer_callback_query(call.id)

    try:
        bot.edit_message_reply_markup(call.message.chat.id, call.message.message_id,
reply_markup=None)
    except Exception:
        pass

print("Ініціалізація бази даних...")
db.init_db()
print("База даних готова.")
hu.register_handlers(bot)
hg.register_handlers(bot)

print("Бот запущено. Очікування повідомлень...")
bot.infinity_polling(timeout=30, long_polling_timeout=20)

```

A.2 Текст файлу auth.py

```

import bcrypt
import re
from datetime import datetime
import database as db

```

```
CAMERA_TYPES = ("webcam", "ip", "screen")
```

```
def hash_password(password: str) -> str:
```

```
    return bcrypt.hashpw(password.encode(), bcrypt.gensalt()).decode()
```

```
def verify_password(password: str, hashed: str) -> bool:
```

```
    return bcrypt.checkpw(password.encode(), hashed.encode())
```

```
def _validate_email(email: str) -> bool:
```

```
    return bool(re.match(r"^\w\.-]+@[\w\.-]+\.\w{2,}$", email))
```

```
def _validate_phone(phone: str) -> bool:
```

```
    return bool(re.match(r"^\+?\d{10,15}$", phone))
```

```
def _validate_date(date_str: str):
```

```
    for fmt in ("%d.%m.%Y", "%d/%m/%Y", "%Y-%m-%d"):
```

```
        try:
```

```
            return datetime.strptime(date_str, fmt).date()
```

```
        except ValueError:
```

```
            continue
```

```
    return None
```

```
def register_user(login: str, password: str, name: str, birth_date_str: str,
```

```
                  email: str, guardian_phone: str,
```

```
                  camera_type: str, camera_source: str, tg_id=None):
```

```
    \
```

```
    \
```

```
    login = login.strip()
```

```
    password = password.strip()
```

```
    name = name.strip()
```

```
    email = email.strip().lower()
```

```
    guardian_phone = guardian_phone.strip()
```

```
    camera_type = camera_type.strip().lower()
```

```
    camera_source = camera_source.strip()
```

```

if len(login) < 3:
    return False, "Логін має бути не менше 3 символів."
if len(password) < 6:
    return False, "Пароль має бути не менше 6 символів."
if not name:
    return False, "Ім'я не може бути порожнім."
if not _validate_email(email):
    return False, "Невірний формат пошти."
if not _validate_phone(guardian_phone):
    return False, "Невірний номер телефону опікуна."
if camera_type not in CAMERA_TYPES:
    return False, f"Тип камери має бути одним із: {' '.join(CAMERA_TYPES)}."
if not camera_source:
    return False, "Вкажіть джерело камери."

birth_date = _validate_date(birth_date_str)
if not birth_date:
    return False, "Невірний формат дати. Використовуйте ДД.ММ.РРРР."

if db.get_user_by_login(login):
    return False, "Користувач з таким логіном вже існує."
if db.get_user_by_email(email):
    return False, "Користувач з такою поштою вже існує."

hashed = hash_password(password)
user_id = db.create_user(
    login, hashed, name, birth_date, email,
    guardian_phone, camera_type, camera_source, tg_id
)
return True, user_id

def login_user(login: str, password: str):
    \
    \

```

\
 \

```
login = login.strip()
user = db.get_user_by_login(login)
if not user:
    return False, "Користувача з таким логіном не знайдено."
if not verify_password(password, user[2]):
    return False, "Невірний пароль."
return True, user
```

```
def restore_user_password(email: str):
```

\
 \
 \

```
email = email.strip().lower()
if not _validate_email(email):
    return False, "Невірний формат пошти."
user = db.get_user_by_email(email)
if not user:
    return False, "Акаунт з такою поштою не знайдено."
return True, user
```

```
def change_user_password(user_id: int, old_password: str, new_password: str):
```

```
user = db.get_user_by_login
conn = db.get_conn()
try:
    with conn.cursor() as cur:
        cur.execute("SELECT password FROM users WHERE id = %s", (user_id,))
        row = cur.fetchone()
finally:
    db.release_conn(conn)
```

```
if not row:
```

```

        return False, "Користувача не знайдено."
    if not verify_password(old_password, row[0]):
        return False, "Старий пароль невірний."
    if len(new_password.strip()) < 6:
        return False, "Новий пароль має бути не менше 6 символів."

    db.update_user_password(user_id, hash_password(new_password.strip()))
    return True, "Пароль змінено."

def register_guardian(login: str, password: str, name: str,
                      email: str, phone: str, tg_id=None):
    \
    \

    login = login.strip()
    password = password.strip()
    name = name.strip()
    email = email.strip().lower()
    phone = phone.strip()

    if len(login) < 3:
        return False, "Логін має бути не менше 3 символів."
    if len(password) < 6:
        return False, "Пароль має бути не менше 6 символів."
    if not name:
        return False, "Ім'я не може бути порожнім."
    if not _validate_email(email):
        return False, "Невірний формат пошти."
    if not _validate_phone(phone):
        return False, "Невірний формат номеру телефону."

    if db.get_guardian_by_login(login):
        return False, "Опікун з таким логіном вже існує."
    if db.get_guardian_by_email(email):
        return False, "Опікун з такою поштою вже існує."

```

```

if db.get_guardian_by_phone(phone):
    return False, "Опікун з таким телефоном вже існує."

```

```

hashed = hash_password(password)
guardian_id = db.create_guardian(login, hashed, name, email, phone, tg_id)
return True, guardian_id

```

```

def login_guardian(login: str, password: str):

```

```

\
\
\

```

```

    login = login.strip()
    guardian = db.get_guardian_by_login(login)
    if not guardian:
        return False, "Опікуна з таким логіном не знайдено."
    if not verify_password(password, guardian[2]):
        return False, "Невірний пароль."
    return True, guardian

```

```

def restore_guardian_password(email: str):

```

```

\
\

```

```

    email = email.strip().lower()
    if not _validate_email(email):
        return False, "Невірний формат пошти."
    guardian = db.get_guardian_by_email(email)
    if not guardian:
        return False, "Акаунт з такою поштою не знайдено."
    return True, guardian

```

```

def change_guardian_password(guardian_id: int, old_password: str, new_password:
str):
    conn = db.get_conn()

```

```

try:
    with conn.cursor() as cur:
        cur.execute("SELECT password FROM guardians WHERE id = %s",
(guardian_id,))
        row = cur.fetchone()
    finally:
        db.release_conn(conn)

    if not row:
        return False, "Опікуна не знайдено."
    if not verify_password(old_password, row[0]):
        return False, "Старий пароль невірний."
    if len(new_password.strip()) < 6:
        return False, "Новий пароль має бути не менше 6 символів."

    db.update_guardian_password(guardian_id, hash_password(new_password.strip()))
    return True, "Пароль змінено."

```

A.3 Текстфайлу camera.py

```

import cv2
import threading
import time
import tempfile
import numpy as np
import math as _math

try:
    import mss
    MSS_AVAILABLE = True
except ImportError:
    MSS_AVAILABLE = False

try:
    from ultralytics import YOLO

```

```

        YOLO_AVAILABLE = True
except ImportError:
    YOLO_AVAILABLE = False
MIN_KP_CONF    = 0.15
DT_FALL        = 0.4
UPRIGHT_ANGLE  = 62
GROUND_ANGLE   = 45
HIP_DROP_THRESH = 0.10
BODY_DROP_THRESH = 0.12
WAS_UPRIGHT_MIN = 60
HISTORY_FRAMES = 100
FALL_COOLDOWN  = 30

_SKELETON =
[(5,6),(5,11),(6,12),(11,12),(5,7),(7,9),(6,8),(8,10),(11,13),(13,15),(12,14),(14,16)]

_yolo_model = None
_detection_threads: dict[int, threading.Event] = {}
_live_stop = threading.Event()

def get_value_ago(history, target_dt):
    if not history:
        return None
    now_t = history[-1][0]
    if now_t - history[0][0] < target_dt * 0.7:
        return None
    for t, val in reversed(history):
        if now_t - t >= target_dt:
            return val
    return history[0][1]

def _load_yolo(model_path: str = "yolov8n.pt") -> "YOLO":
    global _yolo_model

```

```

if _yolo_model is None:
    if not YOLO_AVAILABLE:
        raise RuntimeError("ultralytics is not installed.")
_yolo_model = YOLO(model_path)
return _yolo_model

def _open_capture(camera_type: str, camera_source: str) -> cv2.VideoCapture:
    ct = camera_type.lower()
    if ct == "webcam":
        index = int(camera_source) if camera_source.isdigit() else 0
        cap = cv2.VideoCapture(index)
    elif ct == "ip":
        cap = cv2.VideoCapture(camera_source)
    else:
        raise RuntimeError(f"Unsupported camera type: {camera_type}")
    if not cap.isOpened():
        raise RuntimeError(f"Could not open camera: {camera_type} / {camera_source}")
    return cap

def _grab_screen() -> np.ndarray:
    if not MSS_AVAILABLE:
        raise RuntimeError("mss is not installed.")
    with mss.mss() as sct:
        monitor = sct.monitors[1]
        img = sct.grab(monitor)
        frame = np.array(img)
        return cv2.cvtColor(frame, cv2.COLOR_BGRA2BGR)

def _read_frame(camera_type: str, cap: cv2.VideoCapture | None) -> np.ndarray:
    if camera_type.lower() == "screen":
        return _grab_screen()
    ret, frame = cap.read()

```

```

if not ret:
    raise RuntimeError("Failed to read frame.")
return frame

```

```

def _encode_jpeg(frame: np.ndarray) -> bytes:
    ok, buf = cv2.imencode(".jpg", frame)
    if not ok:
        raise RuntimeError("Failed to encode frame.")
    return buf.tobytes()

```

```

def _draw_pose(frame: np.ndarray) -> np.ndarray:
    try:
        model = _load_yolo("yolov8s-pose.pt")
        results = model(frame, verbose=False)
        for r in results:
            if r.keypoints is None or r.keypoints.conf is None:
                continue
            for idx in range(len(r.keypoints.conf)):
                kp = r.keypoints.xy[idx].tolist()
                conf = r.keypoints.conf[idx].tolist()
                for i, j in _SKELETON:
                    if conf[i] >= MIN_KP_CONF and conf[j] >= MIN_KP_CONF:
                        cv2.line(frame,
                                (int(kp[i][0]), int(kp[i][1])),
                                (int(kp[j][0]), int(kp[j][1])),
                                (0, 255, 0), 2)
                for i, (x, y) in enumerate(kp):
                    if conf[i] >= MIN_KP_CONF:
                        cv2.circle(frame, (int(x), int(y)), 4, (0, 255, 0), -1)
    except Exception:
        pass
    return frame

```

```

def _hip_y(result, person_idx: int, frame_h: int) -> float | None:
    if result.keypoints is None or result.keypoints.conf is None:
        return None
    if person_idx >= len(result.keypoints.conf):
        return None
    kp = result.keypoints.xy[person_idx].tolist()
    conf = result.keypoints.conf[person_idx].tolist()
    if conf[11] < MIN_KP_CONF or conf[12] < MIN_KP_CONF:
        return None
    return ((kp[11][1] + kp[12][1]) / 2) / frame_h

```

```

def _shoulder_y(result, person_idx: int, frame_h: int) -> float | None:
    if result.keypoints is None or result.keypoints.conf is None:
        return None
    if person_idx >= len(result.keypoints.conf):
        return None
    kp = result.keypoints.xy[person_idx].tolist()
    conf = result.keypoints.conf[person_idx].tolist()
    if conf[5] < MIN_KP_CONF or conf[6] < MIN_KP_CONF:
        return None
    return ((kp[5][1] + kp[6][1]) / 2) / frame_h

```

```

def _torso_angle(result, person_idx: int = 0) -> float | None:
    if result.keypoints is None or result.keypoints.conf is None:
        return None
    if person_idx >= len(result.keypoints.conf):
        return None
    kp = result.keypoints.xy[person_idx].tolist()
    conf = result.keypoints.conf[person_idx].tolist()
    for idx in (5, 6, 11, 12):
        if conf[idx] < MIN_KP_CONF:
            return None
    sx = (kp[5][0] + kp[6][0]) / 2

```

```

sy = (kp[5][1] + kp[6][1]) / 2
hx = (kp[11][0] + kp[12][0]) / 2
hy = (kp[11][1] + kp[12][1]) / 2
return _math.degrees(_math.atan2(abs(sy - hy), abs(sx - hx)))

```

```

def get_photo(camera_type: str, camera_source: str) -> bytes:
    if camera_type.lower() == "screen":
        frame = _grab_screen()
    else:
        cap = _open_capture(camera_type, camera_source)
        try:
            frame = _read_frame(camera_type, cap)
        finally:
            cap.release()
    return _encode_jpeg(_draw_pose(frame))

```

```

def get_video(camera_type: str, camera_source: str, duration: int = 10) -> str:
    fps = 20
    if camera_type.lower() == "screen":
        sample = _grab_screen()
        h, w = sample.shape[:2]
        cap = None
    else:
        cap = _open_capture(camera_type, camera_source)
        w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    tmp = tempfile.NamedTemporaryFile(suffix=".avi", delete=False)
    tmp.close()
    out = cv2.VideoWriter(tmp.name, cv2.VideoWriter_fourcc(*"XVID"), fps, (w, h))
    try:
        deadline = time.time() + duration
        while time.time() < deadline:
            frame = _draw_pose(_read_frame(camera_type, cap))

```

```

        out.write(frame)
    finally:
        out.release()
    if cap:
        cap.release()
    return tmp.name

```

```

def start_live(camera_type: str, camera_source: str, frame_callback, fps: int = 5) ->
None:

```

```

    global _live_stop
    _live_stop.clear()
    delay = 1.0 / fps

```

```

    def _stream():
        cap = None if camera_type.lower() == "screen" else _open_capture(camera_type,
camera_source)
        try:
            while not _live_stop.is_set():
                try:
                    frame = _draw_pose(_read_frame(camera_type, cap))
                    frame_callback(_encode_jpeg(frame))
                except Exception:
                    pass
                time.sleep(delay)
        finally:
            if cap:
                cap.release()

```

```

    threading.Thread(target=_stream, daemon=True).start()

```

```

def stop_live() -> None:
    _live_stop.set()

```

```

def start_detection(user_id: int, camera_type: str, camera_source: str,
                    fall_callback, model_path: str = "yolov8s-pose.pt") -> None:
    if user_id in _detection_threads:
        _detection_threads[user_id].set()

    stop_event = threading.Event()
    _detection_threads[user_id] = stop_event

def _detect():
    try:
        model = _load_yolo(model_path)
    except Exception as e:
        print(f"[detection] ✖ Не вдалось завантажити модель: {e}")
        return
    tracked: dict[int, dict] = {}
    try:
        cap = None
        if camera_type.lower() == "screen":
            else
            _open_capture(camera_type, camera_source)
        except Exception as e:
            print(f"[detection] ✖ Не вдалось відкрити камеру: {e}")
            return
        print(f"[detection] ✔ Детекція запущена для user_id={user_id}")

    try:
        while not stop_event.is_set():
            fall_detected = False
            try:
                frame = _read_frame(camera_type, cap)
                fh = frame.shape[0]
                results = model.track(frame, persist=True, verbose=False)
                active_ids: set[int] = set()

                for r in results:
                    if r.bboxes.id is None:
                        continue

```

```

ids = r.bboxes.id.tolist()

for i, tid in enumerate(ids):
    tid = int(tid)
    active_ids.add(tid)

    angle = _torso_angle(r, i)
    hy = _hip_y(r, i, fh)
    sy = _shoulder_y(r, i, fh)

    bbox = r.bboxes.xyxy[i].tolist()
    bw = bbox[2] - bbox[0]
    bh = bbox[3] - bbox[1]
    aspect = bh / bw if bw > 0 else 2.0

    if tid not in tracked:
        tracked[tid] = {"ah": [], "hh": [], "sh": [], "ar": []}

    now = time.time()
    data = tracked[tid]

    if angle is not None:
        data["ah"].append((now, angle))
        if len(data["ah"]) > HISTORY_FRAMES:
            data["ah"].pop(0)
    if hy is not None:
        data["hh"].append((now, hy))
        if len(data["hh"]) > HISTORY_FRAMES:
            data["hh"].pop(0)
    if sy is not None:
        data["sh"].append((now, sy))
        if len(data["sh"]) > HISTORY_FRAMES:
            data["sh"].pop(0)
    data["ar"].append((now, aspect))
    if len(data["ar"]) > HISTORY_FRAMES:

```

```

data["ar"].pop(0)

ah = data["ah"]
hh = data["hh"]
sh = data["sh"]
ar = data["ar"]
was_upright = len(ah) > 0 and max(val for t, val in ah) >
WAS_UPRIGHT_MIN
was_tall = len(ar) > 0 and max(val for t, val in ar) > 1.3

sig1 = False
if angle is not None and angle < GROUND_ANGLE and was_upright
and aspect < 1.0:

    angle_ago = get_value_ago(ah, 0.8)
    if angle_ago is not None and angle_ago > UPRIGHT_ANGLE:
        sig1 = True

sig2 = False
if hy is not None and was_upright:
    hip_ago = get_value_ago(hh, DT_FALL)
    if hip_ago is not None:
        hip_drop = hy - hip_ago
        sig2 = (hip_drop > HIP_DROP_THRESH and
                angle is not None and angle < GROUND_ANGLE)

sig3 = False
if sy is not None and hy is not None and was_upright:
    sh_ago = get_value_ago(sh, DT_FALL)
    hip_ago3 = get_value_ago(hh, DT_FALL)
    if sh_ago is not None and hip_ago3 is not None:
        sh_drop = sy - sh_ago
        hd3 = hy - hip_ago3
        sig3 = (sh_drop > BODY_DROP_THRESH and hd3 >
BODY_DROP_THRESH
                and angle is not None and angle < GROUND_ANGLE)

```

```

        sig_bbox = False
        if aspect < 0.9 and was_tall:
            ar_ago = get_value_ago(ar, 0.6)
            if ar_ago is not None and ar_ago > 1.3:
                sig_bbox = True

        if sig1 or sig2 or sig3 or sig_bbox:
            data["ah"].clear()
            data["hh"].clear()
            data["sh"].clear()
            data["ar"].clear()
            fall_detected = True

        for lost in set(tracked) - active_ids:
            tracked.pop(lost)

    except Exception:
        pass

    if fall_detected:
        fall_detected = False
        try:
            fall_callback()
        except Exception as e:
            print(f"[detection] ✕fall_callback error: {e}")
            stop_event.wait(timeout=FALL_COOLDOWN)

        time.sleep(0.05)
    finally:
        if cap:
            cap.release()
        _detection_threads.pop(user_id, None)

threading.Thread(target=_detect, daemon=True).start()

```

```
def stop_detection(user_id: int) -> None:
    if user_id in _detection_threads:
        _detection_threads[user_id].set()
```

А.4 Текстфайл database.py

```
import os
import psycopg2
from psycopg2 import pool
from datetime import datetime
from dotenv import load_dotenv

load_dotenv()

DB_CONFIG = {
    "host": os.getenv("DB_HOST", "localhost"),
    "port": int(os.getenv("DB_PORT", 5432)),
    "dbname": os.getenv("DB_NAME", "fall_detection"),
    "user": os.getenv("DB_USER", "postgres"),
    "password": os.getenv("DB_PASSWORD", ""),
}

_pool = None

def init_pool():
    global _pool
    _pool = psycopg2.pool.SimpleConnectionPool(1, 10, **DB_CONFIG)

def get_conn():
    return _pool.getconn()

def release_conn(conn):
    _pool.putconn(conn)
```

```

def init_db():
    init_pool()
    conn = get_conn()
    try:
        with conn.cursor() as cur:
            cur.execute("""
                CREATE TABLE IF NOT EXISTS guardians (
                    id      SERIAL PRIMARY KEY,
                    login   VARCHAR(100) UNIQUE NOT NULL,
                    password VARCHAR(255) NOT NULL,
                    name    VARCHAR(150) NOT NULL,
                    email   VARCHAR(150) UNIQUE NOT NULL,
                    phone   VARCHAR(20) UNIQUE NOT NULL,
                    tg_id   BIGINT UNIQUE
                )
            """)

            cur.execute("""
                CREATE TABLE IF NOT EXISTS users (
                    id      SERIAL PRIMARY KEY,
                    login   VARCHAR(100) UNIQUE NOT NULL,
                    password VARCHAR(255) NOT NULL,
                    name    VARCHAR(150) NOT NULL,
                    birth_date DATE NOT NULL,
                    email   VARCHAR(150) UNIQUE NOT NULL,
                    guardian_phone VARCHAR(20) NOT NULL,
                    camera_type VARCHAR(20) NOT NULL,
                    camera_source VARCHAR(255) NOT NULL,
                    tg_id   BIGINT UNIQUE
                )
            """)

            cur.execute("""
                CREATE TABLE IF NOT EXISTS alerts (
                    id      SERIAL PRIMARY KEY,

```

```

        user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
        guardian_id INTEGER REFERENCES guardians(id) ON DELETE SET
NULL,

        detected_at TIMESTAMP NOT NULL DEFAULT NOW(),
        responded BOOLEAN,
        response VARCHAR(10)
    )
    """

```

```

        conn.commit()
    finally:
        release_conn(conn)

```

```

def create_user(login, password, name, birth_date, email,
                guardian_phone, camera_type, camera_source, tg_id=None):
    conn = get_conn()
    try:
        with conn.cursor() as cur:
            cur.execute("""
                INSERT INTO users
                (login, password, name, birth_date, email,
                 guardian_phone, camera_type, camera_source, tg_id)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
                RETURNING id
            """, (login, password, name, birth_date, email,
                 guardian_phone, camera_type, camera_source, tg_id))
            user_id = cur.fetchone()[0]
        conn.commit()
        return user_id
    finally:
        release_conn(conn)

```

```

def get_user_by_login(login):
    conn = get_conn()
    try:

```

```

        with conn.cursor() as cur:
            cur.execute("SELECT * FROM users WHERE login = %s", (login,))
            return cur.fetchone()
    finally:
        release_conn(conn)

def get_user_by_email(email):
    conn = get_conn()
    try:
        with conn.cursor() as cur:
            cur.execute("SELECT * FROM users WHERE email = %s", (email,))
            return cur.fetchone()
    finally:
        release_conn(conn)

def get_user_by_tg_id(tg_id):
    conn = get_conn()
    try:
        with conn.cursor() as cur:
            cur.execute("SELECT * FROM users WHERE tg_id = %s", (tg_id,))
            return cur.fetchone()
    finally:
        release_conn(conn)

def update_user_tg_id(user_id, tg_id):
    conn = get_conn()
    try:
        with conn.cursor() as cur:
            cur.execute("UPDATE users SET tg_id = %s WHERE id = %s", (tg_id,
user_id))
        conn.commit()
    finally:
        release_conn(conn)

def update_user_password(user_id, new_password):
    conn = get_conn()

```

```

try:
    with conn.cursor() as cur:
        cur.execute("UPDATE users SET password = %s WHERE id = %s",
(new_password, user_id))
        conn.commit()
finally:
    release_conn(conn)

```

```

def delete_user(user_id):
    conn = get_conn()
    try:
        with conn.cursor() as cur:
            cur.execute("DELETE FROM users WHERE id = %s", (user_id,))
            conn.commit()
    finally:
        release_conn(conn)

```

```

def create_guardian(login, password, name, email, phone, tg_id=None):
    conn = get_conn()
    try:
        with conn.cursor() as cur:
            cur.execute("""
                INSERT INTO guardians (login, password, name, email, phone, tg_id)
                VALUES (%s, %s, %s, %s, %s, %s)
                RETURNING id
            """, (login, password, name, email, phone, tg_id))
            guardian_id = cur.fetchone()[0]
            conn.commit()
            return guardian_id
    finally:
        release_conn(conn)

```

```

def get_guardian_by_login(login):
    conn = get_conn()
    try:

```

```

        with conn.cursor() as cur:
            cur.execute("SELECT * FROM guardians WHERE login = %s", (login,))
            return cur.fetchone()
    finally:
        release_conn(conn)

def get_guardian_by_email(email):
    conn = get_conn()
    try:
        with conn.cursor() as cur:
            cur.execute("SELECT * FROM guardians WHERE email = %s", (email,))
            return cur.fetchone()
    finally:
        release_conn(conn)

def get_guardian_by_phone(phone):
    conn = get_conn()
    try:
        with conn.cursor() as cur:
            cur.execute("SELECT * FROM guardians WHERE phone = %s", (phone,))
            return cur.fetchone()
    finally:
        release_conn(conn)

def get_guardian_by_tg_id(tg_id):
    conn = get_conn()
    try:
        with conn.cursor() as cur:
            cur.execute("SELECT * FROM guardians WHERE tg_id = %s", (tg_id,))
            return cur.fetchone()
    finally:
        release_conn(conn)

def update_guardian_tg_id(guardian_id, tg_id):
    conn = get_conn()

```

```

try:
    with conn.cursor() as cur:
        cur.execute("UPDATE guardians SET tg_id = %s WHERE id = %s", (tg_id,
guardian_id))
    conn.commit()
finally:
    release_conn(conn)

def update_guardian_password(guardian_id, new_password):
    conn = get_conn()
    try:
        with conn.cursor() as cur:
            cur.execute("UPDATE guardians SET password = %s WHERE id = %s",
(new_password, guardian_id))
        conn.commit()
    finally:
        release_conn(conn)

def delete_guardian(guardian_id):
    conn = get_conn()
    try:
        with conn.cursor() as cur:
            cur.execute("DELETE FROM guardians WHERE id = %s", (guardian_id,))
        conn.commit()
    finally:
        release_conn(conn)

def log_fall(user_id, guardian_id):
    conn = get_conn()
    try:
        with conn.cursor() as cur:
            cur.execute("""
                INSERT INTO alerts (user_id, guardian_id, detected_at)
                VALUES (%s, %s, %s)
                RETURNING id
            """)

```

```

        """', (user_id, guardian_id, datetime.now()))
        alert_id = cur.fetchone()[0]
        conn.commit()
        return alert_id
    finally:
        release_conn(conn)

def update_fall_response(alert_id, responded, response=None):
    conn = get_conn()
    try:
        with conn.cursor() as cur:
            cur.execute("""
                UPDATE alerts
                SET responded = %s, response = %s
                WHERE id = %s
            """, (responded, response, alert_id))
        conn.commit()
    finally:
        release_conn(conn)

def get_fall_history_by_user(user_id):
    conn = get_conn()
    try:
        with conn.cursor() as cur:
            cur.execute("""
                SELECT id, detected_at, responded, response
                FROM alerts
                WHERE user_id = %s
                ORDER BY detected_at DESC
            """, (user_id,))
        return cur.fetchall()
    finally:
        release_conn(conn)

def get_fall_history_by_guardian(guardian_id):

```

```

conn = get_conn()
try:
    with conn.cursor() as cur:
        cur.execute("""
            SELECT a.id, u.name, a.detected_at, a.responded, a.response
            FROM alerts a
            JOIN users u ON u.id = a.user_id
            WHERE a.guardian_id = %s
            ORDER BY a.detected_at DESC
            """, (guardian_id,))
        return cur.fetchall()
finally:
    release_conn(conn)

```

A.5 Текстфайлыhandlers_guardian.py

```

import os
import auth
import database as db
import camera
from telebot.types import (
    ReplyKeyboardMarkup, KeyboardButton,
    ReplyKeyboardRemove, InlineKeyboardMarkup, InlineKeyboardButton
)

G_IDLE    = "g_idle"
G_LOGIN   = "g_login"
G_PASSWORD = "g_password"
G_FORGOT  = "g_forgot"
G_REG_LOGIN = "g_reg_login"
G_REG_PASS = "g_reg_pass"
G_REG_NAME = "g_reg_name"
G_REG_EMAIL = "g_reg_email"
G_REG_PHONE = "g_reg_phone"
G_MENU    = "g_menu"

```

```

_sessions: dict[int, dict] = {}

def _sess(tg_id: int) -> dict:
    if tg_id not in _sessions:
        _sessions[tg_id] = {"step": G_IDLE, "data": {}, "guardian": None}
    return _sessions[tg_id]

def _step(tg_id: int) -> str:
    return _sessions.get(tg_id, {}).get("step", "")

def _set(tg_id: int, step: str) -> None:
    _sess(tg_id)["step"] = step

def _put(tg_id: int, key: str, val) -> None:
    _sess(tg_id)["data"][key] = val

def _get(tg_id: int, key: str):
    return _sess(tg_id)["data"].get(key)

def _clear(tg_id: int) -> None:
    _sessions.pop(tg_id, None)

def _kb_auth():
    kb = ReplyKeyboardMarkup(resize_keyboard=True)
    kb.add(KeyboardButton("👤 Реєстрація"), KeyboardButton("👤 Вхід"))
    return kb

def _kb_menu():
    kb = ReplyKeyboardMarkup(resize_keyboard=True)
    kb.add(KeyboardButton("📜 Історія падінь"))
    kb.add(KeyboardButton("📷 Запросити фото"), KeyboardButton("📷 Запросити відео"))
    kb.add(KeyboardButton("📺 Live трансляція"))
    kb.add(KeyboardButton("👤 Змінити акаунт"), KeyboardButton("👤 Видалити акаунт"))

```

```

return kb

def _kb_stop_live():
    kb = InlineKeyboardMarkup()
    kb.add(InlineKeyboardButton("⏹ Зупинити трансляцію",
callback_data="live_stop"))
    return kb

def _kb_remove():
    return ReplyKeyboardRemove()

def _get_ward(guardian_row):

    phone = guardian_row[5]
    conn = db.get_conn()
    try:
        with conn.cursor() as cur:
            cur.execute("SELECT * FROM users WHERE guardian_phone = %s",
(phone,))

            return cur.fetchone()
    finally:
        db.release_conn(conn)

def start_guardian_flow(bot, message):

    tg_id = message.from_user.id
    _clear(tg_id)

    existing = db.get_guardian_by_tg_id(tg_id)
    if existing:
        _sess(tg_id)["guardian"] = existing
        _set(tg_id, G_MENU)
        bot.send_message(tg_id, f"👤 3 поверненням, {existing[3]}!",
reply_markup=_kb_menu())
    return

```

```

_set(tg_id, G_IDLE)
bot.send_message(tg_id, "Оберіть дію:", reply_markup=_kb_auth())

def _ask_reg_login(bot, tg_id: int):
    _set(tg_id, G_REG_LOGIN)
    bot.send_message(tg_id, "□ Реєстрація опікуна\n\nВведіть логін:",
reply_markup=_kb_remove())

def _ask_reg_pass(bot, tg_id: int):
    _set(tg_id, G_REG_PASS)
    bot.send_message(tg_id, "Введіть пароль (мін. 6 символів):")

def _ask_reg_name(bot, tg_id: int):
    _set(tg_id, G_REG_NAME)
    bot.send_message(tg_id, "Введіть ваше ім'я:")

def _ask_reg_email(bot, tg_id: int):
    _set(tg_id, G_REG_EMAIL)
    bot.send_message(tg_id, "Введіть електронну пошту:")

def _ask_reg_phone(bot, tg_id: int):
    _set(tg_id, G_REG_PHONE)
    bot.send_message(tg_id, "Введіть ваш номер телефону (наприклад
+380XXXXXXXXXX):\n"
    "□ Саме цей номер підопічний вказує при реєстрації.")

def _finish_registration(bot, tg_id: int):
    d = _sess(tg_id)["data"]
    ok, result = auth.register_guardian(
        login=d["login"],
        password=d["password"],
        name=d["name"],
        email=d["email"],
        phone=d["phone"],
        tg_id=tg_id

```

```

    )
    if not ok:
        bot.send_message(tg_id, f"✗ {result}\n\nСпробуйте ще раз.",
reply_markup=_kb_auth())
        _set(tg_id, G_IDLE)
        return

    guardian = db.get_guardian_by_tg_id(tg_id)
    _sess(tg_id)["guardian"] = guardian
    _set(tg_id, G_MENU)
    bot.send_message(tg_id, f"✓Рєєстрація успішна! Ласкаво просимо, {d['name']}!",
        reply_markup=_kb_menu())

def _ask_login(bot, tg_id: int):
    _set(tg_id, G_LOGIN)
    kb = ReplyKeyboardMarkup(resize_keyboard=True)
    kb.add(KeyboardButton("□ Назад"), KeyboardButton("□ Забув логін"))
    bot.send_message(tg_id, "□ Вхід\n\nВведіть логін:", reply_markup=kb)

def _ask_password(bot, tg_id: int):
    _set(tg_id, G_PASSWORD)
    kb = ReplyKeyboardMarkup(resize_keyboard=True)
    kb.add(KeyboardButton("□ Назад"), KeyboardButton("□ Забув пароль"))
    bot.send_message(tg_id, "Введіть пароль:", reply_markup=kb)

def _ask_forgot_email(bot, tg_id: int):
    _set(tg_id, G_FORGOT)
    kb = ReplyKeyboardMarkup(resize_keyboard=True)
    kb.add(KeyboardButton("□ Назад"))
    bot.send_message(tg_id,
"Введіть пошту якою ви реєструвались.\n"
"На неї надійде інструкція з відновлення.",
reply_markup=kb)

def _show_history(bot, tg_id: int):

```

```

guardian = _sess(tg_id)["guardian"]
if not guardian:
    return
rows = db.get_fall_history_by_guardian(guardian[0])
if not rows:
    bot.send_message(tg_id, "❑ Історія падінь підопічного порожня.",
reply_markup=_kb_menu())
    return

lines = ["❑ *Історія падінь підопічного:*"]
for i, (rid, user_name, detected_at, responded, response) in enumerate(rows, 1):
    dt = detected_at.strftime("%d.%m.%Y %H:%M")
    if responded is None:
        status = "❑ В обробці"
    elif responded and response == "yes":
        status = "✔Відповів: Все гаразд"
    elif responded and response == "no":
        status = "✗Відповів: Потрібна допомога"
    else:
        status = "❑ Без відповіді — сповіщено вас"
    lines.append(f"{i}. {dt} ({user_name}) — {status}")

lines.append(f"\nВсього: {len(rows)} подій")
bot.send_message(tg_id, "\n".join(lines), parse_mode="Markdown",
reply_markup=_kb_menu())

def _handle_photo(bot, tg_id: int):
    guardian = _sess(tg_id)["guardian"]
    ward = _get_ward(guardian)
    if not ward:
        bot.send_message(tg_id, "✗ Підопічного не знайдено. Переконайтесь що він
зареєструвався.")
        return

msg = bot.send_message(tg_id, "❑ Отримую знімок...")

```

```

def _do():
    try:
        photo_bytes = camera.get_photo(ward[7], ward[8])
        bot.delete_message(tg_id, msg.message_id)
        bot.send_photo(tg_id, photo_bytes, caption=f"□ Знімок з камери підопічного
{ward[3]}")
    except Exception as e:
        try:
            bot.edit_message_text(f"✗Помилка камери: {e}", tg_id, msg.message_id)
        except Exception:
            pass

import threading
threading.Thread(target=_do, daemon=True).start()

def _handle_video(bot, tg_id: int):
    guardian = _sess(tg_id)["guardian"]
    ward = _get_ward(guardian)
    if not ward:
        bot.send_message(tg_id, "✗Підопічного не знайдено.")
        return

    msg = bot.send_message(tg_id, "□ Записую відео (10 сек)...")

def _do():
    video_path = None
    try:
        video_path = camera.get_video(ward[7], ward[8], duration=10)
        bot.delete_message(tg_id, msg.message_id)
        with open(video_path, "rb") as f:
            bot.send_video(tg_id, f, caption=f"□ Відео з камери підопічного
{ward[3]}")
    except Exception as e:
        try:
            bot.edit_message_text(f"✗Помилка камери: {e}", tg_id, msg.message_id)

```

```

        except Exception:
            pass
    finally:
        if video_path and os.path.exists(video_path):
            os.remove(video_path)

import threading
threading.Thread(target=_do, daemon=True).start()

def _handle_live(bot, tg_id: int):
    guardian = _sess(tg_id)["guardian"]
    ward = _get_ward(guardian)
    if not ward:
        bot.send_message(tg_id, "❌ Підопічного не знайдено.")
        return

    bot.send_message(tg_id, "📺 Live трансляція розпочата.",
reply_markup=_kb_stop_live())

def _send_frame(jpeg_bytes: bytes):
    try:
        bot.send_photo(tg_id, jpeg_bytes)
    except Exception:
        pass

camera.start_live(ward[7], ward[8], _send_frame, fps=1)

def register_handlers(bot) -> None:

    @bot.message_handler(func=lambda m: m.text == "📺 Реєстрація" and
_step(m.from_user.id) == G_IDLE)
    def on_register(message):
        _ask_reg_login(bot, message.from_user.id)

```

```
@bot.message_handler(func=lambda m: m.text == "□ Вхід" and
_step(m.from_user.id) == G_IDLE)
```

```
def on_login(message):
    _ask_login(bot, message.from_user.id)
```

```
@bot.message_handler(func=lambda m: m.text == "□ Назад" and
_step(m.from_user.id) in (G_LOGIN, G_PASSWORD, G_FORGOT))
```

```
def on_back(message):
    tg_id = message.from_user.id
    _set(tg_id, G_IDLE)
    _sess(tg_id)["data"].clear()
    bot.send_message(tg_id, "Оберіть дію:", reply_markup=_kb_auth())
```

```
@bot.message_handler(func=lambda m: _step(m.from_user.id) == G_REG_LOGIN)
```

```
def reg_login(message):
    tg_id = message.from_user.id
    _put(tg_id, "login", message.text.strip())
    _ask_reg_pass(bot, tg_id)
```

```
@bot.message_handler(func=lambda m: _step(m.from_user.id) == G_REG_PASS)
```

```
def reg_pass(message):
    tg_id = message.from_user.id
    _put(tg_id, "password", message.text.strip())
    _ask_reg_name(bot, tg_id)
```

```
@bot.message_handler(func=lambda m: _step(m.from_user.id) == G_REG_NAME)
```

```
def reg_name(message):
    tg_id = message.from_user.id
    _put(tg_id, "name", message.text.strip())
    _ask_reg_email(bot, tg_id)
```

```
@bot.message_handler(func=lambda m: _step(m.from_user.id) == G_REG_EMAIL)
```

```
def reg_email(message):
    tg_id = message.from_user.id
    _put(tg_id, "email", message.text.strip())
```

```
_ask_reg_phone(bot, tg_id)
```

```
@bot.message_handler(func=lambda m: _step(m.from_user.id) ==
G_REG_PHONE)
```

```
def reg_phone(message):
    tg_id = message.from_user.id
    _put(tg_id, "phone", message.text.strip())
    _finish_registration(bot, tg_id)
```

```
@bot.message_handler(func=lambda m: _step(m.from_user.id) == G_LOGIN and
m.text not in ("⬅ Назад", "⬅ Забув логін"))
```

```
def login_step(message):
    tg_id = message.from_user.id
    _put(tg_id, "login", message.text.strip())
    _ask_password(bot, tg_id)
```

```
@bot.message_handler(func=lambda m: m.text == "⬅ Забув логін" and
_step(m.from_user.id) == G_LOGIN)
```

```
def forgot_login(message):
    _ask_forgot_email(bot, message.from_user.id)
```

```
@bot.message_handler(func=lambda m: _step(m.from_user.id) == G_PASSWORD
and m.text not in ("⬅ Назад", "⬅ Забув пароль"))
```

```
def password_step(message):
    tg_id = message.from_user.id
    login = _get(tg_id, "login")
    ok, result = auth.login_guardian(login, message.text.strip())
    if not ok:
        bot.send_message(tg_id, f"❌{result}")
        return
    guardian = result
    db.update_guardian_tg_id(guardian[0], tg_id)
    guardian = db.get_guardian_by_tg_id(tg_id)
    _sess(tg_id)["guardian"] = guardian
    _set(tg_id, G_MENU)
```

```

        bot.send_message(tg_id, f"✓ Вхід успішний! Вітаємо, {guardian[3]}!",
reply_markup=_kb_menu())

```

```

@bot.message_handler(func=lambda m: m.text == "□ Забув пароль" and
_step(m.from_user.id) == G_PASSWORD)

```

```

def forgot_password(message):
    _ask_forgot_email(bot, message.from_user.id)

```

```

@bot.message_handler(func=lambda m: _step(m.from_user.id) == G_FORGOT and
m.text != "□ Назад")

```

```

def forgot_email_step(message):
    tg_id = message.from_user.id
    ok, result = auth.restore_guardian_password(message.text.strip())
    if not ok:
        bot.send_message(tg_id, f"✗{result}")
        return
    bot.send_message(tg_id,
        "✓ Якщо акаунт з такою поштою існує — інструкція з відновлення
надіслана.\n"
        "(Функція відновлення поки в розробці)",
        reply_markup=_kb_auth())
    _set(tg_id, G_IDLE)

```

```

@bot.message_handler(func=lambda m: m.text == "□ Історія падінь" and
_step(m.from_user.id) == G_MENU)

```

```

def history(message):
    _show_history(bot, message.from_user.id)

```

```

@bot.message_handler(func=lambda m: m.text == "□ Запросити фото" and
_step(m.from_user.id) == G_MENU)

```

```

def photo(message):
    _handle_photo(bot, message.from_user.id)

```

```

@bot.message_handler(func=lambda m: m.text == "□ Запросити відео" and
_step(m.from_user.id) == G_MENU)

```

```

def video(message):
    _handle_video(bot, message.from_user.id)

    @bot.message_handler(func=lambda m: m.text == "□ Live трансляція" and
_step(m.from_user.id) == G_MENU)
    def live(message):
        _handle_live(bot, message.from_user.id)

    @bot.callback_query_handler(func=lambda c: c.data == "live_stop")
    def stop_live(call):
        camera.stop_live()
        try:
            bot.answer_callback_query(call.id)
        except Exception:
            pass
        try:
            if call.message:
                bot.edit_message_reply_markup(call.message.chat.id,
call.message.message_id, reply_markup=None)
            except Exception:
                pass
            tg_id = call.from_user.id
            bot.send_message(tg_id, "●Трансляцію зупинено.", reply_markup=_kb_menu())

    @bot.message_handler(func=lambda m: m.text == "□ Змінити акаунт" and
_step(m.from_user.id) == G_MENU)
    def switch_account(message):
        tg_id = message.from_user.id
        guardian = _sess(tg_id).get("guardian")
        if guardian:
            db.update_guardian_tg_id(guardian[0], None)
        camera.stop_live()
        _clear(tg_id)
        bot.send_message(tg_id, "□ Ви вийшли з акаунту.", reply_markup=_kb_auth())

```

```

@bot.message_handler(func=lambda m: m.text == "❑ Видалити акаунт" and
_step(m.from_user.id) == G_MENU)
def delete_account(message):
    tg_id = message.from_user.id
    guardian = _sess(tg_id).get("guardian")
    if guardian:
        camera.stop_live()
        db.delete_guardian(guardian[0])
        _clear(tg_id)
    bot.send_message(tg_id, "❑ Акаунт видалено.", reply_markup=_kb_remove())

```

A.6 Текстфайлыhandlers_user.py

```

import auth
import database as db
import camera
import notifications
from telebot.types import (
    ReplyKeyboardMarkup, KeyboardButton,
    ReplyKeyboardRemove
)

```

```

IDLE      = "idle"
U_LOGIN   = "u_login"
U_PASSWORD = "u_password"
U_FORGOT   = "u_forgot"
U_REG_LOGIN = "u_reg_login"
U_REG_PASS = "u_reg_pass"
U_REG_NAME = "u_reg_name"
U_REG_BIRTH = "u_reg_birth"
U_REG_EMAIL = "u_reg_email"
U_REG_GUARD = "u_reg_guard"
U_REG_CAMTYPE = "u_reg_camtype"
U_REG_CAMSRC = "u_reg_camsrc"
U_MENU     = "u_menu"

```

```

_sessions: dict[int, dict] = {}

def _sess(tg_id: int) -> dict:
    if tg_id not in _sessions:
        _sessions[tg_id] = {"step": IDLE, "data": {}, "user": None}
    return _sessions[tg_id]

def _step(tg_id: int) -> str:
    return _sessions.get(tg_id, {}).get("step", "")

def _set(tg_id: int, step: str) -> None:
    _sess(tg_id)["step"] = step

def _put(tg_id: int, key: str, val) -> None:
    _sess(tg_id)["data"][key] = val

def _get(tg_id: int, key: str):
    return _sess(tg_id)["data"].get(key)

def _clear(tg_id: int) -> None:
    _sessions.pop(tg_id, None)

def _kb_auth():
    kb = ReplyKeyboardMarkup(resize_keyboard=True)
    kb.add(KeyboardButton("👉 Реєстрація"), KeyboardButton("👉 Вхід"))
    return kb

def _kb_menu():
    kb = ReplyKeyboardMarkup(resize_keyboard=True)
    kb.add(KeyboardButton("👉 Історія падінь"))
    kb.add(KeyboardButton("👉 Змінити користувача"))
    kb.add(KeyboardButton("👉 Видалити акаунт"))
    return kb

def _kb_cam_type():

```

```

kb = ReplyKeyboardMarkup(resize_keyboard=True)
kb.add(KeyboardButton("📷 Webcam"))
kb.add(KeyboardButton("📷 IP-камера"))
kb.add(KeyboardButton("📷 Захоплення екрану"))
return kb

def _kb_forgot():
    kb = ReplyKeyboardMarkup(resize_keyboard=True)
    kb.add(KeyboardButton("🏠 Назад"))
    return kb

def _kb_remove():
    return ReplyKeyboardRemove()

def start_user_flow(bot, message):

    tg_id = message.from_user.id
    _clear(tg_id)

    existing = db.get_user_by_tg_id(tg_id)
    if existing:
        _sess(tg_id)["user"] = existing
    _set(tg_id, U_MENU)
    _start_detection(bot, tg_id, existing)
    bot.send_message(tg_id, f"👉 3 поверненням, {existing[3]}!",
reply_markup=_kb_menu())
    return

    _set(tg_id, IDLE)
    bot.send_message(tg_id, "Оберіть дію:", reply_markup=_kb_auth())

def _start_detection(bot, tg_id: int, user_row) -> None:
    user_id = user_row[0]
    cam_type = user_row[7]
    cam_source = user_row[8]

```

```
guard_phone = user_row[6]
```

```
def _on_fall():
```

```
    try:
```

```
        notifications.on_fall_detected(bot, tg_id, user_id, guard_phone)
```

```
    except Exception as e:
```

```
        try:
```

```
            bot.send_message(tg_id, f"Помилка сповіщення: {e}")
```

```
        except Exception:
```

```
            pass
```

```
try:
```

```
    camera.start_detection(user_id, cam_type, cam_source, _on_fall)
```

```
    bot.send_message(tg_id, "Детекція падінь активна. Камера запущена.")
```

```
except Exception as e:
```

```
    bot.send_message(tg_id, f"Не вдалось запустити камеру: {e}")
```

```
def _ask_reg_login(bot, tg_id: int):
```

```
    _set(tg_id, U_REG_LOGIN)
```

```
    bot.send_message(tg_id, "❑ Реєстрація\n\nВведіть логін:",
```

```
reply_markup=_kb_remove())
```

```
def _ask_reg_pass(bot, tg_id: int):
```

```
    _set(tg_id, U_REG_PASS)
```

```
    bot.send_message(tg_id, "Введіть пароль (мін. 6 символів):")
```

```
def _ask_reg_name(bot, tg_id: int):
```

```
    _set(tg_id, U_REG_NAME)
```

```
    bot.send_message(tg_id, "Введіть ваше ім'я:")
```

```
def _ask_reg_birth(bot, tg_id: int):
```

```
    _set(tg_id, U_REG_BIRTH)
```

```
    bot.send_message(tg_id, "Введіть дату народження (ДД.ММ.РРРР):")
```

```
def _ask_reg_email(bot, tg_id: int):
```

```

_set(tg_id, U_REG_EMAIL)
bot.send_message(tg_id, "Введіть вашу електронну пошту:")

def _ask_reg_guard(bot, tg_id: int):
    _set(tg_id, U_REG_GUARD)
    bot.send_message(tg_id, "Введіть номер телефону опікуна (наприклад
+380XXXXXXXXXX):")

def _ask_cam_type(bot, tg_id: int):
    _set(tg_id, U_REG_CAMTYPE)
    bot.send_message(tg_id, "Оберіть тип камери:", reply_markup=_kb_cam_type())

def _ask_cam_source(bot, tg_id: int):
    cam_type = _get(tg_id, "camera_type")
    _set(tg_id, U_REG_CAMSRC)
    if cam_type == "webcam":
        bot.send_message(tg_id, "Введіть індекс камери (0 — вбудована, 1 —
зовнішня):",
                        reply_markup=_kb_remove())
    elif cam_type == "ip":
        bot.send_message(tg_id, "Введіть URL потоку (наприклад
rtsp://192.168.1.1/stream):",
                        reply_markup=_kb_remove())
    else:
        _put(tg_id, "camera_source", "screen")
        _finish_registration(bot, tg_id)

def _finish_registration(bot, tg_id: int):
    d = _sess(tg_id)["data"]
    ok, result = auth.register_user(
        login=d["login"],
        password=d["password"],
        name=d["name"],
        birth_date_str=d["birth_date"],
        email=d["email"],

```

```

        guardian_phone=d["guardian_phone"],
        camera_type=d["camera_type"],
        camera_source=d["camera_source"],
        tg_id=tg_id
    )
    if not ok:
        bot.send_message(tg_id, f"✘ {result}\n\nСпробуйте ще раз.",
reply_markup=_kb_auth())
        _set(tg_id, IDLE)
        return

    user = db.get_user_by_tg_id(tg_id)
    _sess(tg_id)["user"] = user
    _set(tg_id, U_MENU)
    _start_detection(bot, tg_id, user)
    bot.send_message(tg_id, f"✔Рєєстрація успішна! Ласкаво просимо, {d['name']}!",
        reply_markup=_kb_menu())

def _ask_login(bot, tg_id: int):
    _set(tg_id, U_LOGIN)
    kb = ReplyKeyboardMarkup(resize_keyboard=True)
    kb.add(KeyboardButton("⏪ Назад"), KeyboardButton("⏩ Забув логін"))
    bot.send_message(tg_id, "⏪ Вхід\n\nВведіть логін:", reply_markup=kb)

def _ask_password(bot, tg_id: int):
    _set(tg_id, U_PASSWORD)
    kb = ReplyKeyboardMarkup(resize_keyboard=True)
    kb.add(KeyboardButton("⏪ Назад"), KeyboardButton("⏩ Забув пароль"))
    bot.send_message(tg_id, "Введіть пароль:", reply_markup=kb)

def _ask_forgot_email(bot, tg_id: int, context: str):
    _put(tg_id, "forgot_context", context)
    _set(tg_id, U_FORGOT)
    bot.send_message(tg_id,
"Введіть пошту якою ви реєструвались.\n"

```

```

"На неї надійде інструкція з відновлення.",
reply_markup=_kb_forgot())

def _show_history(bot, tg_id: int):
    user = _sess(tg_id)["user"]
    if not user:
        return
    rows = db.get_fall_history_by_user(user[0])
    if not rows:
        bot.send_message(tg_id, "❑ Історія падінь порожня.",
reply_markup=_kb_menu())
        return

    lines = ["❑ *Ваша історія падінь:*"]
    for i, (rid, detected_at, responded, response) in enumerate(rows, 1):
        dt = detected_at.strftime("%d.%m.%Y %H:%M")
        if responded is None:
            status = "❑ В обробці"
        elif responded and response == "yes":
            status = "✓ Все гаразд"
        elif responded and response == "no":
            status = "✗ Потрібна допомога"
        else:
            status = "❑ Без відповіді — сповіщено опікуна"
        lines.append(f"{i}. {dt} — {status}")

    lines.append(f"\nВсього: {len(rows)} подій")
    bot.send_message(tg_id, "\n".join(lines), parse_mode="Markdown",
reply_markup=_kb_menu())

def register_handlers(bot) -> None:

    @bot.message_handler(func=lambda m: m.text == "❑ Реєстрація" and
_step(m.from_user.id) == IDLE)
    def on_register(message):

```

```

        _ask_reg_login(bot, message.from_user.id)

    @bot.message_handler(func=lambda m: m.text == "❏ Вхід" and
        _step(m.from_user.id) == IDLE)
    def on_login(message):
        _ask_login(bot, message.from_user.id)

    @bot.message_handler(func=lambda m: m.text == "❏ Назад" and
        _step(m.from_user.id) in (U_LOGIN, U_PASSWORD, U_FORGOT))
    def on_back(message):
        tg_id = message.from_user.id
        _set(tg_id, IDLE)
        _sess(tg_id)["data"].clear()
        bot.send_message(tg_id, "Оберіть дію:", reply_markup=_kb_auth())

    @bot.message_handler(func=lambda m: _step(m.from_user.id) == U_REG_LOGIN)
    def reg_login(message):
        tg_id = message.from_user.id
        _put(tg_id, "login", message.text.strip())
        _ask_reg_pass(bot, tg_id)

    @bot.message_handler(func=lambda m: _step(m.from_user.id) == U_REG_PASS)
    def reg_pass(message):
        tg_id = message.from_user.id
        _put(tg_id, "password", message.text.strip())
        _ask_reg_name(bot, tg_id)

    @bot.message_handler(func=lambda m: _step(m.from_user.id) == U_REG_NAME)
    def reg_name(message):
        tg_id = message.from_user.id
        _put(tg_id, "name", message.text.strip())
        _ask_reg_birth(bot, tg_id)

    @bot.message_handler(func=lambda m: _step(m.from_user.id) == U_REG_BIRTH)
    def reg_birth(message):

```

```

tg_id = message.from_user.id
_put(tg_id, "birth_date", message.text.strip())
_ask_reg_email(bot, tg_id)

```

```

@bot.message_handler(func=lambda m: _step(m.from_user.id) == U_REG_EMAIL)

```

```

def reg_email(message):

```

```

    tg_id = message.from_user.id
    _put(tg_id, "email", message.text.strip())
    _ask_reg_guard(bot, tg_id)

```

```

@bot.message_handler(func=lambda m: _step(m.from_user.id) ==
U_REG_GUARD)

```

```

def reg_guard(message):

```

```

    tg_id = message.from_user.id
    _put(tg_id, "guardian_phone", message.text.strip())
    _ask_cam_type(bot, tg_id)

```

```

@bot.message_handler(func=lambda m: _step(m.from_user.id) ==
U_REG_CAMTYPE)

```

```

def reg_cam_type(message):

```

```

    tg_id = message.from_user.id
    text = message.text
    mapping = {
        "☐ Webcam": "webcam",
        "☐ IP-камера": "ip",
        "☐ Захоплення екрану": "screen"
    }

```

```

    cam_type = mapping.get(text)

```

```

    if not cam_type:

```

```

        bot.send_message(tg_id, "Будь ласка, оберіть варіант з клавіатури.")
        return

```

```

    _put(tg_id, "camera_type", cam_type)

```

```

    _ask_cam_source(bot, tg_id)

```

```

        @bot.message_handler(func=lambda m: _step(m.from_user.id) ==
U_REG_CAMSRC)
    def reg_cam_source(message):
        tg_id = message.from_user.id
        _put(tg_id, "camera_source", message.text.strip())
        _finish_registration(bot, tg_id)

    @bot.message_handler(func=lambda m: _step(m.from_user.id) == U_LOGIN and
m.text not in ("⬅ Назад", "⬅ Забув логін"))
    def login_step(message):
        tg_id = message.from_user.id
        _put(tg_id, "login", message.text.strip())
        _ask_password(bot, tg_id)

    @bot.message_handler(func=lambda m: m.text == "⬅ Забув логін" and
_step(m.from_user.id) == U_LOGIN)
    def forgot_login(message):
        _ask_forgot_email(bot, message.from_user.id, "login")

    @bot.message_handler(func=lambda m: _step(m.from_user.id) == U_PASSWORD
and m.text not in ("⬅ Назад", "⬅ Забув пароль"))
    def password_step(message):
        tg_id = message.from_user.id
        login = _get(tg_id, "login")
        ok, result = auth.login_user(login, message.text.strip())
        if not ok:
            bot.send_message(tg_id, f"❌{result}")
            return
        user = result
        db.update_user_tg_id(user[0], tg_id)
        user = db.get_user_by_tg_id(tg_id)
        _sess(tg_id)["user"] = user
        _set(tg_id, U_MENU)
        _start_detection(bot, tg_id, user)

```

```

        bot.send_message(tg_id, f"✔ Вхід успішний! Вітаємо, {user[3]}!",
reply_markup=_kb_menu())

```

```

@bot.message_handler(func=lambda m: m.text == "□ Забув пароль" and
_step(m.from_user.id) == U_PASSWORD)

```

```

def forgot_password(message):
    _ask_forgot_email(bot, message.from_user.id, "password")

```

```

@bot.message_handler(func=lambda m: _step(m.from_user.id) == U_FORGOT and
m.text != "□ Назад")

```

```

def forgot_email_step(message):
    tg_id = message.from_user.id
    ok, result = auth.restore_user_password(message.text.strip())
    if not ok:
        bot.send_message(tg_id, f"✗{result}")
        return
    bot.send_message(tg_id,

```

```

        f"✔ Якщо акаунт з такою поштою існує — інструкція з відновлення
надіслана.\n"

```

```

        "(Функція відновлення поки в розробці)",

```

```

reply_markup=_kb_auth())
    _set(tg_id, IDLE)

```

```

@bot.message_handler(func=lambda m: m.text == "□ Історія падінь" and
_step(m.from_user.id) == U_MENU)

```

```

def history(message):
    _show_history(bot, message.from_user.id)

```

```

@bot.message_handler(func=lambda m: m.text == "□ Змінити користувача" and
_step(m.from_user.id) == U_MENU)

```

```

def switch_user(message):
    tg_id = message.from_user.id
    user = _sess(tg_id).get("user")
    if user:
        camera.stop_detection(user[0])

```

```

        db.update_user_tg_id(user[0], None)
    _clear(tg_id)
    bot.send_message(tg_id, "❑ Ви вийшли з акаунту.", reply_markup=_kb_auth())

```

```

@bot.message_handler(func=lambda m: m.text == "❑ Видалити акаунт" and
_step(m.from_user.id) == U_MENU)
def delete_account(message):
    tg_id = message.from_user.id
    user = _sess(tg_id).get("user")
    if user:
        camera.stop_detection(user[0])
        db.delete_user(user[0])
    _clear(tg_id)
    bot.send_message(tg_id, "❑ Акаунт видалено.", reply_markup=_kb_remove())

```

A.7 Текст файлу notifications.py

```

import threading
import database as db
from telebot.types import InlineKeyboardMarkup, InlineKeyboardButton

```

```

RESPONSE_TIMEOUT = 60

```

```

_pending: dict[int, dict] = {}

```

```

def _guardian_tg_id(guardian) -> int | None:
    return guardian[6] if guardian else None

```

```

def _notify_guardian(bot, guardian, user_name: str) -> None:
    tg_id = _guardian_tg_id(guardian)
    if not tg_id:
        return
    bot.send_message(

```

```

        tg_id,
        "Увага!\n" + f"Підопічний {user_name} можливо впав і не відповідає.\nБудь
ласка, зв'яжіться з ним негайно."
    )

```

```

def _cancel_pending(user_tg_id: int) -> None:
    entry = _pending.pop(user_tg_id, None)
    if entry and not entry["answered"]:
        entry["timer"].cancel()

def on_fall_detected(bot, user_tg_id: int, user_id: int, guardian_phone: str) -> None:
    if user_tg_id in _pending:
        return

    guardian = db.get_guardian_by_phone(guardian_phone)
    guardian_id = guardian[0] if guardian else None
    alert_id = db.log_fall(user_id, guardian_id)

    markup = InlineKeyboardMarkup()
    markup.add(
        InlineKeyboardButton("Так, все гаразд", callback_data=f"fall_ok:{alert_id}"),
        InlineKeyboardButton("Hi,                потрібна                допомога",
callback_data=f"fall_no:{alert_id}")
    )
    bot.send_message(
        user_tg_id,
        "Зафіксовано можливе падіння!\n\nЗ вами все гаразд?",
        reply_markup=markup
    )

    user = db.get_user_by_tg_id(user_tg_id)
    user_name = user[3] if user else "Невідомий"

def _on_timeout():
    entry = _pending.pop(user_tg_id, None)

```

```

    if not entry or entry["answered"]:
        return
    db.update_fall_response(alert_id, responded=False, response=None)
    _notify_guardian(bot, guardian, user_name)

timer = threading.Timer(RESPONSE_TIMEOUT, _on_timeout)
timer.start()

_pending[user_tg_id] = {
    "alert_id": alert_id,
    "timer": timer,
    "answered": False,
    "guardian": guardian,
    "user_name": user_name,
}

def handle_fall_response(bot, user_tg_id: int, alert_id: int, response: str) -> None:
    entry = _pending.get(user_tg_id)
    if not entry or entry["alert_id"] != alert_id or entry["answered"]:
        return

    entry["answered"] = True
    entry["timer"].cancel()
    _pending.pop(user_tg_id, None)

    if response == "ok":
        db.update_fall_response(alert_id, responded=True, response="yes")
        bot.send_message(user_tg_id, "Добре! Продовжуємо моніторинг.")
    else:
        db.update_fall_response(alert_id, responded=True, response="no")
        bot.send_message(user_tg_id, "Викликаємо допомогу. Тримайтеся!")
        _notify_guardian(bot, entry["guardian"], entry["user_name"])

```

```
def has_pending_alert(user_tg_id: int) -> bool:  
    return user_tg_id in _pending
```

A.8 Текст файла requirements.txt

```
pyTelegramBotAPI==4.21.0
```

```
psycpg2-binary==2.9.9
```

```
bcrypt==4.1.3
```

```
opencv-python==4.10.0.84
```

```
ultralytics==8.2.87
```

```
mss==9.0.2
```

```
numpy==1.26.4
```

```
python-dotenv==1.0.1
```

ДОДАТОК Б
Слайди презентації



Рисунок Б.1 – Слайд 1

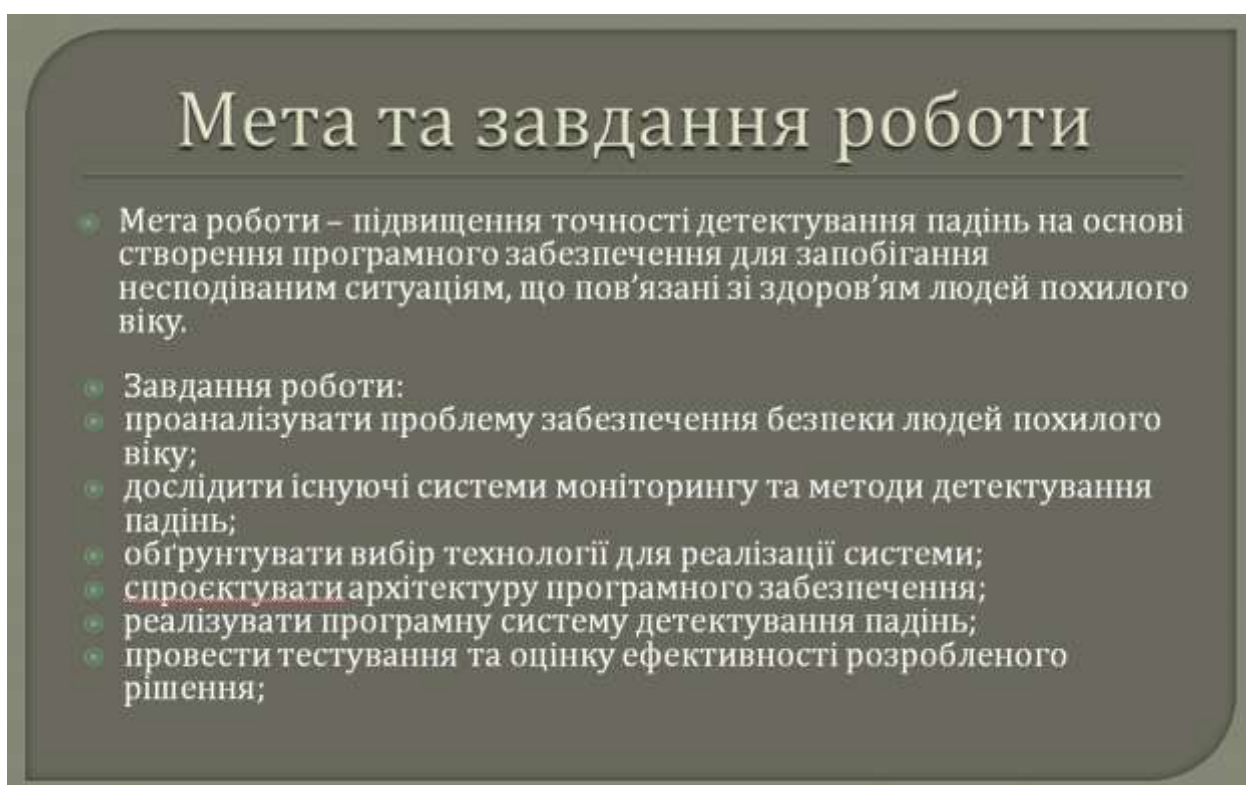


Рисунок Б.2 – Слайд 2

Порівняльний аналіз методів детектування падінь

Метод	Принцип роботи	Переваги	Недоліки	Точність
Порогові методи	Аналіз різких змін прискорення, порівняння з порогом	Простота реалізації, низькі обчислювальні витрати	Висока кількість хибних спрацювань, складність вибору порогу	Низька / середня
Методи машинного навчання	Класифікація на основі ознак сенсорних даних	Висока точність, адаптивність до умов	Потреба у великих даних, висока обчислювальна складність	Висока
Комп'ютерний зір	Аналіз відео, поза силуети людини	Висока точність, не потребує носимих пристроїв	Залежність від освітлення, проблеми конфіденційності	Висока
Гібридні методи	Комбінація сенсорів + відео/ML	Найвища надійність зменшення помилок	Складність реалізації, висока вартість	Дуже висока

Рисунок Б.3 – Слайд 3

Функціональні вимоги

Назва вимоги	Опис вимоги	Пріоритет
Реєстрація користувача	Система повинна забезпечувати введення персональних даних користувача та контактів довіреної особи	Високий
Детектування падіння	Автоматичне визначення падіння на основі даних з камери відео спостереження	Високий
Надсилання сповіщення	Надсилання тривожного повідомлення через Telegram у разі небезпечної ситуації	Високий
Історія повідомлень	Збереження попередніх тривожних сповіщень для подальшого перегляду	Середній

Рисунок Б.4 – Слайд 4

Нефункціональні вимоги

Назва вимоги	Опис вимоги	Пріоритет
Швидкодія	Повідомлення повинно надсилатися без значної затримки	Високий
Надійність	Система повинна працювати стабільно та безперервно	Високий
Простота прискорення	Інтерфейс Telegram – бота має бути зрозумілим для людей похилого віку	Високий
Захист даних	Персональні дані користувача повинні бути захищені	Високий
Модернізація	Можливість подальшого розширення функціоналу системи	Середній

Рисунок Б.5 – Слайд 5

Порівняльний аналіз засобів взаємодії

Критерій	Telegram - бот	Веб - додаток	Мобільний додаток
Вартість розробки	Низька	Середня	Висока
Швидкість розробки	Висока	Середня	Низька
Миттєві сповіщення	Так	Обмежено	Так
Необхідність встановлення	Ні (лише Telegram)	Ні	Так
Перегляд відео з камер	Так	Так	Так
Зручність користування	Висока	Середня	Висока
Підтримка команд користувача	Так	Так	Так
Доступність на різних пристроях	Висока	Висока	Середня

Рисунок Б.6 – Слайд 6

Обґрунтування вибору нейромережі

Критерій	YOLO	Faster R-CNN	SSD	Haar Cascade
Швидкість роботи	Дуже висока	Низька	Висока	Дуже висока
Робота в реальному часі	Так	Ні	Так	Так
Точність <u>детекції</u>	Висока	Дуже висока	Висока	Низька
Виявлення кількох об'єктів	Так	Так	Так	Обмежено
Стійкість до змін освітлення	Висока	Висока	Середня	Низька
Використання нейронних мереж	Так	Так	Так	Ні
Обчислювальні вимоги	Середня	Висока	Середня	Низька
Придатність для систем моніторингу в реальному часі	Дуже висока	Середня	Висока	Низька

Рисунок Б.7 – Слайд 7

Обґрунтування вибору бібліотеки для обробки зображень та відео

Критерії	OpenCV	Pillow(PIL)	Scikit-image	ImageIO
Робота з відео в реальному часі	Так	Ні	Обмежено	Обмежено
Захоплення відео з камери	Так	Ні	Ні	Так
Обробка зображень	Так	Так	Так	Частково
Комп'ютерний зір	Так	Ні	Частково	Ні
Інтеграція з моделями ШІ	Висока	Низька	Середня	Низька
Продуктивність	Висока	Середня	Середня	Середня
Підтримка <u>відеопотоків</u>	Так	Ні	Ні	Частково
Зручність для системи моніторингу	Висока	Низька	Середня	Низька

Рисунок Б.8– Слайд 8

Обґрунтування вибору СКБД

Критерій	PostgreSQL	MySQL	SQLite	MongoDB
Тип СБД	Реляційна (клієнт - серверна)	Реляційна	Вбудована	NoSQL (документна)
Продуктивність	Висока	Висока	Висока (для малих даних)	Висока
Масштабованість	Дуже висока	Висока	Низька	Дуже висока
Підтримка складних запитів	Дуже розвинена	Добра	Обмежена	Обмежена (інший підхід)
Транзакція (ACID)	Повна підтримка	Повна підтримка	Повна підтримка	Часткова
Робота з великими даними	Дуже ефективна	Ефективна	Обмежена	Дуже ефективна
Гнучкість структури даних	Середня	Середня	Низька	Дуже висока
Складність налаштування	Середня	Середня	Дуже низька	Середня
Підтримка багато користувачів	Повна	Повна	Обмежена	Повна
Використання у системах моніторингу	Дуже підходить	Підходить	Для простих систем	Підходить для аналітики

Рисунок Б.9 – Слайд 9

Сценарій взаємодії користувача із системою



Рисунок Б.10 – Слайд 10



Рисунок Б.11 – Слайд 11



Рисунок Б.12 – Слайд 12

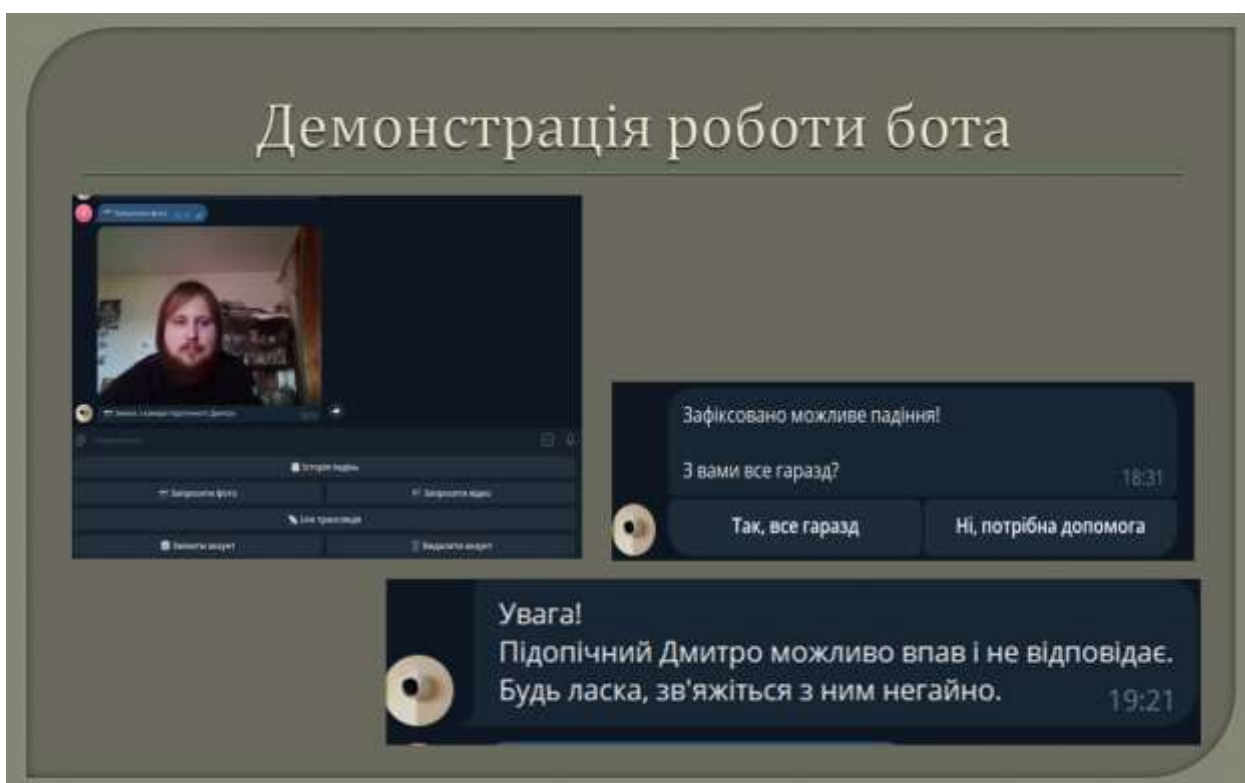


Рисунок Б.13 – Слайд 13

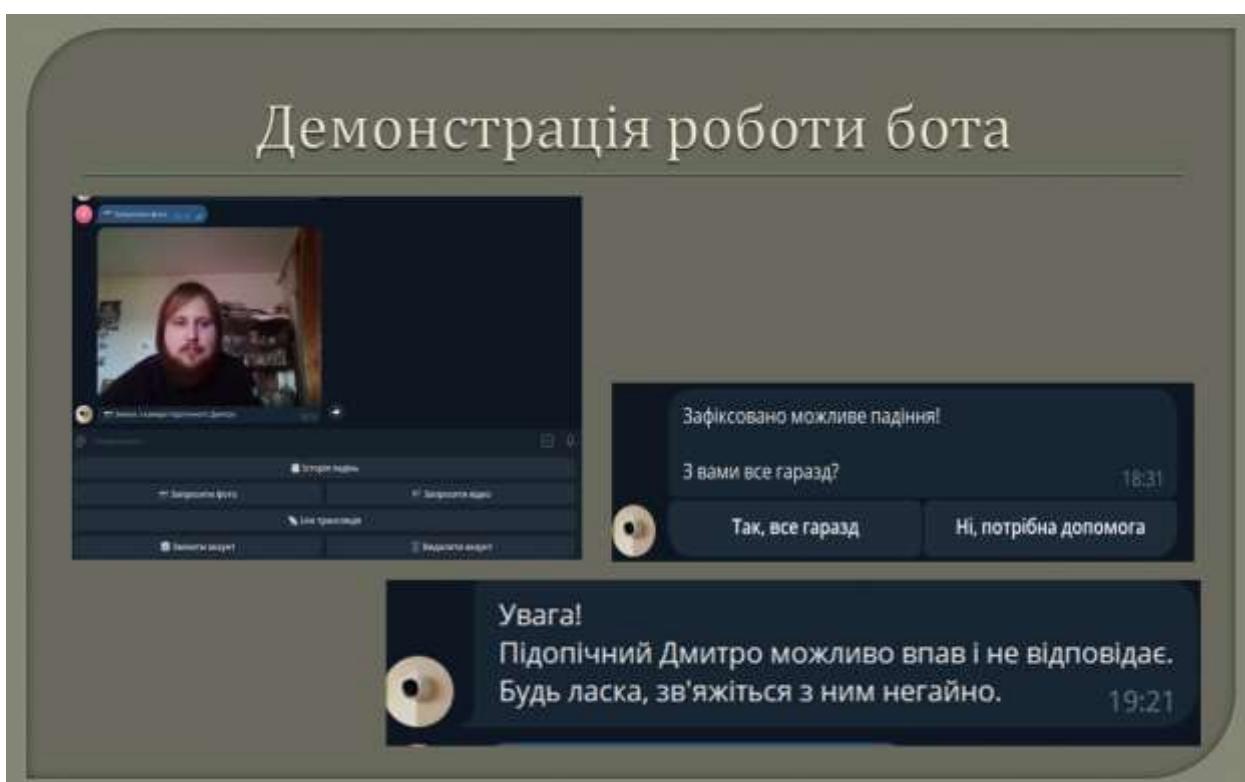


Рисунок Б.14 – Слайд 14

Висновки

- Проаналізовано існуючі методи виявлення падінь та їх недоліки.
- Сформульовано вимоги до ПЗ підсистеми та обґрунтовано вибір інструментарію розробки (мова Python, платформа Visual Studio Code, бібліотека OpenCVpython, бібліотека psycorg2, нейромережа YOLO).
- Спроектовано Telegram-бот з інтерфейсом користувача для нагляду, та інтерфейс опікуна, який наглядає за користувачем.
- Розроблено алгоритм фільтрації хибних спрацювань на основі ігнорування хибних падінь для уникнення складних ситуацій, а також реалізовано незалежні мікросервіси для перевірки опікуном стану користувача.
- Експериментальне дослідження підтвердило, що розроблена підсистема дозволяє врятувати більше життів людей похилого віку з більшою точністю до детектування падінь з 50% (акселерометри та гіроскопи) до 100%.

Рисунок Б.15 – Слайд 15